

Mémoire pour l'obtention de l'Habilitation à Diriger des Recherches

Sorbonne Université

Spécialité

Sciences pour l'Ingénieur

Présentée par

Sonia Belaïd

Experte Cryptologue

CryptoExperts

Design and Analysis of Secure Cryptographic Implementations in the Random Probing Model

soutenue le 9 février 2026

devant le jury composé de :

Yuval Ishai	Professeur à Technion	Rapporteur
Guénaél Renault	Chercheur à l'ANSSI	Rapporteur
Peter Schwabe	Directeur scientifique au MPI-SP	Rapporteur
Christina Boura	Professeure à l'IRIF, Université Paris Cité	Examinatrice
Pierre-Alain Fouque	Professeur à l'Université de Rennes 1	Examineur
Maria Naya-Plasencia	Directrice de recherche DR1 à l'Inria	Examinatrice
Emmanuel Prouff	Chercheur à Sorbonne Université	Examineur
Matthieu Rivain	Expert cryptologue à CryptoExperts	Examineur

À Mathis, Elyes et Clément.

Remerciements

Je souhaite tout d'abord exprimer ma profonde gratitude aux membres de mon jury pour l'honneur qu'ils me font en acceptant d'évaluer cette Habilitation. Je remercie tout particulièrement Yuval Ishai, Guénaél Renault et Peter Schwabe d'avoir rapporté ce manuscrit, ainsi que Christina Boura, Pierre-Alain Fouque, Maria Naya-Plasencia, Emmanuel Prouff et Matthieu Rivain d'en être examinateurs.

Au-delà de leur participation au jury, je tiens à remercier très sincèrement ces personnes pour ce qu'elles ont apporté à mon parcours. Je remercie tout spécialement Yuval, dont les travaux fondateurs sur les attaques par canaux auxiliaires ont profondément inspiré une grande partie de ce manuscrit. Je remercie chaleureusement Guénaél pour ses précieux conseils dans la préparation de cette habilitation, et pour m'avoir encouragée à avancer avec confiance. Je remercie également Peter, dont le soutien m'a accompagnée dans plusieurs responsabilités scientifiques, et qui a aussi contribué à des opportunités marquantes, dont cette invitation à Santiago. Je remercie Christina, pour ses encouragements de tous les instants et pour l'inspiration scientifique qu'elle représente ; son soutien a compté à de nombreuses étapes. Je remercie ensuite Pierre-Alain, qui a joué un rôle essentiel dans mon parcours scientifique ; avec Michel Abdalla, je leur suis très reconnaissante pour leur confiance, leurs conseils et l'exigence intellectuelle qu'ils m'ont transmise pendant ma thèse. Je voudrais aussi adresser un remerciement tout particulier à Maria, qui me suit depuis ma thèse et m'a toujours encouragée dans mes projets ; elle a été, à bien des égards, le déclencheur de cette habilitation. Je tiens également à remercier Emmanuel, qui m'accompagne depuis presque mes débuts en recherche et dont le soutien a été déterminant à de nombreuses étapes, jusqu'aux premiers travaux ayant nourri ce mémoire. Enfin, je remercie très sincèrement Matthieu : il m'a encouragée, soutenue et fait confiance, tout en travaillant avec moi sur l'ensemble des sujets qui constituent ce mémoire, et bien d'autres. Son accompagnement constant, sa générosité intellectuelle et son exigence ont été des moteurs précieux pour mener ce travail à terme.

Je remercie chaleureusement l'ensemble de mes anciens collègues, d'abord chez Oberthur, puis chez Thales et à l'ENS, où j'ai passé des années particulièrement enrichissantes, tant sur le plan humain que scientifique.

Je souhaite ensuite exprimer ma profonde reconnaissance à Matthieu, Louis Goubin et Pascal Paillier, qui m'ont fait confiance en me permettant d'intégrer CryptoExperts. J'ai toujours admiré la qualité et l'impact de leurs travaux, bien avant mon arrivée, et je mesure chaque jour la chance d'évoluer à leurs côtés. Ils m'ont offert un cadre de travail profondément humain, dans lequel j'ai pu grandir et m'épanouir. C'est un honneur de travailler avec eux.

Je remercie très sincèrement tous mes collègues de CryptoExperts, qui rendent ce cadre de travail aussi stimulant que bienveillant, et dont les échanges ont nourri de nombreuses idées présentées dans ce manuscrit : Pascal, Matthieu, Louis, Ryad Benadjila, Thibault Feneuil, Mélissa Rossi, Gaëtan Cassiers, Gabriel Zaid, Nicolas Bon, Victor Normand, Auguste Warmé-Janville, Ghizlen Boukacem, et les anciens : Abdel Rahman Taleb, Aleksei Udovenko, Dahmun Goudarzi, Darius Mercadier, Junwei Wang et Viet Sang Nguyen. Je remercie également nos colocataires puis voisins, en particulier Damien Ligier, Ilaria Chilotti, Jean-Baptiste Orfila et Samuel Tap pour les moments partagés.

Je tiens ensuite à remercier chaleureusement Mélissa, Abdel, Nicolas, Victor, Noémie Akpaki, Ghizlen, Gaëtan et Gabriel, que j'ai eu (ou que j'ai encore) le plaisir d'encadrer en thèse ou d'accompagner en post-doctorat, pour leur engagement, leur curiosité et leur enthousiasme. Une part importante des résultats et des développements présentés dans ce mémoire a été rendue possible grâce à leurs contributions.

Je remercie évidemment l'ensemble de mes co-auteurs et co-autrices, avec qui j'ai eu la chance de partager de nombreux projets. Au-delà des résultats, je leur suis reconnaissante pour la bienveillance et la qualité des discussions qui rendent la recherche si vivante.

Je souhaite enfin adresser quelques remerciements particuliers à celles et ceux qui ont marqué mon parcours de recherche. Je remercie tout d'abord Jean-Sébastien Coron, avec qui j'ai eu la chance de collaborer, pour sa bienveillance, son soutien et son aide, plus que précieuse, dans les projets que j'ai entrepris. Je remercie également Gilles Barthe, Benjamin Grégoire, François Dupressoir et Pierre-Yves Strub, dont les échanges et l'influence ont marqué un tournant dans mes travaux. Je pense aussi à Thomas Roche et Victor Lomné, *les ninjas*, que je prends toujours un grand plaisir à retrouver et avec qui j'apprécie particulièrement collaborer. Un merci tout particulier également à David Pointcheval et Damien Vergnaud pour nos collaborations, co-encadrements et leur aide administrative précieuse. Merci également à François-Xavier Standaert pour nos nombreux échanges et nos collaborations. Je remercie enfin les membres de l'équipe Cosmiq et de l'ANSSI avec qui j'ai un grand plaisir à travailler.

Dans un esprit plus personnel, je remercie mes amis qui m'ont soutenue et encouragée. En particulier, je remercie Damien et Maxime, qui ont toujours pris le temps de m'écouter et de me conseiller.

Je réserve mes derniers remerciements à ma famille, qui a été mon socle tout au long de ce parcours.

Je remercie mes beaux-parents, ainsi que mon beau-frère Benoît et tout particulièrement ma belle-sœur Margaux. Toujours présente, elle nous a aidés à de nombreuses reprises et m'a permis de préserver du temps de travail lorsque la vie de famille s'invitait au premier plan, avec son lot d'imprévus.

Je remercie ma sœur Nadia, mon beau-frère Ahmed, et mon petit Samy, pour leur affection et leur soutien permanent. À Nadia, je dois tout particulièrement une présence constante, une fidélité et une force qui m'accompagnent autant dans mon parcours scientifique que dans ma vie personnelle.

Je remercie profondément mes parents, qui ont toujours été une source d'inspiration et dont le soutien n'a jamais failli ; c'est en partie pour eux que je franchis aujourd'hui cette étape. Au-delà de l'exemple qu'ils m'ont donné et de la confiance qu'ils m'ont transmise, ils sont présents à chaque moment important, dans ma vie professionnelle comme dans ma vie familiale, avec une disponibilité et une bienveillance constantes.

Enfin, je remercie Clément pour son soutien indéfectible et pour tout ce que nous construisons ensemble au quotidien. Merci pour ses conseils et sa patience, parfois jusque dans mes formules de probas, mais surtout pour l'équilibre, la force et le bonheur qu'il m'apporte. Je remercie aussi nos deux merveilleux petits garçons, Mathis et Elyes, pour leur énergie, leur joie de vivre et leur amour, jusqu'aux étoiles, aller-retour, fois l'infini.

Contents

I. Introduction	0
1. Introduction	1
1.1. Motivation	1
1.2. Contributions	2
2. Preliminaries	3
2.1. Notations	3
2.2. (Randomized) Arithmetic Circuits	3
2.3. Masking	5
2.4. Leakage Models	7
2.4.1. Noisy Leakage Model	7
2.4.2. Probing Model	8
2.4.3. Random Probing Model and Reduction	9
II. Random Probing Security	11
3. Random Probing Security	12
3.1. Random Probing Leakage and Security Notion	12
3.2. Simulation Failure Probability	13
3.3. Automatic Verification Tools	14
4. From Random Probing Security to Practical Security	16
4.1. Physical Assumptions	16
4.2. Our Methodology	17
4.3. Design Goals and Remaining Challenges	23
III. Random Probing Composability	25
5. Threshold Random Probing Composability	26
5.1. Motivation	26
5.2. Security Notion	27
5.3. Composition Results	28
6. (Uniformly) Cardinal Random Probing Composability	29
6.1. Motivation	29
6.2. Security Notions and Composition Results	31
6.2.1. Cardinal Random Probing Composability	31
6.2.2. Uniformly Cardinal Random Probing Composability	33
6.3. Comparison of Random Probing Composability Notions	36

IV. Design of Random Probing Composable Gadgets	39
7. Threshold RPC Gadgets from RPE Framework	40
7.1. RPE Framework	40
7.2. RPE Gadgets and Application to AES	43
8. Cardinal RPC Gadgets from Random Probing Partitioned Refresh	45
8.1. Random Probing Partitioned Refresh Gadget	45
8.2. Linear Gadgets	50
8.3. Non-Linear Multiplication	51
8.4. Compiler and Application to AES	54
V. Conclusion	57
Bibliography	60
VI. Appended publications	65
A. Random Probing Security: Verification, Composition, Expansion and New Constructions	66
B. On the Power of Expansion: More Efficient Constructions in the Random Probing Model	107
C. Dynamic Random Probing Expansion with Quasi Linear Asymptotic Complexity	166
D. IronMask: Versatile Verification of Masking Security	224
E. Formal Definition and Verification for Combined Random Fault and Random Probing Security	260
F. New Techniques for Random Probing Security and Application to Raccoon Signature Scheme	316
G. SoK: A Methodology to Achieve Provable Side-Channel Security in Real-World Implementations	359
H. Masked Circuit Compiler in the Cardinal Random Probing Composability Framework	403

Part I.

Introduction

Chapter 1

Introduction

Contents

1.1. Motivation	1
1.2. Contributions	2

1.1. Motivation

Cryptography is everywhere in our daily life to ensure the confidentiality and authentication of our communications and the integrity of our records. While cryptographic schemes are designed to be secure against black-box adversaries whose knowledge is limited to observing a few inputs and outputs, their implementations face significant additional challenges. In particular, they become vulnerable to powerful side-channel attacks, which exploit physical leakages (*e.g.* execution time, power consumption) emitted by the device to recover the secrets being manipulated. Using relatively cheap equipment, side-channel attacks can cause severe damage, such as full key recovery, against unprotected implementations in a matter of seconds [KJJ99].

The community thus had to adapt quickly by developing efficient countermeasures. The most widely deployed approach to mitigate side-channel attacks was independently introduced in 1999 by Chari et al. [CJRR99] and by Goubin and Patarin [GP99], and is now known as masking. The core idea is to split each sensitive variable x in the implementation into n shares, such that any subset of $n - 1$ shares is independent of x . As a result, an adversary must recover all n shares to reconstruct x . While the leakage is generally modeled as a noisy function of the manipulated variables, the masking countermeasure significantly increases the difficulty of an attack when the shares leak independently from one another. In this setting, the complexity of combining information from individual leakages grows exponentially with n [CJRR99].

Once all the sensitive variables are split into n shares, computations must be performed directly on these shares. Traditional gates implementing basic operations are thus replaced by specialized circuits called gadgets, which operate on masked variables. For both Boolean masking (where shares are combined using bitwise addition) and arithmetic masking (where shares are combined by a modular addition), linear gadgets are typically straightforward to implement, as the operations can be performed independently on each share. However, non-linear gadgets are more challenging because the shares cannot be manipulated separately. They require additional fresh randomness to ensure that the leakage of individual operations does not reveal multiple shares simultaneously.

This challenge is formally captured by the notion of $(n - 1)$ -probing security, introduced by Ishai, Sahai, and Wagner [ISW03], which defines security against an adversary capable of observing the exact values of up to $n - 1$ internal variables within the circuit. This probing model is particularly useful because it abstracts the physical leakage into a simpler framework where security proofs are simple to build. It also aligns with the result of Chari et al.'s that combining information from multiple independent leakages becomes exponentially more difficult as the number of shares increases. However,

the probing model does not fully capture advanced side-channel attacks, such as horizontal attacks, which exploit repeated manipulations of the same variables across different operations.

A more suitable framework for capturing such attacks is the noisy leakage model, introduced by Chari et al. [CJRR99] and later formalized by Prouff and Rivain [PR13]. In this model, the attacker observes noisy functions of all intermediate variables, effectively modeling power and electromagnetic side-channel emissions. Although it better reflects the reality of embedded devices, it poses significant challenges for constructing security proofs. Consequently, masking schemes are still primarily analyzed within the probing model, relying on the reduction established by Duc, Dziembowski, and Faust [DDF14], which shows how security in the noisy leakage model can be derived from security in the probing model, albeit with significant loss in tightness.

This reduction introduced an intermediate leakage model known as the random probing model, to which security in the noisy leakage model tightly reduces. In the random probing model, each intermediate variable leaks independently with a fixed probability p , also referred to as leakage rate. A circuit is considered secure in this model if the probability that the leaked wires reveal information about the secrets remains negligible. Because security proofs in the random probing model seem to be more tractable than in the noisy leakage model, it appears as the best compromise for establishing the theoretical security of masking schemes.

While some schemes already offered random probing security, several of them were designed with a leakage rate p that decreases with the number of shares [ISW03, DDF14, GJR18]. Only a few relied on a constant leakage rate [Ajt11, ADF16, AIS18] (which better reflects the device’s noise level) and among them only the approach of Ananth, Ishai, and Sahai [AIS18] made it explicit.

1.2. Contributions

In this manuscript, I present the contributions I have made, in close collaboration with my academic and industrial co-authors, to the security of masking schemes in the random probing model. After presenting the necessary background in Chapter 2, I dedicate Part II to the random probing security. This part includes our formalization of the concept, our practical methodology for evaluating the random probing security of small circuits supported by the development of optimized automated tools, and a comprehensive approach for transforming random probing secure implementations into physically secure implementations that resist side-channel attacks on microcontrollers.

While directly proving random probing security for small circuits is feasible with existing techniques, scaling these proofs to more complex circuits, such as cryptographic primitives, quickly becomes impractical. Indeed, constructing security proofs manually is impossible with current theoretical methods, and automated verification tools quickly face prohibitive complexity. To overcome this challenge, it is necessary to decompose large circuits into smaller blocks that satisfy specific composability properties. This modular approach makes it possible to achieve global random probing security by securely combining these blocks according to well-defined composition rules.

Accordingly, Part III presents the composition frameworks we have developed to achieve random probing security. We begin with the basic notion of threshold random probing composability and then introduce the stronger notions of (uniformly) cardinal random probing composability, which allow for tighter security bounds. Finally, we compare these simulation-based approaches with another framework from the literature, which we refer to as general random probing composability.

The final part of this manuscript focuses on the design of gadgets that satisfy our new composition notions, in order to build circuits that are globally random probing secure. The first strategy we explore allows for the construction of gadgets with arbitrarily high security levels, but at a significant cost and with security bounds that are not particularly tight. The second strategy, tailored for tighter composition approaches, does not yet offer analytical formulas but enables the construction of gadgets with a measurable level of security at a lower cost. Finally, we discuss applications of these techniques to the standard AES, before outlining future research directions.

Chapter 2

Preliminaries

Contents

2.1. Notations	3
2.2. (Randomized) Arithmetic Circuits	3
2.3. Masking	5
2.4. Leakage Models	7
2.4.1. Noisy Leakage Model	7
2.4.2. Probing Model	8
2.4.3. Random Probing Model and Reduction	9

2.1. Notations

Along the manuscript, $\mathbb{K}$ shall denote a finite field or a finite ring. For any $n \in \mathbb{N}$, we shall denote $[n]$ the integer set $[n] = [1, n] \cap \mathbb{Z}$. We shall further denote $\mathcal{P}([n]) = \{S \subseteq [n]\}$ the power set of $[n]$ and $\mathcal{S}_n$ the symmetric group on n elements, *i.e.* the set of all possible permutations of the set $\{1, 2, \dots, n\}$.

Distributions. Two probability distributions D_1 and D_2 are ε -close, denoted as $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε , namely

$$\Delta_{\text{SD}}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon,$$

where $p_{D_1}(\cdot)$ and $p_{D_2}(\cdot)$ represent the probability mass functions of D_1 and D_2 . We write $D_1 \stackrel{\text{id}}{=} D_2$ to indicate that D_1 and D_2 are *identically distributed*, meaning they have exactly the same probability mass function, *i.e.* $p_{D_1}(x) = p_{D_2}(x)$ for all x .

Distribution Envelopes. Since exact probabilities are often difficult to compute directly, we instead rely on upper bounds expressed through the notion of *probability envelope*. Specifically, for a discrete random variable X , we define $\mathcal{E}$ as an upper envelope of its distribution, meaning that for any value x , the probability $\Pr(X = x)$ is bounded above by $\mathcal{E}(x)$, which we denote by $X \lesssim \mathcal{E}$.

When dealing with probability distributions conditioned on disjoint events, we use collections of probability envelopes indexed by potentially complex structures. For example, for $m, n \in \mathbb{N}^*$, we may consider collections $\mathcal{E} = (\mathcal{E}_{(t_1^{\text{out}}, \dots, t_m^{\text{out}})})$ indexed by tuples $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$ as well as collections $\mathcal{E} = (\mathcal{E}_{(J_1, \dots, J_m)})$ indexed by sets $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$. In this case, each tuple corresponds to a distinct distribution and its associated envelope.

2.2. (Randomized) Arithmetic Circuits

We now define a (randomized) arithmetic circuit, and extend this definition to incorporate randomness not only in the gates themselves, but also in the wiring between gates.

(Randomized) Arithmetic Circuits. An *arithmetic circuit* over a finite field (or finite ring) $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations on $\mathbb{K}$. The circuit is built from a base set of gates $\mathbb{B} = \{g : \mathbb{K}^\ell \rightarrow \mathbb{K}^m\}$, such as addition gates $(x_1, x_2) \mapsto x_1 + x_2$, multiplication gates $(x_1, x_2) \mapsto x_1 \cdot x_2$, and copy gates $x \mapsto (x, x)$. We further consider *probabilistic* (or *randomized*) arithmetic circuits, which also include random gates. Each random gate outputs a fresh value sampled uniformly at random from $\mathbb{K}$. The randomness used by these gates is drawn from a *random tape* denoted by $\rho \in \{0, 1\}^*$.

A (randomized) arithmetic circuit is formally composed of input gates with fan-in zero and fan-out one, and output gates with fan-in one and fan-out zero. Evaluating an ℓ -input m -output circuit C with a random tape ρ involves writing an input $\mathbf{x} \in \mathbb{K}^\ell$ into the input gates, processing the gates sequentially from inputs to outputs using ρ (only for probabilistic evaluation), and finally reading the output $\mathbf{y} \in \mathbb{K}^m$ from the output gates. This is denoted as $\mathbf{y} = C(\rho, \mathbf{x})$, or simply $\mathbf{y} = C(\mathbf{x})$, assuming that ρ is uniformly sampled from $\{0, 1\}^{|\rho|}$.

During evaluation, each wire in the circuit is assigned a value in $\mathbb{K}$. The tuple of all wire values is referred to as a *wire assignment* of C on input $\mathbf{x}$. Formally, we define a probabilistic algorithm called the *assign-wires sampler*, which takes as input a (randomized) arithmetic circuit C , a fixed random tape $\rho \in \{0, 1\}^*$, a set of wire labels $\mathcal{W}$ (a subset of all wire labels in C), and an input $\mathbf{x}$. It outputs a $|\mathcal{W}|$ -tuple $\mathbf{w} \in \mathbb{K}^{|\mathcal{W}|}$, defined as:

$$\mathbf{w} \leftarrow \text{AssignWires}(C, \rho, \mathcal{W}, \mathbf{x}) ,$$

where $\mathbf{w}$ corresponds to the exact values assigned to the wires labeled by $\mathcal{W}$ during the evaluation of $C(\rho, \mathbf{x})$. Notably, $\mathbf{w}$ is deterministically determined by C , ρ , $\mathcal{W}$ and $\mathbf{x}$. We can also induce a probability distribution over $\mathbf{w}$ by replacing the fixed random tape ρ with a uniform random variable R over $\{0, 1\}^{|\rho|}$, yielding the random variable:

$$\mathbf{W} \leftarrow \text{AssignWires}(C, R, \mathcal{W}, \mathbf{x}).$$

Analogously, we define the *assign-gates sampler*, AssignGates , which takes as input a set of gate labels (a subset of all gate labels in C) rather than wires, and assigns to each gate its internal state, represented as an arbitrary function of the gate's inputs, during the evaluation of C on input $\mathbf{x}$ with random tape ρ or random variable R .

Randomized-Wiring Circuits. A *randomized-wiring arithmetic circuit* extends the concept of randomized arithmetic circuits by introducing randomness not only in the gates but also in the *wiring*, *i.e.* the connections between gates. Specifically, a randomized-wiring arithmetic circuit C consists of a fixed set of gates, while the wires connecting these gates are sampled according to a predefined probability distribution. The randomness for this wiring process is drawn from a separate *random tape* denoted by τ . Thus, randomized-wiring circuits involve two distinct random tapes:

- ρ , controlling the randomness of the gates (random values in $\mathbb{K}$),
- τ , controlling the randomness of the wiring configuration.

We denote by C^τ the arithmetic circuit obtained by instantiating C with a particular realization of τ . Note that for any fixed choice of τ , C^τ behaves as a (randomized) arithmetic circuit.

This model is particularly useful for describing masked circuits where operations are performed on shares whose indices are randomly permuted. We will use such designs in Chapters 6 and 8.

It is important to emphasize that in this setting, the corresponding random tape τ (controlling the wiring randomness) is *fully leaked to the adversary*. That is, the simulator in the random probing model receives C^τ for a randomly chosen τ . Consequently, the randomness in wiring is fundamentally different from the randomness from random gates. First, they operate over different domains: random wiring typically involves selecting indices from $[n]$, whereas random gates produce values in the base field (or ring) $\mathbb{K}$. Second, and more critically, while random gates must be implemented with

mechanisms that limit leakage (quantified by the leakage probability p in the random probing model), randomized wiring does *not* require such protection. Since the wiring randomness is entirely leaked to the adversary, its generation might rely on much lighter mechanisms in practice.

Throughout this manuscript, we will not always explicitly specify the random tape τ which refers to the randomness that determines the wiring configuration in randomized-wiring arithmetic circuits, *i.e.* how gates are interconnected. We will indicate τ only when it is relevant to the context (*e.g.* when sharings are actually randomly permuted, see Section 6.2.2), to avoid unnecessarily complicating the notation.

2.3. Masking

Among the various approaches investigated by the community to counteract side-channel attacks, the *masking* countermeasure remains the most widely deployed in practice. Since it has been observed that physical leakages strongly depend on the data being processed, the principle of masking consists in applying secret sharing at the computation level in order to randomize intermediate variables and mitigate side-channel information leakage.

Concretely, for any integer $n > 1$, a $(n - 1)^{th}$ -order masking scheme splits each sensitive variable $x \in \mathbb{K}$ into n shares $(x_1, x_2, \dots, x_n) \in \mathbb{K}^n$, where $n - 1$ shares are sampled uniformly at random and the last one is computed to satisfy:

$$x = x_1 \star x_2 \star \dots \star x_n,$$

for some operation $\star$ under which $(\mathbb{K}, \star)$ typically forms a group. In this setting, any subset of at most $n - 1$ shares reveals no information about the secret. In the following, we use the notation $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n) \in \mathbb{K}^n$ to denote an n -sharing of a secret value $x \in \mathbb{K}$, when n is clear from the context. More generally, given a vector $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{K}^\ell$, we denote by $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ an n -sharing of $\mathbf{x}$, that is $(\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket)$.

Once all the sensitive variables are split into n shares, computations must be performed directly on these shares. Traditional gates implementing basic operations are thus replaced by specialized components called *gadgets*, which operate on masked variables. In this manuscript, we focus exclusively on Boolean masking schemes (where shares are combined using bitwise addition) and arithmetic masking schemes (*i.e.* when shares are combined by a modular addition). Bitwise additions and modular additions will both be denoted by $+$, the exact meaning being clear from the context.

For both Boolean and arithmetic masking, linear gadgets are typically straightforward to implement, as the operations can be performed independently on each share. However, non-linear gadgets are more challenging because the shares cannot be manipulated separately; they require additional fresh randomness to ensure that the leakage of individual atomic operations does not reveal multiple shares simultaneously. If implemented correctly, the masking countermeasure naturally forces an adversary to observe all n shares to recover any information about the secret, thereby leveraging the exponential difficulty of jointly exploiting a growing number of noisy variables.

To reason formally about masked constructions, we introduce two fundamental operators that model the relationship between shared and unshared data. The first is the *n-linear decoding mapping*, denoted by LinDec , which reconstructs a value from its n shares by summing them: for any $n \in \mathbb{N}$ and $(x_1, \dots, x_n) \in \mathbb{K}^n$, we define

$$\text{LinDec}(x_1, \dots, x_n) = x_1 + \dots + x_n.$$

This mapping extends component-wise to tuples of masked inputs. For any $n, \ell \in \mathbb{N}$ and inputs $(\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \in (\mathbb{K}^n)^\ell$, we define:

$$\text{LinDec} : (\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \mapsto (\text{LinDec}(\llbracket x_1 \rrbracket), \dots, \text{LinDec}(\llbracket x_\ell \rrbracket))$$

The second operator is the n -linear encoding algorithm, denoted by LinEnc , which probabilistically maps a value (or vector of values) to uniformly distributed shares such that decoding recovers the original input.

Definition 1 (Linear Sharing). *Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an n -sharing of x is a random vector $\llbracket x \rrbracket \in \mathbb{K}^n$ such that $\text{LinDec}(\llbracket x \rrbracket) = x$. The sharing is said to be uniform if, for any subset $I \subseteq [n]$ with $|I| < n$ the tuple $\llbracket x \rrbracket_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. An n -linear encoding is a probabilistic algorithm LinEnc that, given an input tuple $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{K}^\ell$, outputs a tuple $\llbracket \mathbf{x} \rrbracket = (\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \in (\mathbb{K}^n)^\ell$ where each $\llbracket x_i \rrbracket$ is a uniform n -sharing of x_i for every $i \in [\ell]$.*

In the following, we refer to an $(n\text{-share}, \ell\text{-to-}m)$ gadget as a randomized arithmetic circuit that maps an input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ to an output $\llbracket \mathbf{y} \rrbracket \in (\mathbb{K}^n)^m$. This gadget guarantees that if $\mathbf{x} = \text{LinDec}(\llbracket \mathbf{x} \rrbracket) \in \mathbb{K}^\ell$ and $\mathbf{y} = \text{LinDec}(\llbracket \mathbf{y} \rrbracket) \in \mathbb{K}^m$, then $\mathbf{y} = g(\mathbf{x})$ for some function g .

Example. Figure 2.1 illustrates a $(2\text{-share}, 2\text{-to-}1)$ gadget C_{ex} , which computes a multiplication.

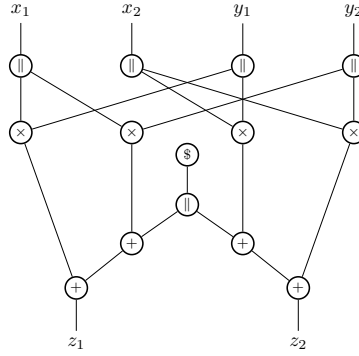


Figure 2.1.: Example of a randomized arithmetic circuit C_{ex} with two 2-share inputs $(\llbracket x \rrbracket = (x_1, x_2), \llbracket y \rrbracket = (y_1, y_2)) \in (\mathbb{K}^2)^2$, one 2-share output $\llbracket z \rrbracket = (z_1, z_2) \in \mathbb{K}^2$, addition gates (+), multiplication gates ($\times$) and copy gates ($\parallel$), and a random gate (\$).

Definition 2 (Circuit Compiler). *Let $n, \ell, m \in \mathbb{N}$. A circuit compiler with linear decoding mapping consists of a pair of algorithms $(\text{CC}, \text{LinEnc})$ defined as follows:*

- CC (circuit compilation) is a deterministic algorithm that takes as input a (randomized) arithmetic circuit C and outputs a randomized arithmetic circuit $\hat{C}$,
- LinEnc (n -linear input encoding) is a probabilistic algorithm as defined in Definition 1,

which satisfy the following properties:

- **Correctness:** For any (randomized) arithmetic circuit C of input length ℓ , with random tape $\rho \in \{0, 1\}^*$, and for every $\mathbf{x} \in \mathbb{K}^\ell$, it holds that:

$$\Pr(\text{LinDec}(\hat{C}(\hat{\rho}, \llbracket \mathbf{x} \rrbracket)) = C(\rho, \mathbf{x}) \mid \llbracket \mathbf{x} \rrbracket \leftarrow \text{LinEnc}(\mathbf{x})) = 1,$$

where $\hat{C} = \text{CC}(C)$, and $\hat{\rho} \in \{0, 1\}^*$ derived from ρ and $\hat{C}$.

- **Efficiency:** For some security parameter $\kappa \in \mathbb{N}$, the running time of $\text{CC}(C)$ is $\text{poly}(\kappa, |C|)$, the running time of $\text{LinEnc}(\mathbf{x})$ is $\text{poly}(\kappa, |\mathbf{x}|)$ and the running time of $\text{LinDec}(\llbracket \mathbf{y} \rrbracket)$ is $\text{poly}(\kappa, |\llbracket \mathbf{y} \rrbracket|)$, where $\text{poly}(\kappa, \ell) = \mathcal{O}(\kappa^{e_1} \ell^{e_2})$ for some constants e_1, e_2 .

2.4. Leakage Models

To prove the security of masking schemes, it is essential to formally characterize the side-channel leakage an attacker may extract, which leads to the notion of leakage models.

In this section, we first present a very realistic leakage model for capturing the power and electromagnetic side-channel emissions, namely the *noisy leakage model*, and we acknowledge that there is currently no known method to directly and accurately prove the security of masking schemes in this model. We then recall one of the most widely used leakage models, the *probing model*, which provides a less accurate representation of the physical reality of embedded devices, but significantly simplifies the task of constructing security proofs. Finally, we discuss the model that constitutes the focus of this manuscript: the *random probing model*. This model benefits from a tight reduction, which ensures that security in the random probing model implies security in the noisy leakage model. It makes it a powerful tool for analyzing physical security, while remaining more tractable for security proofs than directly working in the noisy leakage setting, albeit more complex than the classical probing model.

2.4.1. Noisy Leakage Model

The *noisy leakage model*, first introduced by Chari, Jutla, Rao, and Rohatgi in [CJRR99] and later generalized by Prouff and Rivain in [PR13] is widely recognized for capturing well the physical reality of power and electromagnetic side-channel emissions. It can be viewed as a specialization of the *only computation leaks* paradigm [MR04], which formalizes the assumption that a cryptographic computation is composed of a sequence of elementary operations, each leaking a function f of the specific portion X of the device's state it accesses.

Formal Definition. In the noisy leakage model, a leaking computation is modeled by a sequence of *elementary operations* $(g_i)_i$ that access a shared memory called the *internal state*. Each elementary operation reads its input and writes its output to this internal state. When processed on some input $\mathbf{x}_i$, an elementary operation g_i reveals a *noisy leakage function* of its input. This leakage function takes two arguments: the value $\mathbf{x}_i$ held in the accessed part of the internal state (*data isolation assumption*) and a random string ρ_i long enough to model the noise. Over an entire execution, the leaked information consists of the sequence $(f_i(\mathbf{x}_i, \rho_i))_i$, where the $\mathbf{x}_i$ are the successive intermediate values read from the internal state by each g_i , and the ρ_i are fresh random strings. Notably, all the ρ_i involved in successive operations are drawn uniformly and independently (*independent noise assumption*).

For the sake of simplicity, we omit the explicit random string parameter and write $f_i(\mathbf{x}_i)$, where $\mathbf{x}_i$ is the accessed value. Here, $f_i(\mathbf{x}_i)$ should be understood as a random variable representing the output of a probabilistic algorithm applied to $\mathbf{x}_i$. That is, for a fixed input $\mathbf{x}_i$, the function f_i can produce different outputs according to a certain probability distribution. Even though f_i is a noisy function, we do not model it as a continuous random variable. In practice, the resolution of the leakage is limited by the acquisition chain. Consequently, we treat the leakage as a discrete random variable, taking values in a finite set $\text{Range}(f_i)$ determined by the precision of the measurement process.

The noisy nature of f_i is captured by bounding the bias it introduces in the distribution of a uniformly random variable X when conditioned on the leakage $f_i(X)$. This notion is formalized in the next definition.

Definition 3 (Noisy Function). *Let $\delta \in \mathbb{R}$. A δ -noisy leakage function f over $\mathbb{K}$ is a function defined on the domain $\mathbb{K} \times \{0, 1\}^{|\rho|}$ for some $|\rho| \in \mathbb{N}$, such that*

$$\beta_{\Delta}(X|Y) := \sum_{y \in \text{Range}(f)} \Pr(Y = y) \cdot \Delta(X; (X | Y = y)) \leq \delta, \quad (2.1)$$

where X is a uniformly distributed random variable¹ over $\mathbb{K}$, R is a uniformly distributed random

¹Uniformity is assumed to make the definition sound and simplify the bound on $\beta(X|Y)$; for non-uniform X , the bias increases by a factor equal to the size of $\mathbb{K}$.

variable over $\{0,1\}^{|\rho|}$, $Y = f(X, R)$, and Δ denotes a distance measure between two probability distributions over $\mathbb{K}$.

In the original definition from [PR13], the authors used for Δ the Euclidean distance based on the L_2 norm. The authors of [DDF14] then proposed to adopt the statistical distance based on the normalized L_1 norm. More recently, Prest et al. suggested in [PGMP19] an alternative distance based on the *relative error*. We recall below the definitions of the L_1 statistical distance (Δ_{SD}) and the *relative error* (Δ_{MRE}), both based on the *pointwise mutual information*.

Definition 4 (Pointwise Mutual Information). *Let X, Y be discrete random variables over $\mathcal{X}, \mathcal{Y}$ respectively. For any $x \in \mathcal{X}, y \in \mathcal{Y}$, the exponential form of the pointwise mutual information (PMI) is defined as:*

$$\text{PMI}_{X,Y}(x, y) = \frac{P[X = x, Y = y]}{P[X = x] \cdot P[Y = y]} - 1.$$

Definition 5 (Statistical Distance and Relative Error). *Let X, Y be discrete random variables over $\mathcal{X}, \mathcal{Y}$ respectively. We can define the L_1 statistical distance (SD) as follows:*

$$\Delta_{SD}(X; (X|Y = y)) = \frac{1}{2} \sum_{x \in \mathcal{X}} \Pr[X = x] \cdot |\text{PMI}_{X,Y}(x, y)|$$

and the *relative error* (RE) as follows:

$$\Delta_{MRE}(X; (X|Y = y)) = \max_{x \in \mathcal{X}} |\text{PMI}_{X,Y}(x, y)|.$$

From the above definitions, we can derive the security against δ -noisy leakage.

Definition 6 (Security against δ -Noisy Leakage). *Let X be a uniformly distributed random variable over $\mathbb{K}^\ell$. A randomized arithmetic circuit C , which takes as input an encoding of X , and consisting of a set G of internal gates (excluding input and output gates, which are merged when composing circuits) is said to be ε -secure against δ -noisy leakage with respect to a distance measure Δ if:*

$$\beta_\Delta(X|f_1(X_1), \dots, f_{|G|}(X_{|G|})) \leq \varepsilon$$

for any collection of δ -noisy leakage functions $f_1, \dots, f_{|G|}$, where for each $1 \leq i \leq |G|$, X_i denotes an arbitrary function of the inputs to the i^{th} gate during the evaluation of C .

Example. Getting back to the circuit C_{ex} of Figure 2.1, X would represent the pair $(x = x_1 + x_2, y = y_1 + y_2) \in \mathbb{K}^2$ and the $|G| = 14$ gates would each leak a different function of their inputs.

Limitations. Prouff and Rivain made a significant contribution in [PR13] by introducing a compiler that transforms any randomized arithmetic circuit into one that is ε -secure against δ -noisy leakage. However, their construction relies on the use of *leak-free* gates, which process secret sharings without leaking any information. Due to this practical limitation, most masking schemes are instead proven secure in the probing model, which, although less accurate in capturing physical leakage, enables the construction of complete security proofs. This model is recalled in the next section.

2.4.2. Probing Model

The *probing model*, introduced by Ishai, Sahai, and Wagner in 2003 [ISW03], is one of the most widely used leakage models in side-channel analyses. Informally, the t -probing model assumes that during the evaluation of a circuit, an adversary can observe the exact values carried by up to t wires of their

choice. A circuit is therefore t -probing secure if the values of any t intermediate variables, known as *observations* or *probes*, reveal no information about the secret inputs.

Although, in practice, side-channel leakages are close to noisy functions of the processed data, the probing model is motivated by the intuition that extracting information from noisy observations of t variables becomes exponentially hard as t grows. This makes the model a useful abstraction for proving security, even though it does not fully capture the physical realities of embedded devices.

Formal Definition. We now recall the formal definition of t -probing security. This notion relies on the existence of a probabilistic polynomial-time (PPT) *simulator*², which is a probabilistic algorithm designed to produce outputs that are distributed identically to the leakage an adversary would observe.

Definition 7 (t -Probing Leakage). *Let C be a randomized arithmetic circuit, $\mathcal{W}$ a set of t wire labels. Let $|\rho|$ denote the size of the random tape for C and let R be a uniform random variable over $\{0, 1\}^{|\rho|}$. We define the leakage corresponding to $\mathcal{W}$ as the following random variable*

$$\mathcal{L}_{\mathcal{W}}(C, \llbracket \mathbf{x} \rrbracket) \stackrel{\text{def}}{=} \text{AssignWires}(C, R, \mathcal{W}, \llbracket \mathbf{x} \rrbracket)$$

for any input $\mathbf{x} \in \mathbb{K}^\ell$ of C , where $\mathcal{L}_{\mathcal{W}}(C, \llbracket \mathbf{x} \rrbracket)$ represents a tuple of $t = |\mathcal{W}|$ random variables corresponding to the assignments of the wires with label in $\mathcal{W}$ for an evaluation of $C(R, \llbracket \mathbf{x} \rrbracket)$.

Definition 8 (t -Probing Security). *Let C be a randomized arithmetic circuit. C is said to be t -probing secure if there exists a simulator Sim such that, for any input $\mathbf{x} \in \mathbb{K}^\ell$, and for every set of wires $\mathcal{W}$ of C such that $|\mathcal{W}| \leq t$, the following holds:*

$$\text{Sim}(C, \mathcal{W}) \stackrel{\text{id}}{=} \mathcal{L}_{\mathcal{W}}(C, \llbracket \mathbf{x} \rrbracket)$$

Example. In the circuit C_{ex} in Figure 2.1, each internal wire, except for the outputs, depends on at most one share of each input (*i.e.* either x_1 , x_2 , or none of them and either y_1 , y_2 , or none of them). The output wires are generally excluded from the analysis, as by convention they are regarded as the inputs of subsequent gadgets. Even if they were included, each output (taken individually) could be perfectly simulated by sampling a random element of $\mathbb{K}$, thanks to the presence of the fresh randomness. C_{ex} is thus 1-probing secure.

Limitations. Most masking schemes are proven to be secure in the t -probing model up to $t = n - 1$ where n denotes the number of shares. However, this security guarantee does not fully reflect the realities of embedded devices, in which an attacker observes side-channel leakages that are more accurately modeled as noisy functions of the manipulated data. This gap motivates the introduction of the intermediate model and the reduction described in the following section.

2.4.3. Random Probing Model and Reduction

Recall that while the noisy leakage model provides an accurate representation of physical side-channel leakages, it does not lend itself easily to efficient security proofs. Conversely, the probing model significantly simplifies the construction of proofs but fails to realistically capture the leakage behavior of embedded devices. To bridge this gap, Duc, Dziembowski, and Faust established in 2014 a reduction showing that security in the noisy leakage model can be derived from security in the probing model [DDF14]. This reduction is built upon an intermediate leakage model known as the *random probing model*.

Informally, the random probing model assumes that each intermediate variable in a circuit C leaks with a fixed probability p . A circuit is deemed secure if the probability ε that the leaking wires

²All simulators in this manuscript are assumed to be probabilistic polynomial-time (PPT). For brevity, this will not be explicitly mentioned in subsequent definitions and properties.

collectively reveal any information about the secret remains negligible. Unlike the probing model, the random probing model naturally captures horizontal attacks, which exploit repeated manipulations of variables within an implementation. More generally, the security in the noisy leakage model reduces to the security in the random probing model, making it a powerful and realistic abstraction for analyzing physical security.

The reduction from noisy leakage security to probing security proceeds in two main steps. First, by applying the Chernoff bound [Che52], it can be shown that any circuit C secure in the t -probing model is also secure in the (p, ε) -random probing model, where the leakage probability satisfies $p = (t + 1)/(2|C|)$ and the attacker's distinguishing advantage ε is upper bounded by $\exp(-(p \cdot |C|)/3)$, with $|C|$ denoting the number of gates in the original circuit. However, this first step suffers from a major limitation: the tolerated leakage probability p decreases as the circuit size $|C|$ grows. The second step, and the key lemma of the reduction, establishes that any circuit which is (p, ε) -secure in the random probing model is also (δ, ε) -secure in the noisy leakage model (for the same parameter ε) defined with respect to the statistical distance Δ_{SD} (see Equation 2.1) and for any $\delta \leq p/|\mathbb{K}^2|$, where $\mathbb{K}^2$ is the input space of the internal gates (or elementary operations)³. This result was later extended to the noisy leakage model defined with respect to the maximum relative error Δ_{MRE} in the work of Prest et al. [PGMP19]. Those security reductions directly hold from the following key lemma.

Lemma 1 ([DDF14, PGMP19]). *Let $\phi_p : \mathcal{X} \rightarrow \mathcal{X} \cup \{\perp\}$ the randomized function defined for every $p \in [0, 1]$ as*

$$\phi_p(x) = \begin{cases} \perp & \text{with probability } 1 - p \\ x & \text{with probability } p \end{cases}$$

Let $f : \mathcal{X} \rightarrow \mathcal{Y}$ be a δ -noisy leakage function (w.r.t. Δ_{SD} or Δ_{MRE}). There exists a randomized function $f' : \mathcal{X} \cup \{\perp\} \rightarrow \mathcal{Y}$ such that for every $x \in \mathcal{X}$ we have

$$f(x) = f'(\phi_p(x)) \quad \text{with} \quad \begin{cases} p \leq \delta \cdot |\mathcal{X}| & \text{if } \beta_{\Delta_{\text{SD}}}(X|f(X)) \leq \delta \\ p \leq \delta & \text{if } \beta_{\Delta_{\text{MRE}}}(X|f(X)) \leq \delta \end{cases}$$

We recall that the MRE is a worst-case metric, contrary to the SD, which is an average-case metric. This explains the tighter reduction (*i.e.* no loss induced by the size of the input space) using the MRE from the noisy model to the random probing model since the latter is also a worst-case model.

These insights motivate the adoption of the random probing model, which benefits from a strong connection with noisy leakage model (security in the latter reduces to security in the former) and thus offers a more realistic and robust framework for analyzing the physical security of cryptographic implementations. Its formal definition, along with fundamental tools for verifying its security properties, will be presented in the next chapter.

³The internal state of a gate can be described by both its input wires, each defined on $\mathbb{K}$, hence the input space is $\mathbb{K}^2$.

Part II.

Random Probing Security

Chapter 3

Random Probing Security

Contents

3.1. Random Probing Leakage and Security Notion	12
3.2. Simulation Failure Probability	13
3.3. Automatic Verification Tools	14

This chapter focuses on the random probing security, as formalized in our 2020 work [BCP⁺20] (Appendix A) and further extended in subsequent publications. We also describe the methodology for computing security advantages, which forms the foundation of our verification tools.

3.1. Random Probing Leakage and Security Notion

Let $p \in [0, 1]$ denote a constant leakage probability parameter, also referred to as the *leakage rate*. In the p -random probing model, the evaluation of an arithmetic circuit C leaks the value carried by each wire with probability p , where all wire leakage events are mutually independent.

To formally define the random probing leakage of a circuit, we introduce a *leaking-wires sampler*. This is a probabilistic algorithm that, given a (randomized) arithmetic circuit C and a probability $p \in [0, 1]$, outputs a set $\mathcal{W}$ of wire labels, denoted as

$$\mathcal{W} \leftarrow \text{LeakingWires}(C, p),$$

where each wire label from the circuit C (but the output ones) is included in $\mathcal{W}$ independently with probability p . Output wires are excluded, since they are merged when composing circuits. Analogously, we define the *leaking-gates sampler*, denoted $\mathcal{G} \leftarrow \text{LeakingGates}(C, p)$, which outputs a set $\mathcal{G}$ of gate labels rather than wire labels. Here again, input and output gates are excluded for the same compositional reason.

Example. Figure 3.1 illustrates both samplers on C_{ex} .



Figure 3.1.: Circuit C_{ex} with LeakingWires (on the left) and LeakingGates (on the right).

Definition 9 (Random Probing Security). *Let $\ell \in \mathbb{N}$ and $p, \varepsilon \in [0, 1]$. A randomized arithmetic circuit C is (p, ε) -random probing secure (or RPS for short) with respect to encoding Enc if there exists a simulator Sim such that for every input $\mathbf{x} \in \mathbb{K}^\ell$, the distribution of $\text{Sim}(C)$ is ε -close to the distribution of $\text{AssignWires}(C, R, \mathcal{W}, \llbracket \mathbf{x} \rrbracket)$ where*

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(C, p) && \text{Sampling the leaking wires.} \\ \llbracket \mathbf{x} \rrbracket &\leftarrow \text{Enc}(\mathbf{x}) && \text{Sampling an encoding of the input.} \end{aligned}$$

for $|\rho|$ the size of the random tape for C and R a uniform random variable on $\{0, 1\}^{|\rho|}$. The parameter ε , referred to as the simulation failure probability, directly bounds the distinguishing advantage of any adversary trying to distinguish between the real leakage and the simulated leakage.

We equivalently define (p, ε) -random probing security for a circuit in the gate leakage setting, where we use LeakingGates and AssignGates instead of LeakingWires and AssignWires .

Informally, in the random probing model, although a large number of wires may leak simultaneously, the probability of such events inherently limits their impact. For example, if $\mathcal{W}$ includes the labels of all the shares of some input $x \in \mathbb{K}$, the simulator will not be able to produce a valid output without knowing x . Fortunately, this event only happens with a probability p^n (n being the number of shares) which becomes negligible as n grows.

This notion of random probing security can be further generalized to encompass circuits that handle:

- **Auxiliary inputs**, as encountered in cryptosystems using random nonces that can be partially leaked without compromising security. A typical example is the masked lattice-based signature scheme Raccoon, in which certain random nonces may be exposed to an attacker without harming security. In security proofs, these random nonces are treated not as shares but as additional inputs [dPKPR24]. In the random probing security notion with auxiliary inputs, this partial leakage is modeled by allowing the simulator to request specific subsets of coordinates for each auxiliary input. The cardinalities of these subsets follow a list of distributions, which are parameters of the generalized security notion.
- **Public outputs**, which refer to intermediate variables intentionally unmasked at specific stages, for example, after sufficient noise has been added or following a successful verification test. Such public outputs are common in lattice-based post-quantum schemes. In the generalized model, these additional unmasked outputs are fully disclosed to the simulator.

The corresponding generalized definition is provided in our paper [BRR25] (see Appendix F).

3.2. Simulation Failure Probability

As stated in Definition 9, the simulation failure probability ε serves as an upper bound on the side-channel attacker's distinguishing advantage. In this section, we explain how to compute ε as a function of the leakage probability p for basic gadgets.

Consider a randomized arithmetic circuit C composed of s wires labeled from 1 to s , and let R denote a uniform random variable over $\{0, 1\}^{|\rho|}$, where $|\rho|$ is the size of C 's random tape. For any subset $\mathcal{W} \subseteq [s]$, we define $\delta_{\mathcal{W}}$ as follows:

$$\delta_{\mathcal{W}} = \begin{cases} 0 & \text{if } \exists \text{ Sim, Sim}(C) \stackrel{\text{id}}{=} \text{AssignWires}(C, R, \mathcal{W}, \llbracket x \rrbracket) \text{ for every input } x, \\ 1 & \text{otherwise.} \end{cases}$$

The simulation failure probability ε from Definition 9 can then be explicitly expressed as a function of p . Namely, we have $\varepsilon = f(p)$, where f is defined for every $p \in [0, 1]$ as:

$$f(p) = \sum_{\mathcal{W} \subseteq [s]} \delta_{\mathcal{W}} \cdot p^{|\mathcal{W}|} \cdot (1 - p)^{s - |\mathcal{W}|} . \quad (3.1)$$

Let c_i denote the number of subsets $\mathcal{W} \subseteq [s]$ of cardinality i for which $\delta_{\mathcal{W}} = 1$, namely for which the simulation fails, we have

$$c_i = \sum_{\substack{\mathcal{W} \subseteq [s] \\ |\mathcal{W}|=i}} \delta_{\mathcal{W}} .$$

With this notation, Equation 3.1 simplifies to:

$$f(p) = \sum_{i=1}^s c_i \cdot p^i \cdot (1-p)^{s-i} . \quad (3.2)$$

For any circuit C achieving t -probing security (see Definition 8), we have $\delta_{\mathcal{W}} = 0$ for all subsets $\mathcal{W}$ such that $|\mathcal{W}| \leq t$. Consequently, the corresponding coefficients c_i are zero for $i \leq t$ and $f(p)$ simplifies to:

$$f(p) = \sum_{i=t+1}^s c_i \cdot p^i \cdot (1-p)^{s-i} .$$

Moreover, by definition, the coefficients c_i satisfy the upper bound $c_i \leq \binom{s}{i}$, which leads to the following upper-bound on $f(p)$:

$$f(p) \leq \sum_{i=t+1}^s \binom{s}{i} \cdot p^i \cdot (1-p)^{s-i} .$$

Example. Let us consider the 2-share multiplication gadget C_{ex} represented in Figure 3.1 that takes as inputs the 2-sharings $(x_1, x_2) \in \mathbb{K}^2$ and $(y_1, y_2) \in \mathbb{K}^2$ of x and y respectively, and outputs the 2-sharing $(z_1, z_2) \in \mathbb{K}^2$. It can be verified that the described multiplication gadget consists of 21 wires (excluding the two output wires, as per convention):

- four input shares, each used twice after a copy gate, accounting for 12 wires;
- one random value used twice, accounting for 3 wires;
- six intermediate variables (excluding the two outputs), accounting for 6 wires.

Since this gadget is 1-probing secure (see Section 2.4.2), every set containing a single wire can be simulated, implying $c_1 = 0$. However, this is not the case for all pairs of wires. For example, the pairs (x_1, x_2) or (y_1, y_2) cannot be simulated jointly without the knowledge of x or y , so $c_2 \geq 2$. Computing each coefficient c_i for $2 \leq i \leq 21$ (with $c_{21} = 1$) in (3.2) yields the exact expression of $f(p)$ for C_{ex} .

3.3. Automatic Verification Tools

Equation 3.2 serves as the foundation for most automatic verification tools in the random probing model. In practice, these tools aim at determining the values of the coefficients c_i , which capture the contribution of each subset of wires to the overall security of the implementation. The main challenge therefore lies in designing efficient methods to compute or approximate accurately these coefficients in increasingly complex circuits.

In 2020, we introduced VRAPS [BCP⁺20] (Appendix A), the first tool capable of verifying random probing security:

<https://github.com/CryptoExperts/VRAPS>.

While inspired by my earlier work on `maskVerif` [BBD⁺15, BBC⁺19], developed with my co-authors as the first tool able to efficiently verify probing security at higher orders, `VRAPS` goes further by checking *all* tuples of wires rather than only subsets of size t . It was later complemented by `STRAPS` [CFOS21], which introduced a probabilistic mode to improve scalability. Both tools, however, cannot provide complete verification.

Building on these advances, we released `IronMask` in 2022 [BMRT22], as a major step to address this issue. `IronMask` supports the verification of masking gadgets involving both *linear randomness* (e.g. addition, copy, and refresh gadgets) and *quadratic randomness* (e.g. multiplication gadgets with input refreshing). Most importantly, it provides *complete verification results* for both classes of gadgets. The tool has since been extended to address stronger adversarial models, including combined attacks that exploit both fault injection and random probing leakage [BFG⁺24] (Appendix E):

<https://github.com/CryptoExperts/IronMask>.

Example. With `IronMask`, we can typically compute the coefficients of the polynomial f for the circuit C_{ex} so that for any leakage rate $p \in [0, 1]$, the random probing advantage of C_{ex} is $\varepsilon = f(p)$. Specifically, the tool returns the coefficients c_i of Equation 3.2:

$$(c_1, \dots, c_{21}) = (0, 49, 737, 4763, 18735, 52798, 115338, 203064, 293800, 352692, 352714, 293930, 203490, 116280, 54264, 20349, 5985, 1330, 210, 21, 1).$$

In particular, it can be verified that $c_1 = 0$ since the gadget is 1-probing secure, and that $c_{21} = 1$ as expected for the tuple containing all 21 wires. Moreover, we can observe that $c_2 = 49$, meaning that 49 wire pairs cannot be simulated without the knowledge of a secret input.

The most recent (published) development in this direction is the `INDIANA` framework by Beierle et al. [BFG⁺25], which achieves exact verification for arbitrary masked circuits. While `INDIANA` represents a major advance, it still faces scalability limitations and cannot verify a full AES-128 implementation instantiated with more than two shares.

Finally, in collaboration with Gaëtan Cassiers, we recently developed a probabilistic tool, named `perseus`, to evaluate the random probing security of complex circuits (e.g. AES-128 masked with eight shares). While this tool also relies on `maskVerif` verification rules (and is therefore not fully complete), it enables a more accurate evaluation of the random probing security of large circuits. The key idea is to exploit the fact that many intermediate gadgets satisfy composability properties in the probing model, and to leverage these properties to reduce the verification task.

Chapter 4

From Random Probing Security to Practical Security

Contents

4.1. Physical Assumptions	16
4.2. Our Methodology	17
4.3. Design Goals and Remaining Challenges	23

In the previous chapters, we discussed the importance of the random probing model, as security in this model implies security in the noisy leakage model, which is known to well capture the reality of embedded devices. However, translating a theoretically proven random probing secure implementation into a physically secure implementation on a real device is not as simple as merely applying the reduction. In short, the reduction must be handled carefully with meaningful parameters, and the noisy leakage model remains idealized, requiring certain assumptions to be either made explicit or relaxed in order to achieve practical, concrete security on an actual device.

In this chapter, building on our Systematization of Knowledge published in 2025 [BCM⁺25] (Appendix G), we first explicit the physical assumptions on which the noisy leakage model relies. We then present a comprehensive methodology for transforming random probing secure implementations (or even abstract cryptographic algorithms) into physically secure implementations resistant to side-channel attacks on microcontrollers. Finally, we highlight the remaining challenges arising from physical limitations and from the overhead induced by our design choices and non-tight proofs.

4.1. Physical Assumptions

The noisy leakage model has been argued to capture power and electromagnetic leakages effectively. In all generality, an elementary operation processing a value $\mathbf{x}$ gives rise to a leakage trace $f(\mathbf{x})$ which is a multivariate random variable (also called a random vector) following a distribution whose parameters depend on $\mathbf{x}$. In most practical contexts, this distribution is well approximated by a multivariate Gaussian $\mathcal{N}(\mathbf{m}_x, \Sigma)$ for some parameters $\mathbf{m}_x$ (mean vector) and Σ (covariance matrix), see *e.g.* [CRR03, SLP05]. Such parameters can be inferred in practice through a profiling of the device, from which we obtain the noisy leakage metric δ by evaluating Equation 2.1.

We still need to stress that, as is, the noisy leakage model relies on two assumptions about the underlying physical device which might not be verified in practice without further care.

Assumption 1 (Data isolation). *A leakage function f_i corresponding to the elementary operation $g_i(\mathbf{x}_i)$ only depends on the current state $\mathbf{x}_i$ and not on previously accessed parts of the state: $\mathbf{x}_{i-1}$, $\mathbf{x}_{i-2}$, $\dots$*

However, in practice, physical phenomena such as glitches and signal transitions can break this implicit assumption. Glitches are unintended transient pulses caused by variations in circuit delays,

which may reveal information on several intermediate variables. Meanwhile, transitions correspond to changes in logic states that can momentarily leak sensitive information about two intermediate variables simultaneously. Therefore, particular care must be taken to enforce data isolation for the model to remain valid.

Note that this assumption can be relaxed to allow the leakage to be composed of linear combinations of independent noisy leakage functions [BDF⁺17]. Such linear combinations do not affect the statistical security order since given leakage samples of which the statistical moments up to a given order are independent of any sensitive variable, their linear combination will have the same guarantees.

Assumption 2 (Noise independence). *The leakage noises from the successive elementary operations are independent from each other. Formally, the random tape ρ_i in each $f_i(\mathbf{x}_i, \rho_i)$ is sampled as a fresh uniform string.*

In practice, this assumption does not easily hold: if one cuts a leakage trace into several sub-traces corresponding to successive elementary operations, the noises in the successive sub-traces would likely include some part of dependency. Indeed, a correlation exists between successive leakage points, which makes multivariate statistics particularly useful for side-channel attacks [CRR03].

In the following, we shall refer to the original noisy leakage model, which relies on the two aforementioned assumptions as the *idealized noisy leakage model*. We will explore how to relax or enforce those physical assumptions to reduce the security of a physical implementation to that of an abstract implementation in the idealized noisy leakage model, which subsequently reduces to the random probing security.

4.2. Our Methodology

The overall process of transforming an abstract circuit into a secure physical implementation involves several steps. As illustrated in Figure 4.1, these steps can be split into two phases:

- (i) a *characterization phase* which only depends on the device and the side-channel acquisition tool (*i.e.* without any knowledge of the abstract circuit). It includes the implementation of specific gates (Step 1), the analysis of Assumption 1 (data isolation) to exhibit a relevant whitening procedure (Step 2), the characterization of the leakage (Step 3), the enforcement and relaxation of Assumption 2 (noise independence) (Step 4), and the estimation of the noisy leakage parameter (Step 5);
- (ii) a *compilation phase* using the outputs of the characterization phase to turn an abstract circuit C , with a set of gates $\mathcal{G}$, into a practically secure implementation for a given security level λ (Step 6). This phase relies on the usage of a secure random probing compiler, which takes as input an abstract circuit, a leakage probability parameter p , and a desired security level λ , and produces a protected implementation that meets the corresponding security guarantees¹.

All intermediate steps are described at a high level in the following subsections. In [BCM⁺25] (see Appendix G), we formally prove the validity of our approach and present the experiments conducted to validate its practical feasibility. Additionally, we introduce dedicated procedures for achieving data and noise isolation.

Step 1: Implementing Abstract Gates

The first step of our methodology consists of implementing abstract gates as software routines. A physical elementary operation abstracted as a gate by the noisy leakage model first looks up its operands from memory (the computation state), then executes a sequence of arithmetic instructions

¹Our methodology is described in the context of software implementations only, where elementary operations correspond to software routines.

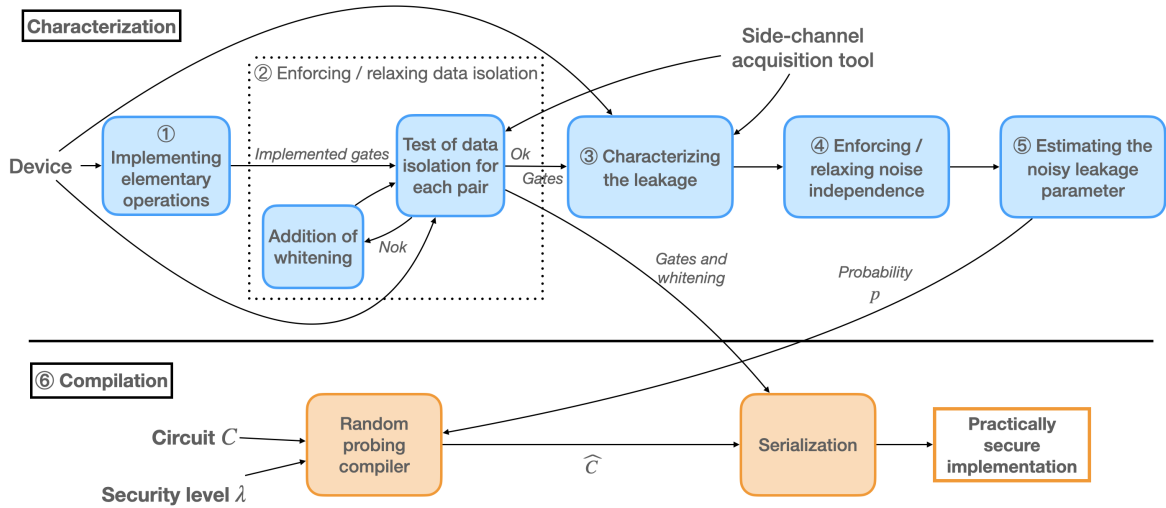


Figure 4.1.: Illustration of our methodology

(implementing the gate functionality $g \in \mathcal{G}$), and finally writes back the result to memory. This process generates some side-channel leakage depending on the executed instructions and the processed data, which is the leakage of the physical elementary operation abstracted by the noisy leakage model. A developer must first translate this behavior into a software routine on a physical device. We propose to implement such a routine in ARM assembly as follows (with the `xor` operation as an example):

```
operation_xor:
  ldr r0, [r0]
  ldr r1, [r1]
  eor r0, r1, r0 // For other operations, change instruction
  str r0, [r2]
```

with the following C signature:

```
void operation_xor(const uint32* aPtr, const uint32* bPtr, uint32* cPtr);
```

We define a routine for each abstract gate $g \in \mathcal{G}$ which, when executed on the target device, behaves as a physical elementary operation abstracted by the noisy model. From these implementations of the abstract gates, any arithmetic circuit C can be compiled into a physical implementation on the target device. This implementation takes the form of a sequence of calls to the elementary operations, looking like the following C-syntax example:

```
operation1(a1Ptr, b1Ptr, c1Ptr);
operation2(a2Ptr, b2Ptr, c2Ptr);
...
```

The routines `operation1`, `operation2`, ... are all among the implemented gate routines which are mapped from the gates of the circuit. The pointer arguments $(a1Ptr, b1Ptr, c1Ptr)$, $(a2Ptr, b2Ptr, c2Ptr)$, ... are constant addresses triplets which encode the data dependency of the implementation, *i.e.* the wires in the abstract circuit².

Step 2: Enforcing / Relaxing Data Isolation

Once the syntax of elementary operations is fixed, the implementation must satisfy the noisy-leakage model's physical assumptions to apply the security reduction. Our methodology first focuses on the *data isolation* assumption (Assumption 1), which requires the leakage of each elementary operation to depend only on its own inputs, independently of the preceding and following operations.

²The proposed implementation style is admittedly not very efficient, this work mainly targets security and simplicity.

In practice, this assumption rarely holds: consecutive operations often leak jointly because data processed by one operation might be stored in the CPU’s physical state (registers, buses, caches). The leakage of the next operation is then a probabilistic function of both its inputs and this residual state, *i.e.* of previously processed data. This phenomenon is well known [MOW17, MPW22, MKSM22] and may have concrete impact, typically lowering the effective masking order [BGG⁺14].

Consequently, a developer cannot simply implement a circuit as a sequence of Step 1 routines; without isolation, the side-channel security can no longer be reduced to the random probing model. In the following, we address the data isolation assumption (Assumption 1) on two fronts: on the one hand, we *enforce* it at the implementation level, as discussed in prior works [BC22, CS21]; on the other hand we *relax* the model in a controlled way. Additionally, we design a dedicated test to validate data isolation on a target device.

Enforcing isolation via data whitening. Our methodology uses *data whitening* to enforce data isolation. The idea is to invoke a routine on constant or random data whose sole purpose is to purge the CPU state of any dependency on previously processed data. Concretely, after each call to an elementary operation routine, we insert one or more calls whose arguments point to random or constant data in memory. By exercising the same data path with a similar routine, we expect to overwrite any hardware register or microarchitectural buffer that may still carry information from the previous call.

This approach is natural but its effectiveness is device dependent: it hinges on the microarchitecture of the target CPU. In practice, developers may need to empirically tune the number and type of whitening calls (and their placement) to achieve robust and efficient isolation.

Relaxing the model. Even with whitening that suppresses transition and glitch effects across operations, the leakage trace may still fail to decompose into time slices that each correspond to a single operation. Indeed, the leakage is often subject to low-pass filtering inside the target chip or the measurement chain. As a result, the independent intrinsic leakage of many operations will be linearly combined in the measured trace. To account for this, we relax the noisy leakage model to allow the measured trace to be a *linear combination* of independent noisy leakage functions. With this relaxation, we still provide independence between the inputs of the different operations, *i.e.* data isolation.

Testing data isolation. In our paper [BCM⁺25] (see Appendix G), we introduce a novel way to test the effectiveness of a data whitening routine. The idea is to suppose that the leakage distribution can be modeled as a sum of a deterministic function of the first operation’s inputs, a deterministic function of the second operation’s inputs, and some noise value. In other words, we test that the leakage can be decomposed in additive parts that do not jointly depend on the inputs of both operations.

Step 3: Characterizing the Leakage

Once data isolation is enforced and tested, one can safely infer the leakage distribution of each physical elementary operation. This is a classical problem in the side-channel literature, and we can rely on a solid theoretical and practical ground for this step. We rely on the common assumption [CK13, SLP05] that the leakage distribution $\mathcal{Y}$ of an elementary operation with inputs $\mathbf{x} \in \mathcal{V}^\ell$ takes the form of a deterministic function of $\mathbf{x}$ plus an additive (discrete) Gaussian noise:

$$\mathcal{Y}_{\mathbf{x}} = d(\mathbf{x}) + \mathcal{N}(\mathbf{0}, \Sigma) , \quad (4.1)$$

where the deterministic part of the leakage can be written as a linear combination of a predetermined basis of functions $\mathcal{H} = \{h_1, \dots, h_m\}$, *i.e.*:

$$d(\mathbf{x}) = \sum_{i=1}^m \alpha_i \cdot h_i(\mathbf{x}) . \quad (4.2)$$

The choice of the basis of functions $\mathcal{H}$ is determined for each elementary operation routine depending on its internal variables. The basis should at least contain one function for each internal variable bit but might also include monomials of higher degrees due to possible coupling effects [FGM⁺18].

In our methodology, we suggest relying on linear regression in order to estimate the deterministic leakage $d(\cdot)$. It involves acquiring an initial set of ℓ_1 traces, which measure the leakage while executing the operation on ℓ_1 inputs generated uniformly at random. We then use this set to infer the coefficients $\{\alpha_i\}_{i=1,\dots,m}$. Subsequently, we can measure a new set of ℓ_2 traces on uniform random inputs to recover (an estimation of) the covariance matrix Σ .

It is worth emphasizing that while we propose linear regression for leakage estimation, our methodology remains adaptable to other estimation methods. Techniques such as template attacks [CRR03], combined with dimensionality reduction [SA08, CDSU23], and the emerging use of machine learning for side-channel analysis [MDP19, PPM⁺23], all offer viable alternatives. Furthermore, we assume that the engineering effort of optimizing the measurement setup has been performed properly, such that it is infeasible for an adversary to acquire more informative measurements than the ones used for the characterization. Indeed, the characterization steps of the methodology and in particular the estimation of the noise level depend heavily on the optimization of the experimental setup [BUS22, CM24].

Step 4: Enforcing / Relaxing Noise Independence

Next, we consider the noise independence assumption (Assumption 2) needed for the reduction from the noisy leakage to the random probing model. Namely, in the idealized noisy leakage model, the noise that occurs during the execution of an elementary operation is drawn independently of the noise that occurs during the execution of the previous ones. Hence, this assumption must be satisfied in practice. Meanwhile, it is hard to enforce and test since no clear separation of the noise occurs during a leakage trace. In our methodology, we propose a novel way to relax this assumption. Namely, we keep the Gaussianity hypothesis, but we allow the leakage of the different operations to overlap. We characterize this relaxation and directly reflect it on the security level by providing a reduction from the noisy leakage model with potential noise dependence to the idealized noisy leakage model.

Concretely, we propose to partition the noise distribution into multiple distributions, all while minimizing leakage during each operation. In simpler terms, given k consecutive elementary operations of inputs $\{(\mathbf{x}_i)\}_{1 \leq i \leq k}$, we can represent the overall leakage distribution as

$$\mathcal{Y} = \sum_{i=1}^k d_i(\mathbf{x}_i) + \mathcal{N}(\mathbf{0}, \Sigma) \quad (4.3)$$

where $d_i(\mathbf{x}_i)$ are the different deterministic signals of the operations, and the noise drawn from $\mathcal{N}(\mathbf{0}, \Sigma)$ is the global noise. Thanks to the data isolation enforcement and test, the deterministic signals are mutually data independent. Specifically, while the d_i 's might overlap on some time samples, they independently apply to the inputs $\mathbf{x}_i$. This ensures that the global deterministic leakage can be expressed as a sum in Equation 4.3.

In order to relax the noise independence, our approach consists of finding a set of covariance matrices $\{\Sigma_i\}_{i \in [k]}$ such that $\sum_{i \in [k]} \Sigma_i = \Sigma$. This way, we can split the Gaussian noise distribution $\mathcal{N}(\mathbf{0}, \Sigma)$ into k independent Gaussian distributions $\mathcal{N}(\mathbf{0}, \Sigma_1), \dots, \mathcal{N}(\mathbf{0}, \Sigma_k)$. This representation enables us to split the leakage distribution into several functions $\mathcal{Y}_i = d_i(\mathbf{x}_i) + \mathcal{N}(\mathbf{0}, \Sigma_i)$ for every $i \in \{1, \dots, k\}$. An adversary given a leakage sample of each $\mathcal{Y}_i$ is more powerful than an adversary given a sample of the global leakage $\mathcal{Y}$ because the former can always sum the $\mathcal{Y}_i$ samples to get a $\mathcal{Y}$ sample.

Instantiation. In our paper [BCM⁺25] (see Appendix G), we introduce several methods to define such matrices Σ_i , which sum to the original covariance matrix Σ while minimizing the mutual information between the leakage and the signals.

Step 5: Estimating the Noisy Leakage Parameter

Recall that once data isolation is enforced and tested, we can empirically characterize the leakage distribution of each elementary operation as described in Step 3. Assuming that the leakage takes the form of Equation 4.1, we compute the coefficients in the deterministic function for each elementary operation (as defined in Equation 4.2). Then, we can apply the noise relaxation described in Step 4 to get the covariance matrix of the Gaussian noise, which we suppose is the same for each elementary operation, following the representation in our optimization problem.

We can then compute the noisy leakage parameter δ related to the δ -noisy leakage model. As explained in Section 2.4.3, reducing the *idealized* noisy leakage model to the random probing model provides the leakage probability in the latter. More precisely, in order to achieve (δ, ε) -security in the idealized noisy leakage model, an abstract circuit should achieve (p, ε) -security in the random probing model with $p = \gamma \cdot \delta$ for some constant factor γ depending on the noisy leakage metric (*e.g.* $\gamma = |\mathcal{X}|$ for the Δ_{SD} metric, $\gamma = 1$ for the Δ_{MRE} metric). We describe the two main options:

- The original security reduction [DDF14] (DDF reduction) relies on the statistical distance between a (uniform) variable X and the same variable conditioned on its leakage Y : $\delta = \beta_{\Delta_{SD}}(X|Y)$. When reducing to the random probing model, this value is multiplied by the size of the input space $\mathcal{X}$ (*i.e.* the definition set of inputs $\mathbf{x}$ of an elementary operation), hence losing tightness in the tolerated leakage rate through the reduction. In this Δ_{SD} metric, $\gamma = |\mathcal{X}|$.
- In a more recent work [PGMP19] (PGMP reduction), the authors express δ using different noisiness metrics from the pointwise mutual information. From the security reduction viewpoint, the most interesting metric is the maximum relative error (MRE), a worst-case metric (just like the random probing model), contrary to the statistical distance, which is an average-case metric. When computing $\delta = \beta_{\Delta_{MRE}}(X|Y)$, the reduction to the random probing model yields tighter results with $\gamma = 1$ and thus $p = \delta$.

In order to estimate the noisy leakage parameter δ in our methodology, we compute both MRE and SD biases using the inferred leakage model to compare both reductions to the random probing model. Using the pointwise mutual information (Definition 5), we have

$$\begin{aligned} \beta_{\Delta_{MRE}} &= \sum_{Y=\mathbf{y}} P[Y=\mathbf{y}] \max_{X=\mathbf{x}} \left| \frac{P[X=\mathbf{x}, Y=\mathbf{y}]}{P[X=\mathbf{x}] \cdot P[Y=\mathbf{y}]} - 1 \right| \\ &= \sum_{Y=\mathbf{y}} P[Y=\mathbf{y}] \max_{X=\mathbf{x}} \left| \frac{P[Y=\mathbf{y}|X=\mathbf{x}]}{\sum_{X=\mathbf{x}'} P[Y=\mathbf{y}|X=\mathbf{x}']} \cdot \frac{1}{P[X=\mathbf{x}]} - 1 \right| \end{aligned} \quad (4.4)$$

and

$$\beta_{\Delta_{SD}} = \frac{1}{2} \sum_{Y=\mathbf{y}} P[Y=\mathbf{y}] \sum_{X=\mathbf{x}} \left| \frac{P[Y=\mathbf{y}|X=\mathbf{x}]}{\sum_{X=\mathbf{x}'} P[Y=\mathbf{y}|X=\mathbf{x}']} \cdot \frac{1}{P[X=\mathbf{x}]} - 1 \right| \quad (4.5)$$

From the equations above, we need to compute the sum of the conditional probabilities $P[Y=\mathbf{y}|X=\mathbf{x}']$ for $\mathbf{x}' \in \mathcal{X}$ for MRE and SD estimations. Then, in the case of MRE estimation, we need to find the maximal value of the expression given between $|\cdot|$ in Equation 4.4 over the values taken by X , while in the case of the SD estimation, we compute a sum over the expression given between $|\cdot|$ in Equation 4.5. Finally, we must compute the expected value for MRE and SD for Y .

Computing the conditional distribution. In order to estimate the conditional distribution $P[Y=\mathbf{y}|X=\mathbf{x}]$ given a leakage trace and $\mathbf{x}$, we can use the leakage characterization computed earlier, since the conditional distribution is known to be expressed as

$$P[Y=\mathbf{y}|X=\mathbf{x}] = \frac{\exp\left(-\frac{1}{2}(\mathbf{y}-d(\mathbf{x}))^T \Sigma^{-1}(\mathbf{y}-d(\mathbf{x}))\right)}{\sqrt{(2\pi)^n |\Sigma|}}$$

where n is the number of samples in $\mathbf{y}$ [CRR03].

Estimating the expected value. Since the range of Y is very large, we estimate both biases using a Monte Carlo integration method, as detailed below:

$$\beta_{\Delta_{\text{MRE}}} = \frac{\sum_{Y=\mathbf{y}} \max_{X=\mathbf{x}} \left| \frac{P[Y=\mathbf{y}|X=\mathbf{x}]}{\sum_{X=\mathbf{x}'} P[Y=\mathbf{y}|X=\mathbf{x}']} \cdot \frac{1}{P[X=\mathbf{x}]} - 1 \right|}{k},$$

$$\beta_{\Delta_{\text{SD}}} = \frac{\sum_{Y=\mathbf{y}} \sum_{X=\mathbf{x}} \left| \frac{P[Y=\mathbf{y}|X=\mathbf{x}]}{\sum_{X=\mathbf{x}'} P[Y=\mathbf{y}|X=\mathbf{x}']} \cdot \frac{1}{P[X=\mathbf{x}]} - 1 \right|}{2 \cdot k}.$$

In both equations, k denotes the number of leakage vectors $\mathbf{y}$ drawn from the leakage distribution $\mathcal{Y}$. Each leakage vector $\mathbf{y}$ is generated as $\mathbf{y} = d(\mathbf{x}) + \boldsymbol{\varphi}$, where $\mathbf{x}$ is an input generated uniformly at random, $d(\cdot)$ is the deterministic function for an elementary operation, and $\boldsymbol{\varphi}$ is generated from the Gaussian distribution $\mathcal{N}(0, \Sigma)$ following the noise covariance Σ . The MRE being a worst-case metric, it is more expensive to estimate than the (average-case) SD metric, which is the price to pay for the tighter reductions it enables.

Thanks to the law of large numbers, we know that as k approaches infinity, this estimation converges to the expected value. We show in [BCM⁺25] (see Appendix G) in our experimental results that the convergence curve reaches a plateau after some number of samples, which corresponds to the target value.

Step 6: Compiling the Cryptographic Implementation

At this stage, we have estimated the leakage parameter of each isolated noise-independent elementary operation. As defined in Section 4.2, this parameter can be computed using the SD or MRE metric. We obtain an equivalent leakage probability p in the random probing model in both cases by applying the reduction. As already mentioned, the reduction is tighter in the case of MRE, where the leakage probability in the random probing model can be directly set to the estimated noisy leakage parameter.

Different vs. maximum leakage probabilities. As we might obtain different noisy leakage parameters δ_i for the different elementary operations, leading to different leakage probabilities in the random probing model, we consider that all the gates leak with the maximum probability corresponding to maximum noisy metric $\delta = \max_i \delta_i$. A random probing secure circuit with maximum leakage probability is straightforwardly random probing secure with different leakage probabilities.

Gate vs. wire leakage model. In our characterization, we rely on the leakage of the elementary operations abstracted as gates in a circuit. Meanwhile, most random probing secure constructions suppose leakage on wires instead. Our methodology applies an additional transition from the gate to the wire leakage model to circumvent this issue. We can reduce the security of a circuit in the gate random probing model with leakage probability p , to the wire random probing model with leakage probability $\sqrt{p}$, assuming that each gate has at most two inputs.

Compiling a cryptographic implementation. The final step consists of a two-stage compilation process applied to the input arithmetic circuit C representing the target cryptographic implementation.

- One first applies the random probing secure compiler which, for the obtained leakage probability p and the target security level $\varepsilon = 2^{-\lambda}$, transforms C into a functionally equivalent randomized circuit $\hat{C}$ achieving (p, ε) -random probing security.
- One then serializes $\hat{C}$ into a physical implementation, making a sequence of calls to the (whitened) elementary operation routines on the target device. Each elementary operation in the sequence corresponds to a gate in $\hat{C}$ whose output is written in a fresh memory cell. The circuit wiring is hard-coded in the pointer arguments passed to the successive calls to the elementary operation routines.

The obtained physical implementation achieves λ bits of side-channel security for the target side-channel acquisition tool and physical device, and under relaxed or empirically verified physical assumptions. The overall process and provable security guaranty are wrapped up in [BCM⁺25] (see Appendix G).

4.3. Design Goals and Remaining Challenges

In [BCM⁺25] (see Appendix G), we demonstrate our methodology on an AES implementation running on a commercial off-the-shelf (COTS) ARM Cortex-M4 microcontroller, using a random probing compiler based on the expansion technique [BCP⁺20] recalled in Chapter 7. This case study surfaces three main challenges: (i) the intrinsically low-noise levels of the device, (ii) the performance overhead induced by costly isolation, and (iii) the non-tight security proofs of the masking scheme.

(i) The problem of insufficient noise on COTS microcontrollers is well known in the software masking literature [CDSU23, BS21, BCS21] and is therefore not specific to our methodology. Simple COTS devices typically exhibit low noise, which can compromise security (current random probing compilers struggle to produce implementations with reasonable security at high leakage rates p). More complex microcontrollers often exhibit higher noise, but achieving isolation on such platforms, especially when treated as black boxes, poses additional challenges. From a research perspective, the noise level is a significant constraint rather than a fundamental limitation, since one can design custom microcontrollers with higher noise levels (*e.g.* by integrating noise engines).

(ii) Our isolation technique of inserting “cleaning” instructions is suboptimal: it has a large performance cost ($\times 13$ to $\times 16$ overhead in the number of instructions) while being best-effort (and therefore heavily relying on empirical validation). Leveraging micro-architectural knowledge can facilitate achieving high-confidence and high-performance isolation. Further, strengthening the ISA definition to describe leakage [BGG⁺22] or even provide support for isolation [GMPP20, CPW24], could significantly simplify design, improve guarantees, and dramatically reduce the isolation performance overheads at a low hardware cost.

Note that an alternative solution for the noise and isolation challenges is to apply our methodology to hardware implementations. Indeed, such implementations have typically higher noise levels and perform operations in parallel. The approach for ensuring isolation (*i.e.* avoiding glitches, transitions, and couplings) will be different but is already well-understood [FGM⁺18, CS21].

(iii) Regarding the non-tightness of security proofs, the high noise requirements and the number of shares needed to achieve a given security level may seem excessive from a practitioner’s perspective, especially when confronted with the current state-of-the-art attacks or the security level recently proven for a single sharing [BCG⁺23]. Below, we identify several sources of non-tightness and suggest directions for improvement.

- In our experiments, we consider a masking scheme based on the expansion technique [BCP⁺20] recalled in Chapter 7. While this scheme had the state-of-the-art minimum noise level requirement at the time of this study, the scaling of the security level with respect to the leakage parameter p is sub-optimal: an optimal masking scheme with n shares would scale as $\mathcal{O}(p^n)$, while ours has a lower exponent [BRT21, CFOS21]. This issue can be solved by using tighter random probing security proofs such as the ones based on probe distribution tables [CFOS21] or our new proposals described in Chapters 6 and 8.
- The next step in the proof is the reduction of noisy leakage to random probing. Using the MRE metric over the SD metric is already a significant gain, as it avoids a field-size loss in the proof. However, in our experiments, the MRE worst-case metric is significantly larger than the SD, canceling part of the gain. This may be due to the low noise level, as adding noise reduces the MRE vs. SD gap. If the use of worst-case leakage metrics remains an issue on noisy devices, one potential solution is to use the reduction to the average random probing model [DFS15],

which would relax the noise requirements at the cost of a stronger security model (hence more complex masking compiler).

- Finally, the reduction from gate-probing to wire-probing introduces a quadratic loss in the random probing parameter. This loss appears to be an artifact of the proof and should be optimized in future work.

On the practical side, our methodology relies on well-defined and realistic physical assumptions. We propose concrete tests to validate some of these assumptions, while others are supported by physics-based arguments. Although our approach advances towards experimentally validated hypotheses, some assumptions (*e.g.* noise Gaussianity, isolation of consecutive operations implying isolation of non-consecutive ones) could benefit from more direct experimental validation. Further, due to the nature of statistical tests, quantitative assumptions (*e.g.* independence, isolation) can never be proven, only invalidated. A fully-proven approach would instead rely on quantitative variants of these assumptions, which could be proven with high confidence using statistical tests (*e.g.* using statistical power and effect size [WO19]). Finally, there is room for efficiency improvements, such as optimizing the noise splitting or reducing the data isolation performance overhead.

Part III.

Random Probing Composability

Chapter 5

Threshold Random Probing Composability

Contents

5.1. Motivation	26
5.2. Security Notion	27
5.3. Composition Results	28

5.1. Motivation

Exhaustive or generic proofs can be developed to establish the random probing security of small gadgets implementing atomic operations such as additions or multiplications (see *e.g.* in Chapter 3). However, when moving to more complex circuits, constructing security proofs becomes practically impossible to carry out manually with current generic theoretical methods, and often prohibitively complex when using existing automated verification tools. Consequently, a modular approach is typically adopted: small gadgets satisfying specific composability properties are combined to achieve a global security guarantee for the complete implementation.

Building on this idea, several composition frameworks have been proposed to ensure security in the random probing model [ISW03, Ajt11, DDF14, ADF16, GJR18]. Among the latter, the approach of Ananth, Ishai, and Sahai [AIS18] was the first to achieve and make explicit a constant leakage rate. In the same vein, our work on threshold Random Probing Composability (threshold RPC) [BCP⁺20] (Appendix A) introduces a new framework that enables secure composition in the random probing model, with a constant, explicit, and higher leakage rate, and a reduced complexity.

As illustrated in Figure 5.1, our framework makes it possible to construct large random probing secure circuits by assembling smaller gadgets that satisfy the threshold RPC property. This chapter is devoted to detailing this framework, with the next two sections focusing respectively on the formal definition of threshold RPC and on the composability results¹.



Figure 5.1.: Circuit $C : \llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell \rightarrow \llbracket \mathbf{y} \rrbracket \in (\mathbb{K}^n)^m$ decomposed into three gadgets satisfying the threshold RPC (tRPC) property, for global random probing security.

¹Throughout this chapter and the next ones, we will consider *randomized-wiring circuits*: the extension of randomized arithmetic circuits defined in Section 2.2 that allows randomization at the *wire* level (not only within arithmetic operations). This added generality is required for the gadgets and the compiler we construct in the next chapters.

5.2. Security Notion

Our notion of *threshold random probing composability* requires that the random probing leakage of a gadget's internal wires, along with any set of up to t output shares, can be perfectly simulated using only up to t input shares, except with a simulation failure probability ε .

Definition 10 ((Threshold) Random Probing Composability). *Let $n, \ell, m \in \mathbb{N}$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is (t, p, ε) -threshold random probing composable (threshold RPC) for some $t \in \mathbb{N}$ and $p, \varepsilon \in [0, 1]$ if there exists a stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every masked input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, J_2, \dots, J_m) \in \mathcal{P}([n])^m$ of cardinals $|J_1| \leq t, \dots, |J_m| \leq t$, the outputs of Experiment 1*

Experiment 1: RPC	
$\tau \leftarrow \$$	<i>Sampling the random tape for the wire configuration.</i>
$\mathcal{W} \leftarrow \text{LeakingWires}(G^\tau, p)$	<i>Sampling the leaking wires.</i>
$I_1, \dots, I_\ell \leftarrow \text{Sim}_1(G^\tau, \mathcal{W}, J_1, \dots, J_m)$	<i>From the indexes of the leaking outputs, Sim_1 determines the indexes of the required inputs.</i>
$out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket _{I_1}, \dots, \llbracket x_\ell \rrbracket _{I_\ell})$	<i>From the selected input shares, Sim_2 returns simulated values for the internal leaking wires out_1 and output leaking wires out_2</i>
return $(I_1, \dots, I_\ell, out_1, out_2)$	

are such that

$$1. \quad \Pr(|I_1| > t \vee \dots \vee |I_\ell| > t) \leq \varepsilon \quad (5.1)$$

2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\text{AssignWires}(G^\tau, R, \mathcal{W}, \llbracket \mathbf{x} \rrbracket)}_{\text{Internal random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right)$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(G^\tau, p)$ and R the uniform random variable representing the random tape for G 's random gates.

In the above definition, the first-pass simulator Sim_1 determines the necessary input shares $(I_1, \dots, I_\ell)$ for the second-pass simulator Sim_2 to produce a perfect simulation of the leaking wires defined by the set $\mathcal{W}$ together with the output shares defined by the collection of sets $\mathbf{J} = (J_1, \dots, J_m)$. Note that there always exists such a collection of sets $(I_1, \dots, I_\ell)$ since $([n], \dots, [n])$ trivially allows a perfect simulation whatever $\mathcal{W}$ and $\mathbf{J}$. However, the goal of Sim_1 is to return a collection of sets $(I_1, \dots, I_\ell)$ with cardinals at most t . The idea behind this constraint is to keep the following composition invariant: for each gadget we can achieve a perfect simulation of the leaking wires plus t shares of each output sharing from t shares of each input sharing.

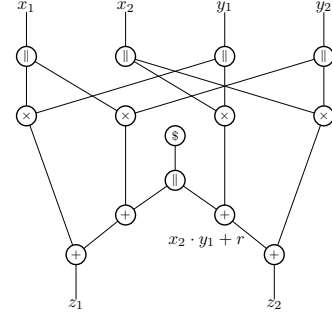
We shall call *failure event* the event that at least one of the sets $I_1, \dots, I_\ell$ output of Sim_1 has cardinality greater than t . It occurs whenever Sim_2 requires more than t shares of one input sharing to be able to produce a perfect simulation of the leaking wires (*i.e.* the wires with label in $\mathcal{W}$) together with the output shares in $\llbracket \mathbf{y} \rrbracket|_{\mathbf{J}}$. In that case, the composition invariant is broken. When (t, p, ε) -threshold RPC is achieved, the failure event probability is upper bounded by ε according to Equation 5.1.

Automatic Verification. The IronMask tool, briefly discussed in Section 3.3, also computes the attacker's advantage for the *threshold RPC* security notion. The overall workflow mirrors the evaluation used for random probing security, but with a key difference that it requires considering all possible tuples of t shares per output in the overall leakage and computing the worst-case failure probability, defined as the maximum over all these scenarios.

Example. Using IronMask, we can compute the coefficients c_i that fully determine the polynomial f for the circuit C_{ex} in the threshold RPC setting with $t = 1$:

$$(c_0, \dots, c_{21}) = (0, 4, 131, 1173, 5810, 20230, 54215, 116269, 203489, 293930, 352716, \\ 352716, 293930, 203490, 116280, 54264, 20349, 5985, 1330, 210, 21, 1).$$

This result directly yields the attacker's advantage in this setting by evaluating Equation 3.2 at any leakage rate $p \in [0, 1]$. Compared with the coefficients obtained for random probing security (see Section 3.3), we now observe that c_1 is strictly positive. This reflects that, when conditioning on an output share such as z_1 (the scenarios for both output shares are identical), a single additional leaking wire may already yield a simulation failure: a leakage on any of the three wires carrying r , or on the wire carrying $x_2 \cdot y_1 + r$, would require at least both shares of y for a perfect simulation of the pair made of this wire and z_1 .



5.3. Composition Results

Informally, the security of a composed circuit relies on the requirement that no simulation failures occur in any of the individual gadgets, as such failures would break the composition invariant. In practice, the idea is to compute the probability of such simulation failures in the entire circuit to determine the attacker's advantage.

Theorem 1 (Composition). *Let $t \in \mathbb{N}$ and $p, \varepsilon \in [0, 1]$. Any (randomized) arithmetic circuit C composed of $|C|$ (t, p, ε) -threshold RPC gadgets is $(p, |C| \cdot \varepsilon)$ -random probing secure.*

Proof. Let $\mathcal{W}$ denote the set of leaking wires in the randomized arithmetic circuit C , where each wire leaks independently with probability p . We construct a simulator Sim that, given C and $\mathcal{W}$, perfectly simulates the leakage except with probability at most $|C| \cdot \varepsilon$, leveraging the simulators associated with the (t, p, ε) -threshold RPC base gadgets:

1. We begin by partitioning $\mathcal{W}$ into $|C|$ disjoint subsets $\mathcal{W}_i$, for $i \in \{1, \dots, |C|\}$, where each $\mathcal{W}_i$ corresponds to the leakage of the random probing leakage sampler `LeakingWires` when applied to the i -th gadget G_i of C with leakage probability p .
2. The simulation proceeds recursively from the output gadgets towards the inputs. For each end gadget (whose outputs are circuit outputs), we run its simulator $\text{Sim}_1^{G_i}$ on $\mathcal{W}_i$ with an initially empty set of leaking outputs $J = \emptyset$, obtaining a set I of inputs necessary for consistent simulation.
3. Next, we consider the parent gadgets of these end gadgets, *i.e.*, those gadgets whose outputs serve as inputs to the end gadgets. For each such gadget G_i , we execute $\text{Sim}_1^{G_i}$ on $\mathcal{W}_i$, using as J the sets I collected from its child gadgets, yielding a new set of required inputs I . This process iteratively traverses the circuit from outputs to inputs, applying the Sim_1^G simulators to determine the leakage $\mathcal{W}_i$ and the corresponding I and J sets. The simulation fails if, for any gadget, the set I of required inputs exceeds t in cardinality. For a circuit with $|C|$ gadgets, this failure occurs with probability at most $1 - (1 - \varepsilon)^{|C|} \leq |C| \cdot \varepsilon$.
4. If no failure occurs, we complete the simulation by running the second part of the simulators, Sim_2^G , from inputs towards outputs, choosing random initial values $(x_i)_I$ to reconstruct consistent wire values throughout the circuit. This finalizes the construction of our global simulator Sim .

□

Chapter 6

(Uniformly) Cardinal Random Probing Composability

Contents

6.1. Motivation	29
6.2. Security Notions and Composition Results	31
6.2.1. Cardinal Random Probing Composability	31
6.2.2. Uniformly Cardinal Random Probing Composability	33
6.3. Comparison of Random Probing Composability Notions	36

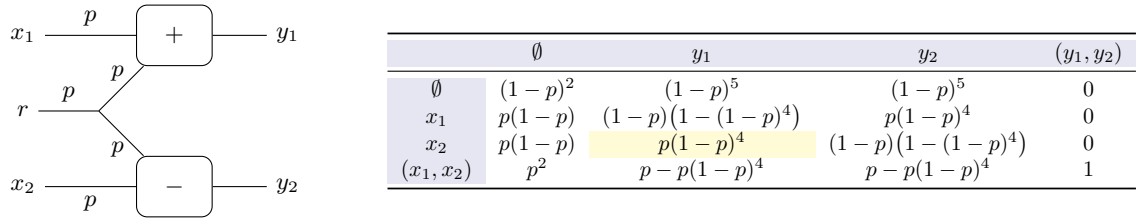
6.1. Motivation

The *(threshold) random probing composability* (threshold RPC) notion, that we introduced in [BCP⁺20] and recalled in Chapter 5, requires that the random probing leakage of a gadget’s internal wires, along with any set of up to t output shares, for some $0 \leq t \leq n - 1$, can be perfectly simulated using at most t input shares, except with a simulation failure probability ε . A key advantage of this notion is its *trivial composability*: any circuit composed of threshold RPC gadgets remains threshold RPC, with the overall failure probability upper bounded by the sum of the individual failure probabilities. However, this notion is not tight, since the need for more than t input shares to simulate the leakage of a single gadget does not necessarily imply failure in a composed setting.

To overcome this limitation, Cassiers et al. [CFOS21] introduced the notion of *general random probing composability* (general RPC). In this framework, the simulation failure probability is defined for each pair $(\mathcal{I}, \mathcal{J})$, where $\mathcal{I}$ is a set of input shares and $\mathcal{J}$ a set of output shares, as the probability that simulating $\mathcal{J}$ together with the internal leakage requires $\mathcal{I}$. These probabilities are gathered in so-called *probe distribution table* (PDT), also referred to as general RPC envelopes in this manuscript.

Example. An illustrative example is provided in Figure 6.1 for a very simple gadget G with two input shares $(x_1, x_2) \in \mathbb{K}^2$, a random value $r \in \mathbb{K}$, and two output shares $(y_1, y_2) \in \mathbb{K}^2$. Each cell in the table indicates the probability that the input shares listed in the corresponding row are necessary and sufficient to simulate the gadget’s internal leakage, together with the output shares specified in the column, given a leakage rate p . For instance, the probability of requiring exactly x_2 to simulate the internal leakage of G (among x_1, x_2 , and the three wires carrying r) together with y_1 (yellow cell in Figure 6.1) is given by the following factors:

- the probability that x_1 does not leak, since otherwise x_1 would be required for the simulation: $1 - p$;
- the probability that none of the wires carrying r leak, because otherwise x_1 would be

Figure 6.1.: Gadget $G : (x_1, x_2) \rightarrow (y_1 = x_1 + r, y_2 = x_2 + r)$ (left) and its PDT (right).

necessary to simulate (y_1, r) : $(1-p)^3$;

- the probability that x_2 does leak, as we want x_2 to be necessary for the simulation: p .

Multiplying these probabilities yields $p(1-p)^4$.

However, computing such tables or envelopes is exponential in the number of input and output shares, making the approach quickly intractable when composing even a small number of gadgets. Specifically, for a $(\ell$ -to- m , n -share) circuit, the general RPC envelope contains (or bounds) $2^{(\ell+m) \cdot n}$ distinct failure probabilities.

To mitigate this issue, we introduced the notion of *cardinal random probing composability* (cardinal RPC) in [BRR25] (Appendix F) as a trade-off between tightness and evaluation complexity. This approach characterizes the simulation failure probability based solely on the number of output shares to be simulated and the number of input shares required for the simulation, without considering the specific sets of shares. Consequently, a cardinal RPC envelope for a $(\ell$ -to- m , n -share) circuit consists of (bounds on) $(n+1)^{(\ell+m)}$ distinct failure probabilities. This complexity is polynomial in the number of shares n , rather than exponential as in the case of general RPC, although it remains exponential in the number of input and output sharings.

Example. An example of such a cardinal RPC envelope is given in Figure 6.2.

	0	1	2
0	$(1-p)^2$	$(1-p)^5$	0
1	$2p(1-p)$	$p(1-p)^4 + (1-p)(1-(1-p)^4)$	0
2	p^2	$p - p(1-p)^4$	1

Figure 6.2.: Cardinal RPC envelope of gadget G from Figure 6.1.

One step further, we introduced in [BRR25] the notion of *uniformly cardinal RPC*. Informally, a gadget is said to be uniformly cardinal RPC if it satisfies general RPC for an envelope collection in which, for any set of input shares of a given cardinality, the envelopes defined over all output share sets of the same cardinality are identical.

It is important to emphasize that uniformly cardinal RPC should not be seen as a method for computing the security level of a gadget. Rather, it is a structural property that characterizes gadgets whose leakage behaviour is sufficiently *balanced* so that cardinal RPC essentially captures all the relevant leakage information. When the general RPC envelope collection used corresponds to the tightest possible one, proving that a gadget is uniformly cardinal RPC with respect to this envelope collection effectively bridges the gap between cardinal RPC and general RPC for that gadget. In this case, we obtain the strong security guarantees of general RPC (relative to the tightest available envelope collection) while retaining the efficiency of verification offered by the cardinal RPC framework.

The uniformity condition thus ensures that no critical leakage information is overlooked beyond what is already encoded in the cardinal structure.

Example. The gadget illustrated in Figure 6.1 does not satisfy uniform cardinal RPC with respect to its tight general RPC envelope (shown on the right). This is because, for example, the probabilities of requiring x_1 or x_2 given y_1 as the output leakage are not equal, demonstrating an imbalance in the leakage behaviour.

In what follows, we first formalize the three security notions and present our main composition theorems for (uniformly) cardinal RPC. Then, we demonstrate on a concrete example (a butterfly network) how they compare to each other in terms of complexity and security.

6.2. Security Notions and Composition Results

We begin by formalizing the notion of general RPC from [CFOS21], which serves as the foundational security property. Intuitively, the idea is to provide a tight upper bound on the probability that a given set of input shares is both necessary and sufficient to simulate the leakage of a gadget along with certain well-defined output shares.

Definition 11 (General Random Probing Composability). *Let $n, \ell, m \in \mathbb{N}$, $p \in [0, 1]$. Let $\mathcal{E}^g$ be a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and defined over $\mathcal{P}([n])^\ell$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^g)$ -general random probing composable if there exists a stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket x \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \dots, J_m)$, the outputs of Experiment 1 (introduced for threshold RPC and that we recall here)*

Experiment 1: RPC	
$\tau \leftarrow \$$	Sampling the random tape for the wire configuration.
$\mathcal{W} \leftarrow \text{LeakingWires}(G^\tau, p)$	Sampling the leaking wires.
$I_1, \dots, I_\ell \leftarrow \text{Sim}_1(G^\tau, \mathcal{W}, J_1, \dots, J_m)$	From the indexes of the leaking outputs, Sim_1 determines the indexes of the required inputs.
$out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket _{I_1}, \dots, \llbracket x_\ell \rrbracket _{I_\ell})$	From the selected input shares, Sim_2 returns simulated values for the internal leaking wires out_1 and output leaking wires out_2
return $(I_1, \dots, I_\ell, out_1, out_2)$	

are such that

1. $(I_1, \dots, I_\ell) \lesssim \mathcal{E}_{J_1, \dots, J_m}^g$,
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\text{AssignWires}(G^\tau, R, \mathcal{W}, \llbracket x \rrbracket)}_{\text{Internal random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(G^\tau, p)$, $\llbracket y \rrbracket \leftarrow G^\tau(\llbracket x \rrbracket)$, and R the uniform random variable representing the random tape for G 's random gates.

6.2.1. Cardinal Random Probing Composability

As explained in Section 6.1, we introduced the notion of cardinal RPC in [BRR25] as a trade-off between general RPC, which does not scale well for complex circuits, and threshold RPC, which does not provide tight security bounds. Its formal definition is recalled right below.

Definition 12 (Cardinal Random Probing Composability). Let $n, \ell, m \in \mathbb{N}$, $p \in [0, 1]$. Let $\mathcal{E}^c$ be a collection of probability envelopes indexed by $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$ and defined over $[0, n]^\ell$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^c)$ -cardinal random probing composable if there exists a stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$, the outputs of Experiment 1 are such that

1. $(|I_1|, \dots, |I_\ell|) \lesssim \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^c$, where $t_1^{\text{out}} = |J_1|, \dots, t_m^{\text{out}} = |J_m|$, and
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\text{AssignWires}(G, R, \mathcal{W}, \llbracket \mathbf{x} \rrbracket)}_{\text{Internal random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$, $\llbracket \mathbf{y} \rrbracket \leftarrow G(\llbracket \mathbf{x} \rrbracket)$, and R the uniform random variable representing the random tape for G 's random gates.

We now explicitly derive the envelope for the cardinal RPC property of a gadget from its envelope for the general RPC property in Lemma 2. This follows directly from Definitions 11 and 12, using the same simulators for both definitions.

Lemma 2. Let $n, \ell, m \in \mathbb{N}$, $p \in [0, 1]$. Let $\mathcal{E}^g$ be a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and defined over $\mathcal{P}([n])^\ell$. Let $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ be an n -share gadget achieving $(p, \mathcal{E}^g)$ -general RPC. Then, G is also $(p, \mathcal{E}^c)$ -cardinal RPC with $\mathcal{E}^c$ defined for all $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$ and for all $(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \in [0, n]^\ell$, as follows:

$$\mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^c(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) = \max_{\substack{J_1, \dots, J_m \\ |J_1|=t_1^{\text{out}}, \dots, |J_m|=t_m^{\text{out}}}} \sum_{\substack{I_1, \dots, I_\ell \\ |I_1|=t_1^{\text{in}}, \dots, |I_\ell|=t_\ell^{\text{in}}}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell)$$

The next lemma constitutes the key building block for the composition of cardinal RPC gadgets.

Lemma 3. Let $n \in \mathbb{N}$, $p \in [0, 1]$, and $\mathcal{E}^1, \mathcal{E}^2$ two collections of $n+1$ probability envelopes over $[0, n]^2$. Let $G_1 : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ be a $(p, \mathcal{E}^1)$ -cardinal RPC gadget and $G_2 : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ be a $(p, \mathcal{E}^2)$ -cardinal RPC gadget. The sequential composition $G_3 : (\mathbb{K}^n)^3 \rightarrow \mathbb{K}^n$ of Figure 6.3 is $(p, \mathcal{E}^3)$ -cardinal RPC where $\mathcal{E}^3$ is a collection of probability envelopes such that

$$\forall \mathbf{t}_{\text{in}} = (t_1^{\text{in}}, t_2^{\text{in}}, t_3^{\text{in}}) \in [0, n]^3, \forall t^{\text{out}} \in [0, n], \quad \mathcal{E}_{t^{\text{out}}}^3(\mathbf{t}_{\text{in}}) = \sum_{i=0}^n \mathcal{E}_i^1(t_1^{\text{in}}, t_2^{\text{in}}) \cdot \mathcal{E}_{t^{\text{out}}-i}^2(t_3^{\text{in}}, i).$$

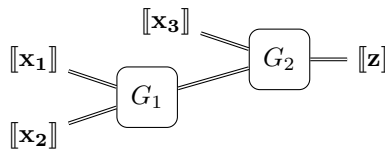


Figure 6.3.: Composition between $G_1 : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ and $G_2 : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$.

The full proof is given in [BRR25] (Appendix F); here, we only outline the main idea. The objective is to compute an upper bound on the probability distribution of the three input cardinalities, conditioned on the cardinality observed at the output. To this end, we first make explicit the shares that are required between G_1 and G_2 . This decomposition isolates their interaction, enabling us to treat the leakage events on G_1 and G_2 independently, except for this common dependency. Finally, we use worst-case estimates to connect the required shares to their cardinalities.

6.2.2. Uniformly Cardinal Random Probing Composability

As discussed previously, general RPC provides the tightest bounds on the adversarial advantage, but comes at the cost of exponential complexity when composing multiple gadgets. Threshold RPC, on the other hand, is highly efficient for composition but results in looser security bounds. Cardinal RPC offers a trade-off, sacrificing some tightness to gain compositional efficiency.

In [BRR25] (Appendix F), we observed that when a gadget's leakage is sufficiently balanced across both its input and output shares (a property we refer to as uniformly cardinal RPC), it achieves an even better trade-off. Intuitively, for a uniformly cardinal RPC gadget, the cardinal RPC envelope captures all the leakage information contained in the original general RPC envelope, while still benefiting from the reduced verification complexity of the cardinal RPC framework. Thus, it effectively combines the best of both worlds: the tight security guarantees of general RPC and the efficiency of cardinal RPC.

In what follows, we decompose the notion of uniformly cardinal RPC into two jointly equivalent sub-properties: *UI cardinal RPC* and *UO cardinal RPC*, which separately capture the effect of uniformity on the input and output shares, respectively. We do not restate the formal definition of uniformly cardinal RPC here, as it follows naturally from these two notions: Lemma 4 states that uniformly cardinal RPC holds if and only if both UI cardinal RPC and UO cardinal RPC hold. The proof, given in [BNR25] (Appendix H), follows directly from the definitions. This decomposition is motivated by the observation that a gadget may satisfy none, exactly one, or both of these sub-properties, with each case having different consequences for composability.

Definition 13 (UI cardinal RPC). *Let $n, \ell, m \in \mathbb{N}$, $p \in [0, 1]$. Let $\mathcal{E}^{ui}$ be a collection of probability envelopes indexed by $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$ over $[0, n]^\ell$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^{ui})$ -UI cardinal random probing composable if it is $(p, \mathcal{E}^{ui})$ -cardinal RPC and if it is $(p, \mathcal{E}^g)$ -general RPC such that the collection of general RPC envelopes $\mathcal{E}^g$ satisfies the two following conditions:*

1. $\forall (J_1, \dots, J_m) \in \mathcal{P}([n])^m$, $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$, $(I'_1, \dots, I'_\ell) \in \mathcal{P}([n])^\ell$ such that $(|I_1|, \dots, |I_\ell|) = (|I'_1|, \dots, |I'_\ell|)$, we have

$$\mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \mathcal{E}_{J_1, \dots, J_m}^g(I'_1, \dots, I'_\ell),$$

2. $\forall (t_1^{in}, \dots, t_\ell^{in}) \in [0, n]^\ell$, $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$, $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ such that $|I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}$, we have

$$\max_{\substack{J_1, \dots, J_m \\ |J_1| = t_1^{out}, \dots, |J_m| = t_m^{out}}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{in}} = \mathcal{E}_{t_1^{out}, \dots, t_m^{out}}^{ui}(t_1^{in}, \dots, t_\ell^{in}).$$

Definition 14 (UO cardinal RPC). *Let $n, \ell, m \in \mathbb{N}$, $p \in [0, 1]$. Let $\mathcal{E}^{uo}$ be a collection of probability envelopes indexed by $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$ over $[0, n]^\ell$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^{uo})$ -UO cardinal random probing composable if it is $(p, \mathcal{E}^{uo})$ -cardinal RPC and if it is $(p, \mathcal{E}^g)$ -general RPC such that the collection of general RPC envelopes $\mathcal{E}^g$ satisfies the two following conditions:*

1. $\forall (J_1, \dots, J_m) \in \mathcal{P}([n])^m$, $(J'_1, \dots, J'_m) \in \mathcal{P}([n])^m$ such that $(|J_1|, \dots, |J_m|) = (|J'_1|, \dots, |J'_m|)$, for all $(t_1^{in}, \dots, t_\ell^{in}) \in [0, n]^\ell$, we have

$$\sum_{\substack{I_1, \dots, I_\ell \\ |I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \sum_{\substack{I_1, \dots, I_\ell \\ |I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}}} \mathcal{E}_{J'_1, \dots, J'_m}^g(I_1, \dots, I_\ell).$$

2. $\forall (t_1^{in}, \dots, t_\ell^{in}) \in [0, n]^\ell$, $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$, $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ such that $|J_1| = t_1^{out}, \dots, |J_m| = t_m^{out}$,

$$\sum_{\substack{I_1, \dots, I_\ell \\ |I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \mathcal{E}_{t_1^{out}, \dots, t_m^{out}}^{uo}(t_1^{in}, \dots, t_\ell^{in}).$$

Lemma 4. Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^u, \mathcal{E}^{ui}, \mathcal{E}^{uo}$ be collections of probability envelopes indexed by $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$ over $[0, n]^\ell$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^u)$ -uniformly cardinal RPC for some $p \in [0, 1]$ if and only if G is $(p, \mathcal{E}^{ui})$ -UI cardinal RPC and $(p, \mathcal{E}^{uo})$ -UO cardinal RPC with $\mathcal{E}^u = \mathcal{E}^{ui} = \mathcal{E}^{uo}$.

6.2.2.1. Tree UI Composition

As discussed earlier, if a $(p, \mathcal{E}^G)$ -general RPC gadget G also satisfies the uniform cardinal RPC property on the envelope collection $\mathcal{E}^G$, then the corresponding uniformly cardinal RPC envelope collection $\mathcal{E}^u$ carries equivalent information in terms of security, while making it possible to verify its composition much more efficiently.

More specifically, we now show that the UI cardinal RPC property alone is already sufficient to achieve tighter security bounds in certain composition settings. Concretely, we consider a gadget constructed as the parallel composition of two gadgets, G_1 and G_2 , which share a common input sharing $\llbracket x \rrbracket$ (as depicted in Figure 6.4). We refer to this type of construction as a tree-UI gadget. To simulate the overall leakage, shares of $\llbracket x \rrbracket$ are needed both for G_1 and G_2 . The key observation is that if at least one of these gadgets, say G_2 , satisfies the UI cardinal RPC property, then the number of shares required to simulate the combined leakage can be tightly bounded, as formalized in Lemma 5. In contrast, if neither gadget satisfies UI cardinal RPC, the estimation becomes looser: in the worst case, the number of required shares must be treated as the sum of those needed for each individual gadget, potentially up to the full number of shares n .

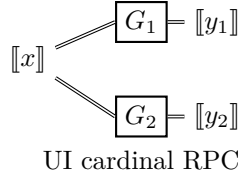


Figure 6.4.: Tree-UI gadget built from a cardinal RPC gadget G_1 and a UI cardinal RPC gadget G_2 .

Lemma 5 (Cardinal RPC of Tree-UI). Let $\llbracket x \rrbracket$ a n -sharing of a secret value $\mathbf{x} \in \mathbb{K}$. Let $G_1 : \mathbb{K}^n \rightarrow \mathbb{K}^n$ and $G_2 : \mathbb{K}^n \rightarrow \mathbb{K}^n$ two n -share gadgets. Let $p \in [0, 1]$. If G_1 is $(p, \mathcal{E}^1)$ -cardinal RPC and G_2 is $(p, \mathcal{E}^{ui2})$ -UI cardinal RPC, then the Tree-UI circuit described in Figure 6.4 is $(p, \mathcal{E})$ -cardinal RPC such that for all $t^{in} \in [0, n]$ and all $(t_1^{out}, t_2^{out}) \in [0, n]^2$ we have

$$\mathcal{E}_{t_1^{out}, t_2^{out}}(t^{in}) = \sum_{i_1=0}^{t^{in}} \sum_{i_2=t^{in}-i_1}^{t^{in}} \frac{\binom{i_1}{i_1+i_2-t^{in}} \cdot \binom{n-i_1}{t^{in}-i_1}}{\binom{n}{i_2}} \cdot \mathcal{E}_{t_1^{out}}^1(i_1) \cdot \mathcal{E}_{t_2^{out}}^{ui2}(i_2)$$

Proof. We use the property of UI cardinal RPC gadgets for which the sets of input shares required for the simulation are equiprobable for equal cardinals. Specifically, we can compute the probability that two sets of input shares of sizes i and j , required to simulate the leakage of both refresh gadgets and their outputs, represent t^{in} input shares. The number of common elements between both sets follows an hypergeometric law, i.e. the probability to have $i + j - t^{in}$ common elements is given by

$$\frac{\binom{i}{j-(t^{in}-i)} \cdot \binom{n-i}{t^{in}-i}}{\binom{n}{j}},$$

hence the final result. $\square$

In our applications, we adopt a conservative model: whenever a sharing is duplicated to feed two distinct gadgets, we insert an explicit *copy* gate and count *three* potential leakages of that sharing:

one for the duplication and one for each of the two uses. In this setting, the envelope of Tree-UI, corresponding to Lemma 5 and denoted by $\mathcal{E}^{\text{UI}}$, takes the form

$$\mathcal{E}_{t_1^{\text{out}}, t_2^{\text{out}}}(t^{\text{in}}) = \sum_{i=0}^{t^{\text{in}}} \mathcal{E}_{t_1^{\text{out}}, t_2^{\text{out}}}^{\text{UI}}(i) \cdot \binom{n-i}{t^{\text{in}}-i} \cdot p^{t^{\text{in}}-i} \cdot (1-p)^{n-t^{\text{in}}}.$$

If copy gates are not explicitly taken into account, *i.e.* if duplications are modeled only by gadgets with fan-out 2, then we can rely directly on Lemma 5, or interpret the above expression, as well as our subsequent empirical results, as upper bounds.

6.2.2.2. Uniformly RPC Gadgets from Composition with Permutations

In [BNR25] (Appendix H), we demonstrate that all the basic gadgets (specifically the refresh, addition, and scalar multiplication gadgets introduced in [BRR25, BNR25]) are inherently UO cardinal RPC, directly from their tight general RPC envelopes.

In the general case, however, a gadget's collection of RPC envelopes may fail to satisfy the UI, UO, or uniform cardinal RPC properties. In such situations, we can nevertheless obtain uniformly cardinal envelopes by a suitable relaxation process. More precisely, any $(p, \mathcal{E}^G)$ -general RPC gadget G with envelope collection $\mathcal{E}^G$ is trivially a $(p, \mathcal{E}^u)$ -uniformly RPC gadget by considering the following:

- A looser collection of general RPC envelopes $\mathcal{E}^{G'}$, where for all $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$, the envelopes satisfy

$$\mathcal{E}_{J_1, \dots, J_m}^{G'}(I_1, \dots, I_\ell) = \max_{\substack{(J'_1, \dots, J'_m) \in \mathcal{P}([n])^m, (I'_1, \dots, I'_\ell) \in \mathcal{P}([n])^\ell \\ |J_1| = |J'_1|, \dots, |J_m| = |J'_m|, |I_1| = |I'_1|, \dots, |I_\ell| = |I'_\ell|}} \mathcal{E}_{J'_1, \dots, J'_m}^G(I'_1, \dots, I'_\ell)$$

thereby relaxing the envelope structure while ensuring the uniform cardinal properties.

- A collection of uniform cardinal RPC envelopes $\mathcal{E}^u$ for all $(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \in [0, n]^\ell$, $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$, and $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ such that $|J_1| = t_1^{\text{out}}, \dots, |J_m| = t_m^{\text{out}}$, and $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ with $|I_1| = t_1^{\text{in}}, \dots, |I_\ell| = t_\ell^{\text{in}}$, defined as

$$\mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^u(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) = \mathcal{E}_{J_1, \dots, J_m}^{G'}(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{\text{in}}}.$$

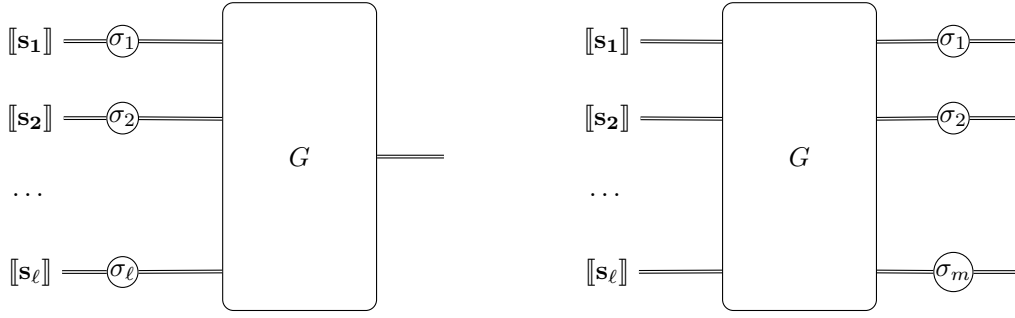
This ensures that the transformed envelope collection $\mathcal{E}^u$ retains all the information from $\mathcal{E}^{G'}$. Similar results can be derived for the UI and UO cardinal RPC notions.

Beyond this relaxation technique, we also introduce a more impactful approach to turn a general RPC gadget into a uniformly, UI or UO cardinal RPC gadget. Specifically, Lemma 6 shows that applying an independent uniformly random permutation to the shares of each input sharing is sufficient to ensure the UI cardinal RPC property and that independently permuting the shares of each output sharing guarantees the UO cardinal RPC property. As a direct corollary, applying independent uniformly random permutations to both the input and output shares of a gadget is sufficient to ensure the uniform cardinal RPC property.

Lemma 6. *Let $n, \ell, m \in \mathbb{N}$, $p \in [0, 1]$ and $\mathcal{E}^G$ represent a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$. Let $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ be an n -share $(p, \mathcal{E}^G)$ -general RPC gadget.*

- *Let $G \circ \sigma : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ denote the composition of G with σ , the concatenation of n independent random permutations $(\sigma_1, \dots, \sigma_\ell) \in \mathcal{S}_n^\ell$ on the ℓ input n -sharings, as depicted in Figure 6.5 (left). Then $G \circ \sigma$ is $(p, \mathcal{E}^{G \circ \sigma})$ -general RPC with, for all $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$,*

$$\mathcal{E}_{J_1, \dots, J_m}^{G \circ \sigma}(I_1, \dots, I_\ell) = \frac{1}{\prod_{i=1}^{\ell} \binom{n}{|I_i|}} \cdot \sum_{\substack{I_1^*, \dots, I_\ell^* \subseteq [n] \\ |I_1^*| = |I_1|, \dots, |I_\ell^*| = |I_\ell|}} \mathcal{E}_{J_1, \dots, J_m}^G(I_1^*, \dots, I_\ell^*) \quad (6.1)$$

Figure 6.5.: Gadget G with permutations on the inputs (left) or on the outputs (right).

In particular, $G \circ \sigma$ is $(p, \mathcal{E}^{ui})$ -UI cardinal RPC with the envelopes $\mathcal{E}^{ui}$ defined as follows. For any $(t_1^{in}, \dots, t_\ell^{in}) \in [0, n]^\ell$ and $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$,

$$\mathcal{E}_{t_1^{out}, \dots, t_m^{out}}^{ui}(t_1^{in}, \dots, t_\ell^{in}) = \max_{\substack{J_1, \dots, J_m \\ |J_1| = t_1^{out}, \dots, |J_m| = t_m^{out}}} \mathcal{E}_{J_1, \dots, J_m}^{G \circ \sigma}(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{in}}$$

for any $I_1, \dots, I_\ell$ where $|I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}$.

- Let $\sigma \circ G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ denote the composition of G with σ , the concatenation of n independent random permutations $(\sigma_1, \dots, \sigma_m) \in \mathcal{S}_n^m$ on the m output n -sharings, as depicted in Figure 6.5 (right). Then $\sigma \circ G$ is $(p, \mathcal{E}^{\sigma \circ G})$ -general RPC with, for all $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$,

$$\mathcal{E}_{J_1, \dots, J_m}^{\sigma \circ G}(I_1, \dots, I_\ell) = \frac{1}{\prod_{i=1}^m \binom{n}{|J_i|}} \cdot \sum_{\substack{J_1^*, \dots, J_m^* \subseteq [n] \\ |J_1^*| = |J_1|, \dots, |J_m^*| = |J_m|}} \mathcal{E}_{J_1^*, \dots, J_m^*}^G(I_1, \dots, I_\ell) \quad (6.2)$$

In particular, $\sigma \circ G$ is $(p, \mathcal{E}^{uo})$ -UO cardinal RPC with the envelopes $\mathcal{E}^{uo}$ defined as follows. For any $(t_1^i, \dots, t_\ell^i) \in [0, n]^\ell$ and $(t_1^o, \dots, t_m^o) \in [0, n]^m$,

$$\mathcal{E}_{t_1^o, \dots, t_m^o}^{uo}(t_1^i, \dots, t_\ell^i) = \sum_{\substack{I_1, \dots, I_\ell \\ |I_1| = t_1^i, \dots, |I_\ell| = t_\ell^i}} \mathcal{E}_{J_1, \dots, J_m}^{\sigma \circ G}(I_1, \dots, I_\ell).$$

To apply an independent uniformly random permutation to the shares, we use the Knuth-Yates shuffling technique, recalled in [BNR25] (Appendix H). Informally, this process consists of iteratively swapping a randomly chosen *unprocessed* share with the last *unprocessed* share in the list. The chosen share is then marked as processed, reducing the pool of candidates for the next iteration. This continues until all shares have been processed, ensuring a uniform random permutation of the entire set. The whole procedure requires generating $n - 1$ random values, each drawn uniformly from a decreasing range $[i]$, for $2 \leq i \leq n$. Additionally, it may involve up to $3(n - 1)$ read/write operations.

6.3. Comparison of Random Probing Composability Notions

We now compare the simulation-based composability notions recalled in this manuscript, with their respective strengths and limitations in terms of security tightness and verification efficiency.

- *General RPC* yields the tightest upper bound on the adversary's advantage, but at high cost: its probability envelope collection lie in $\mathcal{P}([n])^\ell \times \mathcal{P}([n])^m$, of size $2^{n(\ell+m)}$, which is quickly intractable for large numbers of shares, inputs, or outputs.

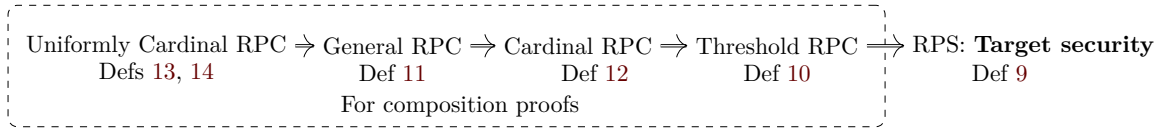


Figure 6.6.: Schematic implications between security notions

- *Cardinal RPC* provides a looser upper bound than *general RPC*, but significantly improves verification time and memory complexity. Envelopes are defined in $[0, n]^\ell \times [0, n]^m$, of size $(n+1)^{\ell+m}$ (polynomial in the number of shares n), so larger values of n become tractable.
- *Uniformly cardinal RPC*, obtained by averaging general RPC over uniformly random permutations of the shares, bridges the two: it preserves the tightness of *general RPC* while retaining the (polynomial) verification cost of *cardinal RPC*.
- *Threshold RPC* yields the loosest upper bound but incurs the least verification cost, making it attractive for large circuits or high number of shares.

Figure 6.6 provides a schematic overview of the logical relationships between the notions; the corresponding formal implications are stated in [BRR25] (Appendix F). To complement this global picture with a concrete benchmark, we analyze a *butterfly network*, a circuit structure commonly used in lattice-based cryptographic algorithms such as the Number Theoretic Transform (NTT). A butterfly network is made of multiple interconnected butterfly operations, each combining two data elements through a single addition, a single subtraction, and a multiplication by a known constant. In what follows, we consider a one-stage butterfly network, corresponding to a single butterfly operation.

One-Stage Butterfly Network. As shown in Figure 6.7, the circuit takes two masked inputs and produces two masked outputs, each consisting of n shares. It is composed of three basic gadgets:

- A scalar multiplication, implemented as the composition of a refreshing gadget and a multiplication of each share by the constant. The refresh gadget we use is cardinal RPC and will be described in Section 8.1 (Chapter 8).
- An addition, constructed by first refreshing both inputs using the same cardinal RPC refresh as for the scalar multiplication, followed by a sharewise addition.
- A subtraction, which produces masking envelopes identical to the addition gadget.

We evaluate the (t, p) -threshold RPC security (see Definition 10) of this butterfly network circuit for various numbers of shares, with $t = \lfloor \frac{n}{2} \rfloor$ and leakage rates $p \in \{2^{-3}, 2^{-5}, 2^{-10}\}$. This evaluation is performed using four different security analyses: (i) the general RPC envelopes; (ii) the UI cardinal RPC envelopes (assuming composition with uniformly random permutations); (iii) the cardinal RPC envelopes; and (iv) the *threshold RPC* bounds derived from the individual gadgets¹.

Initially, we observe that the circuit provides no security guarantees under the threshold RPC notion. This is because threshold RPC requires dedicated copy gadgets to securely duplicate sharings. Without them, duplicating a sharing would amount to summing two independent sets of t input

¹For the general RPC, the UI cardinal RPC and the cardinal RPC composition scenarios, we derive the threshold RPC advantage from the final circuit envelope to provide a consistent comparison.

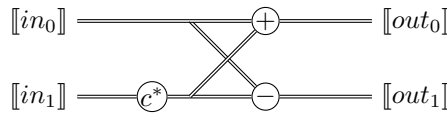


Figure 6.7.: 1-stage butterfly network.

shares, which exceeds the threshold and breaks the security guarantee. To enable a fair comparison, we therefore consider a modified version of the circuit (v2) in which a copy gadget (simply consisting in refreshing each branch) is inserted wherever a sharing is duplicated to feed multiple gadgets.

Figure 6.8 compares the threshold RPC advantages of the 1-stage butterfly network, computed under the different composition methods and circuit variants. In each subfigure, **gRPC**, **ucRPC**, **cRPC** correspond to the use of general, uniformly cardinal, and cardinal envelopes, respectively, to evaluate the global threshold RPC security. **tRPC** refers to the threshold RPC advantage, directly computed from the threshold RPC advantages of the individual gadgets. Labels **v1** to **v4** refer to: the original circuit (v1), the version with refresh gadgets at every copy (v2), the original circuit with permutations at each copy (v3), and the version combining permutations and additional refreshes (v4).

Across all leakage rates p , we confirm that the threshold RPC composition yields weaker advantage even when additional refreshes are inserted (see, *e.g.* **tRPC_v2** vs. **cRPC_v1**). Interestingly, uniformly cardinal RPC is always as good as general RPC (with or without additional refreshes) and sometimes even better (see **uRPC_v3** vs. **gRPC_v1**) thanks to the use of permutations. We observe that uniformly cardinal RPC consistently achieves the best security, followed closely by general RPC, with cardinal RPC trailing behind, with the difference becoming more significant for high leakage rates. Moreover, the impact of using refresh gadgets (variants v2 and v4) also becomes evident, leading to improved security levels compared to the original circuit (variants v1 and v3).

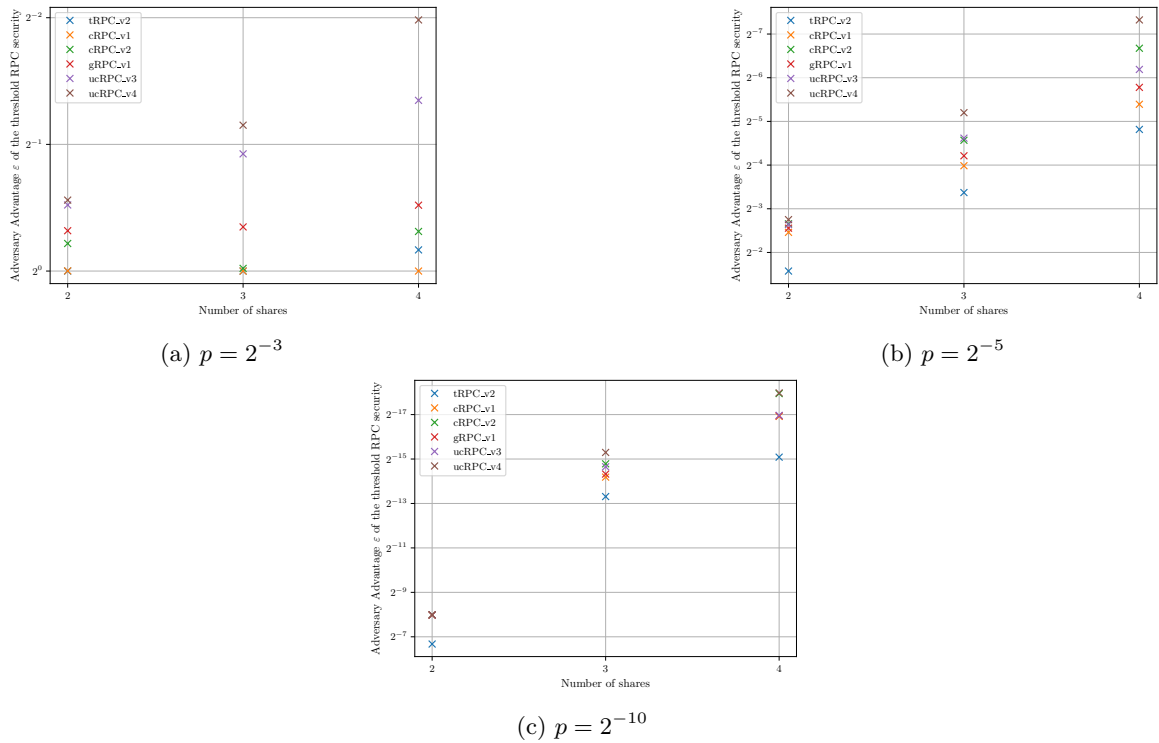


Figure 6.8.: Threshold RPC security of 1-stage butterfly network with respect to the number of shares n and the leakage rates p for different composition approaches.

In [BNR25] (Appendix H), we perform the same tests on a two-stage butterfly network, but this time without the general RPC notion which does not scale. The conclusions align with those drawn for the one-stage butterfly network, though the differences between variants (v1 and v3 versus v2 and v4, which include additional refreshes) are more pronounced. Overall, the results further highlight the benefits of using the composition method grounded in the uniformly cardinal RPC framework.

Part IV.

**Design of Random Probing
Composable Gadgets**

Chapter 7

Threshold RPC Gadgets from RPE Framework

Contents

7.1. RPE Framework	40
7.2. RPE Gadgets and Application to AES	43

In Part III, we presented several composition strategies for achieving global random probing security in a circuit, under the assumption that each individual gadget satisfies certain security properties. In this chapter, we focus on the design of gadgets within the threshold RPC framework of Chapter 5. Recall that in this framework, the overall circuit is (p, ε) -random probing secure if each individual gadget G_i is (p, ε_i) -threshold RPC such that

$$\varepsilon \leq 1 - \prod_i (1 - \varepsilon_i). \quad (7.1)$$

To construct threshold RPC gadgets with the smallest possible adversarial advantage, we revisit the gadget expansion paradigm introduced by Ananth, Ishai, and Sahai for building random probing secure circuit compilers [AIS18]. This leads to our *expanding compiler framework*, which constructs random probing secure circuits from small base gadgets [BCP⁺20, BRT21, BRTV21].

To formalize this framework, we introduce the notion of Random Probing Expandability (RPE). We demonstrate that any base gadget satisfying the RPE property can be securely integrated into the expanding compiler to construct a functionally equivalent gadget with an arbitrarily low threshold RPC advantage. Then, obtaining a set of base gadgets with arbitrarily small threshold RPC advantages directly enables the construction of circuits that achieve random probing security with an arbitrarily low overall advantage, as recalled in Equation 7.1.

7.1. RPE Framework

The basic principle of the gadget expansion strategy is as follows. Assume we have three n -share gadgets G_{add} , G_{mult} , G_{copy} and denote CC the standard circuit compiler for these base gadgets. We can derive three new n^2 -share gadgets by simply applying CC to each gadget: $G_{\text{add}}^{(2)} = \text{CC}(G_{\text{add}})$, $G_{\text{mult}}^{(2)} = \text{CC}(G_{\text{mult}})$ and $G_{\text{copy}}^{(2)} = \text{CC}(G_{\text{copy}})$. This process simply consists in replacing each addition gate in the original gadget by G_{add} , each multiplication gate by G_{mult} and each copy gate by G_{copy} , and by replacing each wire by n wires carrying a sharing of the original wire. Doing so, we obtain n^2 -share gadgets for the addition, multiplication and copy on $\mathbb{K}$. This process can be iterated an arbitrary number of times, say k , to an input circuit C :

$$C \xrightarrow{\text{CC}} \hat{C}_1 \xrightarrow{\text{CC}} \dots \xrightarrow{\text{CC}} \hat{C}_k.$$

The first output circuit $\hat{C}_1$ is the original circuit in which each gate is replaced by a base gadget G_{add} , G_{mult} or G_{copy} . The second output circuit $\hat{C}_2$ is the original circuit C in which each gate is replaced by an n^2 -share gadget $G_{\text{add}}^{(2)}$, $G_{\text{mult}}^{(2)}$ or $G_{\text{copy}}^{(2)}$ as defined above. Equivalently, $\hat{C}_2$ is the circuit $\hat{C}_1$ in which each gate is replaced by a base gadget. In the end, the output circuit $\hat{C}_k$ is hence the original circuit C in which each gate has been replaced by a k -expanded gadget and each wire as been replaced by n^k wires carrying an (n^k) -linear sharing of the original wire. The underlying compiler is called *expanding circuit compiler* which is formally defined hereafter.

Definition 15 (Expanding Circuit Compiler). *Let CC be the standard circuit compiler with sharing order n and base gadgets G_{add} , G_{mult} , G_{copy} . The expanding circuit compiler with expansion level k and base compiler CC is the circuit compiler $(\text{CC}^{(k)}, \text{Enc}^{(k)}, \text{Dec}^{(k)})$ satisfying the following:*

1. *The input encoding $\text{Enc}^{(k)}$ is an (n^k) -linear encoding.*
2. *The output decoding $\text{Dec}^{(k)}$ is the (n^k) -linear decoding mapping.*
3. *The circuit compilation is defined as*

$$\text{CC}^{(k)}(\cdot) = \underbrace{\text{CC} \circ \text{CC} \circ \dots \circ \text{CC}}_{k \text{ times}}(\cdot)$$

The goal of the expansion strategy in the context of random probing security is to replace the leakage probability p of a wire in the original circuit by the failure probability ε in the subsequent gadget simulation. When such a simulation fails, the simulator requires access to all input shares of the gadget, which corresponds to leaking the corresponding wire value in the base case. Security amplification is therefore achieved when $\varepsilon < p$, meaning that the probability of a simulation failure is smaller than the original leakage probability. More generally, if the failure probability can be upper bounded by a function of the leakage probability, namely $\varepsilon < f(p)$ for all $p \in [0, p_{\max}]$ for some $p_{\max} < 1$, then the expanding circuit compiler with expansion level k shall result in a security amplification as

$$p = \varepsilon_0 \xrightarrow{f} \varepsilon_1 \xrightarrow{f} \dots \xrightarrow{f} \varepsilon_k = f^{(k)}(p),$$

which leads to exponential security for suitable functions (e.g. $f : p \mapsto p^2$). This motivates the notion of maximum *tolerated leakage probability*, defined as the largest value p for which the inequality $f(p) < p$ still holds.

To get such amplification, gadgets must satisfy a stronger property than the threshold random probing composability introduced in Definition 10. This stronger requirement is known as *random probing expandability* (RPE). Recall that in the threshold RPC framework, a failure event occurs when more than t input shares are needed to perfectly simulate the leakage of a gadget. For a gadget to be expandable, two additional conditions must be met:

1. **Independence of failure across inputs:** In two-input gadgets, the probability of failure must be independent for each input. This mirrors the assumption that each wire in the base circuit leaks independently.
2. **Simulation recovery under failure:** When a failure occurs in a child gadget, the global simulator must be able to produce a perfect simulation of the full output (that is the full input for which the failure occurs). To make this possible, the simulator is given the clear output (obtained from the simulation of the base case), along with any subset of $n - 1$ output shares. This means that whenever the set J is of cardinal greater than t , the gadget simulator can replace it by any set J' of cardinal $n - 1$.

We now recall the formal definition of the RPE property for two-input, one-output gadgets. It can be simply extended to gadgets with two outputs and/or a single or multiple inputs. We refer the reader to [BCP⁺20] (Appendix A) for the formal definitions of these variants.

Definition 16 (Random Probing Expandability). *Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -random probing expandable (RPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every masked inputs $(\llbracket x \rrbracket, \llbracket y \rrbracket) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \in \mathcal{P}([n])$ and for every $p \in [0, 1]$, the outputs of Experiment 2*

Experiment 2: RPE	
$\mathcal{W} \leftarrow \text{LeakingWires}(G, p)$	<i>Sampling the leaking wires.</i>
$(I_1, I_2, J') \leftarrow \text{Sim}_1(\mathcal{W}, J)$	<i>From the indexes of the leaking outputs, Sim_1 determines the indexes of the required inputs and J'.</i>
$\text{out} \leftarrow \text{Sim}_2^G(\mathcal{W}, J', \llbracket x \rrbracket _{I_1}, \llbracket y \rrbracket _{I_2})$	<i>From the selected input shares and J', Sim_2 returns simulated values for the leaking wires</i>
return $(I_1, I_2, J', \text{out})$	

are such that

1. the failure events $\mathbb{F}_1 \equiv (|I_1| > t)$ and $\mathbb{F}_2 \equiv (|I_2| > t)$ verify

$$\Pr(\mathbb{F}_1) = \Pr(\mathbb{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = \varepsilon^2 \quad (7.2)$$

with $\varepsilon = f(p)$ (in particular $\mathbb{F}_1$ and $\mathbb{F}_2$ are mutually independent),

2. J' is such that $J' = J$ if $|J| \leq t$ and $J' \subseteq [n]$ with $|J'| = n - 1$ otherwise,
3. the output distribution satisfies

$$\text{out} \stackrel{id}{=} (\text{AssignWires}(G, \mathcal{W}, (\llbracket x \rrbracket, \llbracket y \rrbracket)), \llbracket z \rrbracket|_{J'}) \quad \text{where} \quad \llbracket z \rrbracket = G(\llbracket x \rrbracket, \llbracket y \rrbracket).$$

The requirement of the RPE property that the failure events $\mathbb{F}_1$ and $\mathbb{F}_2$ are mutually independent might seem too strong. In practice it might be easier to show or verify that some gadgets satisfy a weaker notion. We say that a gadget is (t, f) -weak random probing expandable (wRPE) if the failure events verify $\Pr(\mathbb{F}_1) \leq \varepsilon$, $\Pr(\mathbb{F}_2) \leq \varepsilon$ and $\Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) \leq \varepsilon^2$ instead of (7.2) in Definition 16. Although being easier to achieve and to verify, this notion is actually not much weaker as the original RPE. We have the following reduction of RPE to wRPE; see [BCP⁺20] (Appendix A) for the proof.

Lemma 7. *Let $f = \mathbb{R} \rightarrow [0, 0.14]$. Let $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share gadget. If G is (t, f) -wRPE then G is (t, f') -RPE with $f'(\cdot) = f(\cdot) + \frac{3}{2}f(\cdot)^2$.*

In practice, we can evaluate the wRPE property of a gadget using our IronMask tool [BMRT22] and obtain the following failure event probabilities:

$$\Pr(\mathbb{F}_1) = f_1(p), \quad \Pr(\mathbb{F}_2) = f_2(p) \quad \text{and} \quad \Pr(\mathbb{F}_1 \wedge \mathbb{F}_2) = f_{12}(p),$$

for every $p \in [0, 1]$. Then the above proposition implies that the gadget is (p, f) -RPE with

$$f : p \mapsto f_{\max}(p) + \frac{3}{2}f_{\max}(p)^2 \quad \text{with} \quad f_{\max} = \max(f_1, f_2, \sqrt{f_{12}}). \quad (7.3)$$

It remains to show that the above expandability notion is stronger than the composability notion introduced in Definition 10. Formally, we have the following implication:

Lemma 8. *Let $f = \mathbb{R} \rightarrow \mathbb{R}$ and $n \in \mathbb{N}$. Let G be an n -share gadget. If G is (t, f) -RPE then G is (t, f') -threshold RPC, with $f'(\cdot) = 2 \cdot f(\cdot)$.*

Proof. We consider a (t, f) -RPE n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$. The $(t, 2 \cdot f)$ -random composability property is directly implied by the (t, f) -random probing expandability by making use of the exact same simulators and observing that

$$\Pr((|I_1| > t) \vee (|I_2| > t)) \leq \Pr(|I_1| > t) + \Pr(|I_2| > t) = 2 \cdot \varepsilon.$$

□

The RPE notion and its reduction with threshold RPC leads to the following theorem.

Theorem 2. *Let $n \in \mathbb{N}$ and $f : \mathbb{R} \rightarrow \mathbb{R}$. Let $G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}$ be n -share gadgets for the addition, multiplication and copy on $\mathbb{K}$. Let CC be the standard circuit compiler with sharing order n , using $G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}$ as base gadgets. Let $\text{CC}^{(k)}$ be the expanding circuit compiler obtained by applying k layers of expansion starting from CC . If the base gadgets $G_{\text{add}}, G_{\text{mult}}$ and G_{copy} satisfy the (t, f) -RPE property then the compiler $\text{CC}^{(k)}$ is $(p, 2 \cdot f^{(k)}(p))$ -random probing secure. Namely, for every (randomized) arithmetic circuit C , the compiled circuit $\hat{C} = \text{CC}(C)$ is $(p, |C| \cdot 2 \cdot f^{(k)}(p))$ -random probing secure, where $|C|$ denotes the number of gates in the original circuit.*

The complexity of the expanding compiler can be expressed in terms of the target random probing security advantage $2^{-\kappa}$, and relates to the notion of amplification order recalled below.

Definition 17 (Amplification Order). *Let $f : \mathbb{R} \rightarrow \mathbb{R}$ such that $f(p) = c_d p^d + \mathcal{O}(p^{d+\varepsilon})$ as p tends to 0, for some $c_d > 0$ and $\varepsilon > 0$. We say that f has amplification order d .*

Given $t > 0$ and a gadget G , let d be the largest integer such that G achieves (t, f) -RPE for some f of amplification order d . Then G is said to have amplification order d (with respect to t).

As shown in [BCP⁺20] (Appendix A), the complexity of the expanding compiler relates to the (minimum) amplification order of the three gadgets used in the base compiler CC . If the latter achieves (t, f) -RPE with an amplification order d , the expanding compiler achieves $(p, 2^{-\kappa})$ -random probing security with a complexity blowup of $\mathcal{O}(\kappa^e)$ for an exponent e satisfying

$$e = \frac{\log N_{\max}}{\log d} \quad \text{with} \quad N_{\max} = \max \left(N_{\text{m,m}}, \text{eigenvalues} \left(\begin{pmatrix} N_{\text{a,a}} & N_{\text{c,a}} \\ N_{\text{a,c}} & N_{\text{c,c}} \end{pmatrix} \right) \right)$$

where $N_{\text{x,y}}$ denotes the number of gates “x” in the base gadget “y”, with “m” meaning multiplication, “a” meaning addition, and “c” meaning copy.

7.2. RPE Gadgets and Application to AES

In [BRT21] (Appendix B), we introduced generic RPE gadgets supporting an arbitrary number of shares, achieving near-optimal amplification orders. The expanding compiler proposed in [BRT21, BRTV21] (Appendix C) yields quadratic complexity in general, and even quasi-linear in specific settings depending on the tolerated leakage probability and under some conditions on the base field.

In the same work, we also proposed small, concrete gadgets that enable a more precise estimation of security and implementation costs. Our prior research has shown that using fewer than three shares cannot yield meaningful amplification orders [BCP⁺20], and that maximal amplification orders can only be obtained with an odd number of shares [BRT21]. We therefore focus on the 3-share gadgets for copy, addition, and multiplication, which achieve the maximal amplification order $d = 2$ that we recall below.

$$\begin{aligned} G_{\text{add}} : c_1 &= (r_1 + a_1) + (r_3 + b_1), & G_{\text{copy}} : c_1 &= r_1 + a_1, & d_1 &= r_3 + a_1, \\ c_2 &= (r_2 + a_2) + (r_4 + b_2), & c_2 &= r_2 + a_2, & d_2 &= r_4 + a_2, \\ c_3 &= ((r_1 + r_2) + a_3) + ((r_3 + r_4) + b_3), & c_3 &= (r_1 + r_2) + a_3, & d_3 &= (r_3 + r_4) + a_3. \end{aligned}$$

$$\begin{aligned} G_{\text{mult}} : i_{1,1} &= r_1 + b_1; & i_{1,2} &= r_2 + b_2; & i_{1,3} &= (r_1 + r_2) + b_3; & a'_1 &= r_7 + a_1; \\ i_{2,1} &= r_3 + b_1; & i_{2,2} &= r_4 + b_2; & i_{2,3} &= (r_3 + r_4) + b_3; & a'_2 &= r_8 + a_2; \\ i_{3,1} &= r_5 + b_1; & i_{3,2} &= r_6 + b_2; & i_{3,3} &= (r_5 + r_6) + b_3; & a'_3 &= (r_7 + r_8) + a_3 \\ c_1 &= (a'_1 \cdot i_{1,1} + r_{1,1}) + (a'_1 \cdot i_{1,2} + r_{1,2}) + (a'_1 \cdot i_{1,3} + r_{1,3}) + (r_{1,1} + r_{2,1} + r_{3,1}) \\ c_2 &= (a'_2 \cdot i_{2,1} + r_{2,1}) + (a'_2 \cdot i_{2,2} + r_{2,2}) + (a'_2 \cdot i_{2,3} + r_{2,3}) + (r_{1,2} + r_{2,2} + r_{3,2}) \\ c_3 &= (a'_3 \cdot i_{3,1} + r_{3,1}) + (a'_3 \cdot i_{3,2} + r_{3,2}) + (a'_3 \cdot i_{3,3} + r_{3,3}) + (r_{1,3} + r_{2,3} + r_{3,3}). \end{aligned}$$

To transform an abstract circuit C into a random probing secure implementation, using these 3-share base gadgets, for a given leakage rate p^* and a target security level $2^{-\lambda}$, we proceed as follows¹.

1. Use **IronMask** to evaluate the intermediate wRPE leakage functions of the three base gadgets. This yields exact coefficients c_i for G_{add} and G_{copy} , and upper bounds for G_{mult} (whose exact evaluation is too expensive).
2. Convert these wRPE results to RPE functions via Equation 7.3, with two small refinements:
 - **IronMask** returns two functions for each f_1 , f_2 and f_{12} (because wRPE is decomposed). In each case we take the coefficient-wise maximum of the pair.
 - For the copy gadget G_{copy} , **IronMask** returns four functions corresponding to the four possible cardinality conditions on the two output sets (below/above the threshold t). We form f_{max} by taking the coefficient-wise maximum of these four functions (no square root is applied since there are two outputs but a single input).
3. Take the coefficient-wise maximum of the three resulting RPE polynomials (one per gadget) to obtain a single global polynomial $f(p)$. Determine the maximal leakage rate p_m satisfying $f(p_m) < p_m$: for our 3-share base gadgets, we obtain $p_m = 2^{-7.5}$.
4. Provided $p^* \leq p_m$, compute the minimal number of expansion levels k required for the composed design to meet the target security, *i.e.* choose the smallest integer k such that

$$2 \cdot f^{\circ k}(p^*) \cdot |C| \leq 2^{-\lambda},$$

where $f^{\circ k}$ denotes the k -fold composition of f and $|C|$ is the number of gates in C .

Example. Applying this methodology to the AES-128 encryption setting of [BCP⁺20], we find that for $p = 2^{-7.5}$, achieving 80-bit and 128-bit security requires $k = 10$ and $k = 11$ expansion levels, respectively. For a smaller leakage rate of $p = 2^{-20}$, the required expansion levels drop to $k = 3$ and $k = 4$. The resulting AES implementations correspond to 27 and 81 shares, with estimated complexities of about 2×10^8 computational gates and 5×10^7 random values for 80-bit security, and 3×10^9 gates and 8×10^8 random values for 128-bit security.

Tighter bounds can be achieved with 5-share gadgets, as demonstrated in [BCP⁺20, BRT21] and recalled in Appendices A and B. However, deriving accurate wRPE formulas becomes significantly more involved with current verification tools such as **IronMask**. In addition, combining different compilers, for example, integrating both 3-share and 5-share gadgets, can further improve the balance between security, computational complexity, and tolerated leakage probability, as shown in [BRTV21] (see Appendix C).

¹A Python script is available at <https://github.com/CryptoExperts/poc-expanding-compiler>.

Chapter 8

Cardinal RPC Gadgets from Random Probing Partitioned Refresh

Contents

8.1. Random Probing Partitioned Refresh Gadget	45
8.2. Linear Gadgets	50
8.3. Non-Linear Multiplication	51
8.4. Compiler and Application to AES	54

While Chapter 7 focused on designing threshold RPC gadgets using the expansion strategy, we now turn to the construction of gadgets that satisfy the other composition notions, namely (uniformly) cardinal RPC. We begin with the refresh gadget introduced in [BRR25] (Appendix F), which functionally acts as an identity function while injecting fresh randomness into the input sharing. From this, we derive linear gadgets for addition and linear multiplication. We then present a non-linear multiplication gadget built upon these components, offering competitive performance compared to state-of-the-art designs. Together, these gadgets provide a versatile toolbox for efficient implementations secure in the random probing model. The chapter concludes with a compiler and its application to the AES cipher.

8.1. Random Probing Partitioned Refresh Gadget

We introduce our new refresh gadget, `RPrefresh`, for *random-pair refresh*, described in Algorithm 2. It follows the standard approach of refreshing a sharing by adding a sharing of zero, generated using a zero-encoding algorithm (Definition 18). The novelty lies in the underlying `RPZeroEnc` gadget (Algorithm 1), which samples the zero-sharing. It is parameterized by a number γ of iterations, influencing both complexity and security (up to a certain point).

Definition 18 (ZeroEncoding). *Let $n \in \mathbb{N}$. A zero-encoding algorithm, denoted `ZeroEnc`, is a probabilistic algorithm sampling an n -sharing of 0. In other words, $\llbracket z \rrbracket \leftarrow \text{ZeroEnc}()$ such that $\sum z_i = 0$ (with probability 1).*

The process starts from the all-zero sharing. At each iteration, two share indices are selected uniformly at random, and a fresh random value is added to one share and subtracted from the other. After all iterations, the resulting zero-sharing is added share-wise to the input in `RPrefresh`.

While its conceptual idea is simple, this refreshing technique never outputs a perfectly uniform sharing of zero due to the probabilistic selection of share indices, namely the same pair of shares may be chosen multiple times or not at all. In what follows, we analyze the quality of the output produced by `RPZeroEnc` using the `PartitionZeroEnc` function.

Algorithm 1 RPZeroEnc^γ

Output: $\llbracket z \rrbracket = (z_1, \dots, z_n) \in \mathbb{K}^n$ such that
 $z_1 + \dots + z_n = 0$
1: $\llbracket z \rrbracket = (0, 0, \dots, 0)$ $\triangleright n$ zeros
2: **for** $i = 1$ to γ **do**
3: $(i_1, i_2) \xleftarrow{\$} [1, n]^2$ with $i_1 \neq i_2$
4: $r \xleftarrow{\$} \mathbb{K}$
5: $z_{i_1} \leftarrow z_{i_1} + r$
6: $z_{i_2} \leftarrow z_{i_2} - r$
return $\llbracket z \rrbracket$

Algorithm 2 $\text{RPRefresh}^\gamma(\llbracket a \rrbracket)$

Input: $\llbracket a \rrbracket = (a_1, \dots, a_n) \in \mathbb{K}^n$
Output: $\llbracket s \rrbracket = (s_1, \dots, s_n) \in \mathbb{K}^n$ such that
 $s_1 + \dots + s_n = a_1 + \dots + a_n$
1: $\llbracket z \rrbracket \leftarrow \text{RPZeroEnc}^\gamma()$
2: **for** $i = 1$ to n **do**
3: $s_i \leftarrow a_i + z_i$
return $\llbracket s \rrbracket$

Definition 19 (Partition ZeroEncoding). For $P = (n_1, \dots, n_\ell) \in \mathcal{P}(n)$ an integer partition, we define $\text{PartitionZeroEnc}(P)$ as a function producing a sharing $\llbracket z \rrbracket = \llbracket z_1 \rrbracket \parallel \dots \parallel \llbracket z_\ell \rrbracket$ where $\llbracket z_i \rrbracket$ is a uniformly random n_i -sharing of 0 for every $i \in [1, \ell]$. In other words,

$$\text{PartitionZeroEnc}(n_1, \dots, n_\ell) = \underbrace{z_1, \dots, z_{n_1}}_{\sum z_i = 0}, \underbrace{z_{n_1+1}, \dots, z_{n_1+n_2}}_{\sum z_i = 0}, \dots, \underbrace{z_{n-n_\ell+1}, \dots, z_n}_{\sum z_i = 0}.$$

This definition naturally captures two extremes:

- Worst case: $P = (1, \dots, 1)$ yields an all-zero vector, i.e. $\text{PartitionZeroEnc}(1, \dots, 1) = (0, \dots, 0)$, corresponding to the poorest output quality,
- Ideal case: $P = (n)$ yields a truly uniform n -sharing of 0, achieving the best output quality.

We can now formally relate the output of RPZeroEnc to a composition of PartitionZeroEnc and a permutation, with shared randomness.

Lemma 9. Let $n, \gamma \in \mathbb{N}$. Let ρ be a random tape¹. There exists a partition $P_{\gamma, \rho} \in \mathcal{P}(n)$, a permutation function $\sigma_{\gamma, \rho} : [1, n] \rightarrow [1, n]$ and another random tape $\rho'_{\gamma, \rho}$ such that

$$\text{RPZeroEnc}^\gamma(\rho) = \sigma_{\gamma, \rho}(\text{PartitionZeroEnc}(P_{\gamma, \rho}, \rho'_{\gamma, \rho}))$$

where RPZeroEnc is depicted in Algorithm 1.

Let $\gamma \in \mathbb{N}$ and the random tape ρ be fixed parameters. Here is an intuition of an iterative construction of the partition and the permutation. Before the first iteration, the variable $\llbracket z \rrbracket = (0, \dots, 0)$ is exactly the output of

$$\text{PartitionZeroEnc}(\underbrace{1, 1, \dots, 1}_{n \text{ times}}).$$

Suppose now that the generated indices in the first iteration are $(i_1, i_2) = (1, 2)$, then we have $\llbracket z \rrbracket = (r_1, -r_1, \dots, 0)$ for $r_1 \leftarrow \$$ which is identically distributed to

$$\text{PartitionZeroEnc}(2, \underbrace{1, \dots, 1}_{n-2 \text{ times}}).$$

If the generated indices were different from $(1, 2)$, we would have a distribution identical to

$$\sigma(\text{PartitionZeroEnc}(2, 1, \dots, 1))$$

¹Although we typically omit random tapes in the notation of our functions for simplicity, we make them explicit here to highlight that the same randomness ρ governs both the choice of index pairs and the underlying structure of the generated sharing.

for the same partition but with some coordinate permutation σ .

Now for the second iteration, let $r_2 \leftarrow \$$,

$$\begin{aligned} \text{if } (i_1, i_2) &= (1, 2), \llbracket z \rrbracket = (r_1 + r_2, r_1 - r_2, 0 \dots, 0), \\ \text{if } (i_1, i_2) &= (2, 3), \llbracket z \rrbracket = (r_1, -r_1 + r_2, -r_2, 0 \dots, 0), \\ \text{if } (i_1, i_2) &= (3, 4), \llbracket z \rrbracket = (r_1, -r_1, r_2, -r_2, 0 \dots, 0). \end{aligned}$$

In these cases, $\llbracket z \rrbracket$ will be statistically indistinguishable to

$$\begin{aligned} &\text{PartitionZeroEnc}(2, \underbrace{1, \dots, 1}_{n-2 \text{ times}}) \text{ if } (i_1, i_2) = (1, 2), \\ &\text{PartitionZeroEnc}(3, \underbrace{1, \dots, 1}_{n-3 \text{ times}}) \text{ if } (i_1, i_2) = (2, 3), \\ &\text{PartitionZeroEnc}(2, 2, \underbrace{1, \dots, 1}_{n-4 \text{ times}}) \text{ if } (i_1, i_2) = (3, 4). \end{aligned}$$

All the cases are captured with a coordinate permutation in a similar way.

On and on, one can construct a partition iteratively depending on the randomness of RPZeroEnc . The permutation σ captures the possibility of RPZeroEnc to shuffle the order of the refreshed indices. As the number of iterations accumulates, the distribution of $\llbracket z \rrbracket$ becomes increasingly less partitioned, approaching the ideal partition $P = (n)$. This iterative construction is formalized in a more general case (in the presence of leakage) in Lemma 10.

Let us now address the random probing security of RPZeroEnc . We will show below in Theorem 3 that RPZeroEnc is cardinal RPC (Definition 12). To be able to establish the cardinal RPC property, one needs to take into account the possible leaked iterations in the quality of the refreshing. Naturally, the more leaked randomness, the less qualitative the refreshing is. We introduce the intermediary notion of *random probing partitioned* (RPP) to characterize this quality with respect to the number of iterations and the leakage probability. We later prove the cardinal RPC security of the refresh (Theorem 3) from the RPP security of the zero-encoding gadget.

Definition 20 (Random-Probing-Partition). *Let $n \in \mathbb{N}$ and $\mathcal{E}$ be a distribution envelope over $\mathcal{P}(n)$. An n -share zero-encoding gadget $\text{ZeroEnc} : \emptyset \rightarrow \mathbb{K}^n$ is $(p, \mathcal{E})$ -random-probing-partitioned (RPP) for some $p \in [0, 1]$ if there exists a deterministic algorithm ϕ and a simulator Sim such that the outputs of the random experiment 3 satisfy:*

1. $P \lesssim \mathcal{E}$, and
2. $\llbracket z \rrbracket$ and $\llbracket w \rrbracket + \llbracket u \rrbracket$ are identically distributed.

Experiment 3: Random Probing Partition

$\mathcal{W} \leftarrow \text{LeakingWires}(\text{ZeroEnc}, p)$	<i>Sampling the leaking wires.</i>
$\rho \leftarrow \$$	<i>Sampling a random tape.</i>
$\llbracket z \rrbracket \leftarrow \text{ZeroEnc}(\rho)$	<i>Sampling the 0-encoding $\llbracket z \rrbracket$.</i>
$\text{leak} \leftarrow \text{AssignWires}(\text{ZeroEnc}, \rho, \mathcal{W}, \emptyset)$	<i>Computing the leaked values.</i>
$\llbracket w \rrbracket \leftarrow \text{Sim}(\mathcal{W}, \text{leak})$	<i>Sim outputs an encoding of 0 whose randomness is "weakened" due to the leakage.</i>
$(P, \sigma) \leftarrow \phi(\mathcal{W})$	<i>Computing a partition and a permutation given the leaking wires</i>
$\llbracket u \rrbracket \leftarrow \sigma(\text{PartitionZeroEnc}(P))$	<i>Sampling a 0-encoding corresponding to the "remaining" randomness</i>
return $(P, \llbracket z \rrbracket, \llbracket w \rrbracket, \llbracket u \rrbracket)$	

Intuition of Definition 20. Contrary to most definitions in this manuscript, Definition 20 does not ensure that the random probing leakage can be simulated. The simulator is instead given the leakage

and the leaking wires. The idea of Definition 20 is to show that the output of ZeroEnc is such that

$$\llbracket z \rrbracket \approx \underbrace{\llbracket w \rrbracket}_{\text{leaked 0-encoding}} + \underbrace{\llbracket u \rrbracket}_{\text{fresh partitioned 0-encoding}}.$$

The sharing $\llbracket u \rrbracket$ hence captures the “remaining” randomness of the output sharing $\llbracket z \rrbracket$ given the internal leakage of the gadget. Let us further stress that $\llbracket u \rrbracket$ follows the random distribution $\sigma(\text{PartitionZeroEnc}(P))$ whose randomness is highly related to the partition P . The envelope $\mathcal{E}$ upper bounds the distribution of the partition P defined by $(P, \sigma) = \phi(\text{LeakingWires}(\text{ZeroEnc}, p))$.

The rationale for characterizing RPZeroEnc under the RPP notion is the following. Assume that, after a number of non-leaking iterations, we get a distribution $\llbracket z \rrbracket = \sigma(\text{PartitionZeroEnc}(P))$ for some partition P and coordinate permutation σ . If, during a subsequent iteration, the value of the random r leaks, the output $\llbracket z' \rrbracket$ of this iteration can be expressed as: $\llbracket z' \rrbracket = \llbracket z \rrbracket + \llbracket w \rrbracket$ where $\llbracket w \rrbracket$ is a vector with one coordinate set to r , one to $-r$, and the rest to 0. Given the leakage of r , one can perfectly simulate this $\llbracket w \rrbracket$. This gives the intuition behind the RPP property of RPZeroEnc: non-leaking iterations contribute to reducing the partition structure of $\llbracket u \rrbracket = \sigma(\text{PartitionZeroEnc}(P))$, while leaking iterations result in updates to the $\llbracket w \rrbracket$ component, accumulating the leaked randomness. We note that the leakage can also concern z_{i_1} or z_{i_2} which results in an update of both $\llbracket u \rrbracket$ (or (P, σ)) and $\llbracket w \rrbracket$.

Before formally establishing the RPP security of RPZeroEnc in the next lemma (whose proof is available in [BRR25] and in Appendix F), we introduce the concept of *transition set* of a partition. Assume the output of an iteration corresponds to a partition P . The transition set $\mathcal{T}_j(P)$ contains all partitions P' that can be obtained after an iteration with i_1 and i_2 belonging to the same 0-sharing j of cardinal n_j while the transition set $\mathcal{T}_{j_1, j_2}(P)$ contains all partitions P' that can be obtained when i_1 and i_2 belong to two different 0-sharings j_1 and j_2 of cardinals n_{j_1} and n_{j_2} respectively. These transition sets are displayed in Tables 1 and 2 in [BRR25] (Appendix F), considering the different possible scenarios in terms of leakage. In these tables, expressions such as $(z_{i_1} \oplus z_{i_2}) \wedge \bar{r}$ in the leakage column should be interpreted as a scenario where either z_{i_1} or z_{i_2} leaks (but not both) and r does not leak.

Lemma 10 (RPZeroEnc is RPP). *Let n, γ in $\mathbb{N}$ and $p \in [0, 1]$. The n -share zero-encoding gadget RPZeroEnc^γ is $(p, \mathcal{E}^\gamma)$ -RPP for some distribution envelope $\mathcal{E}^\gamma$ defined over $\mathcal{P}(n)$ constructively as follows. For $\gamma = 0$, we simply have $\mathcal{E}^0(\mathbf{1}) = 1$ and $\mathcal{E}^0(P) = 0$ for all $P \in \mathcal{P}(n) \setminus \{\mathbf{1}\}$, where $\mathbf{1} = (1, 1, \dots, 1)$. Then, for all $\gamma > 0$, we define $\mathcal{E}^{\gamma+1}$ from $\mathcal{E}^\gamma$ as follows:*

1. *for all $P \in \mathcal{P}(n)$, we initiate $\mathcal{E}^{\gamma+1}(P)$ to 0,*
2. *for all $P \in \mathcal{P}(n)$, for all $j \in \{1, \dots, |P|\}$, for all $P' \in \mathcal{T}_j(P)$ from Table 1 in [BRR25] (Appendix F),*

$$\mathcal{E}^{\gamma+1}(P') = \mathcal{E}^{\gamma+1}(P) + \mathbb{P}(P \rightarrow P') \cdot \mathcal{E}^\gamma(P),$$

3. *for all $P \in \mathcal{P}(n)$, for all $j_1, j_2 \in \{1, \dots, |P|\}$ with $j_1 < j_2$, for all $P' \in \mathcal{T}_{j_1, j_2}(P)$ from Table 2 in [BRR25] (Appendix F),*

$$\mathcal{E}^{\gamma+1}(P') = \mathcal{E}^{\gamma+1}(P) + \mathbb{P}(P \rightarrow P') \cdot \mathcal{E}^\gamma(P).$$

In order to simplify the security analysis of RPRefresh, we now introduce an artificial extension of RPZeroEnc, called RPZeroEnc_+ , in which all the output shares are individually manipulated after γ iterations. In terms of leakage, this means that all the output shares of RPZeroEnc can now leak with probability p , in addition to the internal leakage already accounted for in Lemma 10. The proof of Lemma 11 is given in [BRR25] and in Appendix F.

Lemma 11 (RPZeroEnc_+ is RPP). *Let n, γ in $\mathbb{N}$ and $p \in [0, 1]$. The n -share zero-encoding gadget $\text{RPZeroEnc}_+^{\gamma, \gamma}$ is $(p, \mathcal{E}^\gamma)$ -RPP for some distribution envelope $\mathcal{E}^\gamma$ defined over $\mathcal{P}(n)$ constructively as*

follows. The distribution envelope $\mathcal{E}^\gamma$ is initialized to 0 for all the possible partitions. Then, for all k , for all partition $P = (n_1, \dots, n_k)$ in $\mathcal{P}(n)$, for all $(d_1, \dots, d_k)$ such that $d_1 \leq n_1, \dots, d_k \leq n_k$,

$$\mathcal{E}^\gamma(P') = \mathcal{E}^\gamma(P) + \mathcal{E}_{\text{RPZeroEnc}}^\gamma(P) \cdot \prod_{i=1}^k \binom{n_i}{d_i} \cdot p^{d_i} \cdot (1-p)^{n_i-d_i}$$

with $\mathcal{E}_{\text{RPZeroEnc}}^\gamma$ the distribution envelope for the RPP property of RPZeroEnc given in Lemma 10 and $P' = (1, \dots, 1, n_1 - d_1, \dots, n_k - d_k)$.

Building on these results, we arrive at our main theorem: the cardinal RPC security of RPRefresh, whose proof is given in [BRR25] (Appendix F).

Theorem 3 (RPRefresh is Cardinal RPC). *Let $n, \gamma \in \mathbb{N}$, $p \in [0, 1]$. The n -share gadget RPRefresh $^\gamma$ is $(p, (\mathcal{E}_t)_{t \in [0, n]})$ -cardinal RPC with $(\mathcal{E}_t)_{t \in [0, n]}$ defined over $[0, n]$ as follows. For all $t^{\text{in}}, t^{\text{out}} \in [0, n]$,*

$$\mathcal{E}_{t^{\text{out}}}(t^{\text{in}}) = \sum_{i=0}^{t^{\text{in}}} q(i, t^{\text{out}}) \cdot \binom{n-i}{t^{\text{in}}-i} \cdot p^{t^{\text{in}}-i} \cdot (1-p)^{n-t^{\text{in}}}$$

where $q(i, t^{\text{out}})$ is defined constructively as follows:

1. For all $i, t^{\text{out}} \in [0, n]$, initialize $q(i, t^{\text{out}}) \leftarrow 0$.
2. For every $P = (n_1, \dots, n_k) \in \mathcal{P}(n)$, for every $d_1 \in [0, n_1], \dots, d_k \in [0, n_k]$, update

$$q(i, t^{\text{out}}) \leftarrow q(i, t^{\text{out}}) + \mathcal{E}_{\text{RPZeroEnc}_+}^\gamma(P) \cdot \frac{\binom{n_1}{d_1} \cdots \binom{n_k}{d_k}}{\binom{n}{t^{\text{out}}}}$$

for $t^{\text{out}} = d_1 + \dots + d_k$, $i = n_1 \cdot \mathbb{1}_{d_1=n_1} + \dots + n_k \cdot \mathbb{1}_{d_k=n_k}$ and $\mathcal{E}_{\text{RPZeroEnc}_+}^\gamma$ taken from Lemma 11.

Intuitively, $q(i, t^{\text{out}})$ upper bounds the probability that i input shares are required to simulate the leakage on the internal variables of RPRefresh (without considering the input shares) along with the given t^{out} output shares.

Complexity of the envelope computation. The above results provide explicit formulas to compute the envelopes characterizing the cardinal RPC security of our refresh. Following Lemma 10, evaluating those formulas has a complexity $\mathcal{O}(\gamma \cdot |\mathcal{P}(n)| \cdot n^2)$ by iterating over the γ iterations of the refresh, the $|\mathcal{P}(n)|$ partitions of n , and the different pairs of elements within a partition (whose number is lower than n^2). According to the Hardy-Ramanujan asymptotic formula, the number of integer partitions of n can be approximated as

$$|\mathcal{P}(n)| \approx \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right)$$

which is subexponential in n . For instance, $|\mathcal{P}(8)| = 22$, $|\mathcal{P}(16)| = 231$, $|\mathcal{P}(32)| = 8349$, $|\mathcal{P}(64)| < 2^{21}$. In practice, using our (non-optimized) Python implementation available in

<https://github.com/CryptoExperts/EC25-random-probing-Raccoon>,

this computation takes *e.g.*, less than 40 milliseconds for $(n, \gamma) = (8, 50)$, around 9 seconds for $(n, \gamma) = (16, 100)$, around 3 minutes for $(n, \gamma) = (20, 120)$ on a regular laptop.

Figure 8.1 illustrates the threshold RPC security advantage of RPRefresh for various numbers of shares n , with $t = \frac{n}{2}$, and for two leakage probabilities, $p = 2^{-10}$ (Figure 8.1a) and $p = 2^{-16}$ (Figure 8.1b), as a function of the number of random values γ . Those results have been obtained and validated by two independent implementations (from two different developers) of the envelope evaluation. We observe that, after a sufficient number of random values, the security advantage stabilizes, reaching a steady state. This steady state is logically lower bounded by the probability of observing $t + 1$ input shares from a simple random probing leakage of these shares (i.e., each share leaks with probability p), which is given by $\binom{n}{t+1} \cdot p^{t+1} \cdot (1-p)^{n-t-1}$. In the figure, this probability is represented by dotted lines in colors corresponding to n .

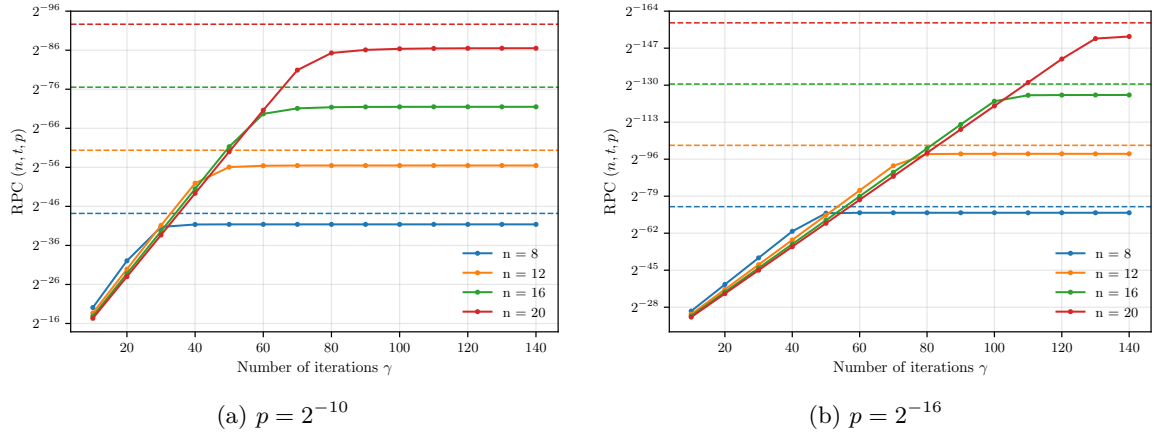


Figure 8.1.: Threshold RPC security advantage ε of RPreFresh for $n \in \{8, 12, 16, 20\}$, $t = \frac{n}{2}$, $p \in \{2^{-10}, 2^{-16}\}$ as a function of the number of iterations γ .

8.2. Linear Gadgets

With RPreFresh now defined, we construct an addition and a scalar multiplication that will serve as building blocks for many common schemes. These constructions generally combine a refresh gadget, called Refresh, with the sharewise operation, as already proposed in [BRT21, BRTV21]. Our main contribution here is to prove the cardinal RPC security of these gadgets from that of the underlying refresh. Note that we have also designed a copy gadget (very similar to the other linear gadget) and a decoding gadget, particularly useful for lattice-based primitives, but we do not provide their descriptions here and instead refer the reader to [BRR25] (or Appendix F) for their complete constructions and proofs.

The linear multiplication gadget was first defined in [BRTV21] with a linear multiplication preceding the refreshing. For consistency with the other basic gadgets, we instead define it here with a refreshing of the shares prior to multiplication: $G_{\text{cmult}}(\llbracket x \rrbracket, c) = \text{Refresh}(\llbracket x \rrbracket) \cdot c$, as illustrated on Figure 8.2.

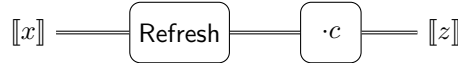


Figure 8.2.: Illustration of G_{cmult} .

Lemma 12 (G_{cmult} is Cardinal RPC). *Let $n \in \mathbb{N}$. Let $\text{Refresh}: \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t \in [0, n]})$ -cardinal RPC for some collection of envelopes $(\mathcal{E}_t^R)_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, G_{cmult} is $(p, (\mathcal{E}_t^{\text{cmult}})_{t \in [0, n]})$ -cardinal RPC where, for all $t^{\text{out}} \in [0, n]$, $\mathcal{E}_{t^{\text{out}}}^{\text{cmult}}$ is defined, for all $t^{\text{in}} \in [0, n]$, as:*

$$\mathcal{E}_{t^{\text{out}}}^{\text{cmult}}(t^{\text{in}}) = \sum_{i=0}^{n-t^{\text{out}}} \binom{n-t^{\text{out}}}{i} \cdot p^i \cdot (1-p)^{n-t^{\text{out}}-i} \cdot \mathcal{E}_{t^{\text{out}}+i}^R(t^{\text{in}}).$$

Proof. The only additional wires that can potentially leak in G_{cmult} compared to Refresh are the output shares of Refresh. The probability that t^{in} input shares are required to simulate the leakage and the final output shares is therefore obtained by computing the product of the leakage probability of i such wires with the probability of Refresh, considering an output composed of the global output plus these extra leaking wires. The probability that i output shares of Refresh leak, out of the $n - t^{\text{out}}$

output shares that remain unknown to the attacker, is given by

$$\binom{n - t^{\text{out}}}{i} \cdot p^i \cdot (1 - p)^{n - t^{\text{out}} - i},$$

which completes the proof. $\square$

G_{add} is the simple addition gadget we introduced in [BRT21]. It is defined as $G_{\text{add}}(\llbracket x \rrbracket, \llbracket y \rrbracket) = \text{Refresh}(\llbracket x \rrbracket) + \text{Refresh}(\llbracket y \rrbracket)$ with a classical sharewise addition, as illustrated in Figure 8.3.

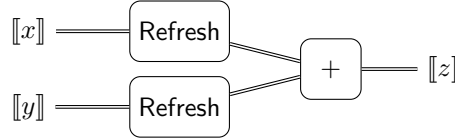


Figure 8.3.: Illustration of G_{add} .

Lemma 13 (G_{add} is Cardinal RPC). *Let $n \in \mathbb{N}$. Let $\text{Refresh}: \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t \in [0, n]})$ -cardinal RPC for some collection envelopes $(\mathcal{E}_t^R)_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, G_{add} is $(p, (\mathcal{E}_t^{\text{add}})_{t \in [0, n]})$ -cardinal RPC where, for all $t^{\text{out}} \in [0, n]$, $\mathcal{E}_{t^{\text{out}}}^{\text{add}}$ is defined, for all $\mathbf{t}_{\text{in}} = (t_1^{\text{in}}, t_2^{\text{in}}) \in [0, n]^2$, as*

$$\begin{aligned} \mathcal{E}_{t^{\text{out}}}^{\text{add}}(\mathbf{t}_{\text{in}}) &= \sum_{i_1=0}^{n-t^{\text{out}}} \binom{n-t^{\text{out}}}{i_1} \cdot p^{i_1} \cdot (1-p)^{n-t^{\text{out}}-i_1} \cdot \mathcal{E}_{t^{\text{out}}+i_1}^R(t_1^{\text{in}}) \\ &\quad \cdot \sum_{i_2=0}^{n-t^{\text{out}}} \binom{n-t^{\text{out}}}{i_2} \cdot p^{i_2} \cdot (1-p)^{n-t^{\text{out}}-i_2} \cdot \mathcal{E}_{t^{\text{out}}+i_2}^R(t_2^{\text{in}}). \end{aligned}$$

Proof. We assume (this is the worst case) that all the t^{out} output shares given to the attacker require the corresponding $2 \cdot t^{\text{out}}$ output shares of the two instances of Refresh to be simulated. As for G_{cmult} , the output shares to simulate for each instance of Refresh are completed by the possible leakage at the input of the addition. In that case, the probabilities for both instances of Refresh to require $t_{\text{in}, \alpha}$ input shares (for $\alpha \in \{1, 2\}$) are independent given the total number of output shares to simulate with the leakage and we obtain the above envelope. $\square$

In general, the addition gadget G_{add} can involve two distinct refresh gadgets, each with a different envelopes collection. Lemma 13 can be easily updated to account for this by getting use of the first envelopes collection in the first sum and the second envelopes collection in the second sum.

8.3. Non-Linear Multiplication

To complete our set of basic gadgets, we aim to construct a non-linear multiplication that satisfies the cardinal RPC security notion. Our design builds on the multiplication scheme originally proposed by [BCPZ16], which follows a classical structure. It first performs an expansion step, called **Expand**, that takes as input two n -sharings and produce an n^2 -sharing of all pairwise products of the input shares. It is followed by a compression step, called **Comp**, to reduce this to an n -sharing of the final result. Although this scheme is heuristically resistant to horizontal attacks and is thus expected to provide strong security in the random probing model, it lacks a formal proof in that context.

We refine the original construction in two key aspects. First, we enhance the expansion step by balancing the refreshing of the shares (as suggested in [CS19]) and incorporating uniformly distributed

random permutations wherever they strengthen security. Our proposal, referred to as **UnifMatMult**, thus constitutes a new instantiation of **Expand**, detailed in Algorithm 3.

Algorithm 3 UnifMatMult

Input: n_x -sharing $\llbracket x \rrbracket$ and n_y -sharing $\llbracket y \rrbracket$

Output: $(n_x \cdot n_y)$ -sharing $\mathbf{M} = (M_{i,j})_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}}$ such that $\sum_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}} M_{i,j} = x \cdot y$

if $n_x = 1$ and $n_y = 1$ **then** $\mathbf{M} \leftarrow (x_1 \cdot y_1)$

else if $n_x = 1$ and $n_y = 2$ **then** $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 & x_1 \cdot y_2 \end{pmatrix}$

else if $n_x = 2$ and $n_y = 1$ **then** $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 \\ x_2 \cdot y_1 \end{pmatrix}$

else if $n_x = 2$ and $n_y = 2$ **then** $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 & x_1 \cdot y_2 \\ x_2 \cdot y_1 & x_2 \cdot y_2 \end{pmatrix}$

else

$\mathbf{x}^{(1)} \leftarrow (x_1, \dots, x_{\lfloor \frac{n_x}{2} \rfloor})$, $\mathbf{x}^{(2)} \leftarrow (x_{\lceil \frac{n_x}{2} \rceil}, \dots, x_{n_x})$

$\mathbf{y}^{(1)} \leftarrow (y_1, \dots, y_{\lfloor \frac{n_y}{2} \rfloor})$, $\mathbf{y}^{(2)} \leftarrow (y_{\lceil \frac{n_y}{2} \rceil}, \dots, y_{n_y})$

$\sigma_1, \sigma_2, \sigma_3, \sigma_4 \xleftarrow{\$} \mathcal{S}_*$

$\mathbf{x}_\sigma^{(1)} \leftarrow \sigma_1(\mathbf{x}^{(1)})$, $\mathbf{x}_\sigma^{(2)} \leftarrow \sigma_2(\mathbf{x}^{(2)})$, $\mathbf{y}_\sigma^{(1)} \leftarrow \sigma_3(\mathbf{y}^{(1)})$, $\mathbf{y}_\sigma^{(2)} \leftarrow \sigma_4(\mathbf{y}^{(2)})$

$\mathbf{M}^{(1,1)} \leftarrow \text{UnifMatMult}(\text{RPRefresh}(\mathbf{x}_\sigma^{(1)}), \text{RPRefresh}(\mathbf{y}_\sigma^{(1)}))$

$\mathbf{M}^{(1,2)} \leftarrow \text{UnifMatMult}(\text{RPRefresh}(\mathbf{x}^{(1)}), \text{RPRefresh}(\mathbf{y}_\sigma^{(2)}))$

$\mathbf{M}^{(2,1)} \leftarrow \text{UnifMatMult}(\text{RPRefresh}(\mathbf{x}_\sigma^{(2)}), \text{RPRefresh}(\mathbf{y}^{(1)}))$

$\mathbf{M}^{(2,2)} \leftarrow \text{UnifMatMult}(\text{RPRefresh}(\mathbf{x}^{(2)}), \text{RPRefresh}(\mathbf{y}^{(2)}))$

$\mathbf{M} = \begin{pmatrix} \mathbf{M}^{(1,1)} & \mathbf{M}^{(1,2)} \\ \mathbf{M}^{(2,1)} & \mathbf{M}^{(2,2)} \end{pmatrix}$

return $\mathbf{M}$

Second, we address a limitation of the original compression procedure, which lacks a recursive structure and makes it challenging to compute the cardinal RPC envelopes efficiently. To resolve this, we introduce a new compression gadget, denoted **TreeAdd** (Algorithm 4), which replaces the original **Comp** while maintaining functional equivalence and facilitating security analysis. This new design leverages a tree-based addition structure that improves both efficiency and the ability to decompose the computation recursively.

Algorithm 4 TreeAdd

Input : $(M_{i,j})_{\substack{1 \leq i \leq k \\ 1 \leq j \leq n}}$ k n -sharings.

Output : An n -sharing of $\sum_{i=1}^k \sum_{j=1}^n M_{i,j}$.

if $k = 1$ **then return** $(M_{1,j})_{1 \leq j \leq n}$

if $k = 2$ **then return** $\text{Add}^\gamma((M_{1,j})_{1 \leq j \leq n}, (M_{2,j})_{1 \leq j \leq n})$

$\llbracket z_1 \rrbracket \leftarrow \text{BasicTreeAdd}((M_{i,j})_{\substack{1 \leq i \leq \lfloor \frac{k}{2} \rfloor \\ 1 \leq j \leq n}})$, $\llbracket z_2 \rrbracket \leftarrow \text{BasicTreeAdd}((M_{i,j})_{\substack{\lceil \frac{k}{2} \rceil \leq i \leq k \\ 1 \leq j \leq n}})$

return $\text{Add}^\gamma(\llbracket z_1 \rrbracket, \llbracket z_2 \rrbracket)$

We developed a script that computes the cardinal RPC envelope of our new multiplication algorithm by combining the envelopes of the basic gadgets together with permutations (see Lemma 5):

<https://github.com/CryptoExperts/AC25-cardinal-rp-compiler>.

Comparison with the state of the art. Hereafter, we compare the security and complexity of our

construction with the two state-of-the-art multiplication gadgets for which random probing security has been established²:

1. The multiplication gadget from [BFO23], referred to as BFO23, builds on the ISW multiplication gadget [ISW03] and follows the same two-step structure: an expansion and a compression. Given two n -sharing $\llbracket x \rrbracket, \llbracket y \rrbracket$, **Expand** computes for any pair $(i, j) \in [n]^2, i \neq j$ the cross-products $x_i \cdot y_j$ and $x_j \cdot y_i$, adds them together with a random value $r_{i,j}$ in one share and stores the random value in another share of the n^2 -sharing. If $i = j$, then it stores $x_i \cdot y_i$ directly. The **Compress** step then reduces this n^2 -sharing to an n -sharing through a tree of additions.
2. The multiplication gadget from [JMB24], referred to as JMB24, also adopts the two-step structure. In the **Expand** phase, it refreshes each sub-sharing before every recursive call, but does not use permutations. Its **Comp** phase follows the original design of [BCPZ16].

Since both works provide analytical bounds on random probing security, we can directly compare their results with ours. A bound on the random probing security of our gadgets can be derived from their cardinal RPC envelope: if our multiplication gadget is $(p, \mathcal{E})$ cardinal RPC then it is (p, ε) random probing secure with

$$\varepsilon = \sum_{\substack{t_1^{\text{in}}, t_2^{\text{in}} \in [0, n] \\ t_1^{\text{in}} = n \text{ or } t_2^{\text{in}} = n}} \mathcal{E}_0(t_1^{\text{in}}, t_2^{\text{in}}).$$

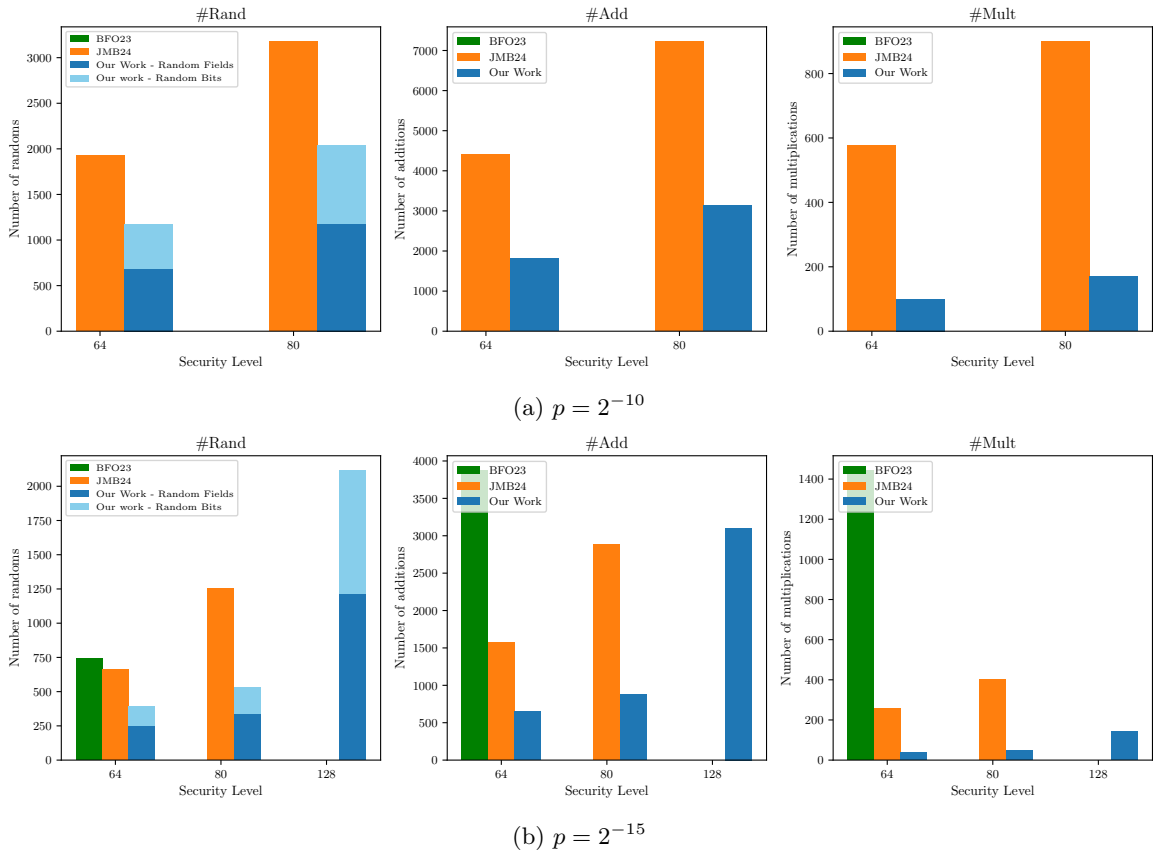


Figure 8.4.: Complexity comparison of three multiplication gadgets with random probing security of $\varepsilon = 2^{-64}, 2^{-80}, 2^{-128}$.

²We omit earlier approaches, such as random probing expansion, whose trade-offs are significantly less favorable.

Relying on this equality, we present in Figure 8.4 a comparison of the complexity of the different multiplication designs for leakage rates $p \in \{2^{-10}, 2^{-15}\}$ and target random probing security levels $\varepsilon \in \{2^{-64}, 2^{-80}, 2^{-128}\}$. For our gadget, the randomness complexity also accounts for the random bits used to select share indices during refresh, as well as those required for the random permutations (highlighted in light blue). As discussed earlier, this type of randomness is likely to be less costly in practice than the randomness needed for random gates.

It is worth noting that BF023 [BFO23] does not appear in several scenarios, as it fails to achieve the selected security levels for some leakage rates, regardless of the number of shares $n \in \mathbb{N}$. Likewise, the analytical formula provided in JMB24 [JMB24] is only valid up to $n = 30$ shares, which is insufficient to reach certain target security levels when the leakage rate is low. As a result, some scenarios include both comparisons, others only JMB24, and a few rely solely on our results.

Overall, Figure 8.4 shows that our proposal generally achieves significantly lower complexity than existing state-of-the-art approaches, except in a few cases with low leakage rates and weak target security levels, where the randomness requirements are comparable to those of previous works.

8.4. Compiler and Application to AES

Our proposed circuit compiler features the following gadgets:

- *Addition and multiplication by a constant:* The linear gadgets $\text{Add}^\gamma, \text{cMult}^\gamma$ from [BRR25] (Appendix F) and corresponding to the instantiation of G_{add} and G_{cmult} of Section 8.2 with RPRefresh ;
- *Addition of a constant:* A linear gadget cAdd^γ for adding a constant $c \in \mathbb{K}$, defined as the composition of RPRefresh with the addition of c to the first share (see [BRR25] or Appendix F for its security proof);
- *Squaring (characteristic 2 only):* For $\mathbb{K} = \mathbb{F}_{2^k}$ with $k \in \mathbb{N}^*$ (as in the AES case), we define a squaring gadget Sq^γ by applying RPRefresh followed by a sharewise squaring. Its cardinal RPC security can be analyzed in the same way as for cMult^γ ;
- *Multiplication:* The UnifMatMult gadget presented in Section 8.3.

Moreover, our compiler inserts random permutations and refresh gadgets in targeted regions of the circuit to deal with copies. Specifically, we consider two situations:

1. *Fan-out 2 setting:* When an n -sharing serves as input to exactly two subsequent gadgets, we simply apply a permutation before the input of one of the two gadgets. This approach preserves the *UI-cardinal RPC* property, as established in Lemma 5.
2. *Fan-out greater than 2 setting:* A dedicated procedure is required to safely reuse the same sharing across multiple gadgets. Without appropriate safeguards, repeated use of the same sharing increases the probability of leakage from each individual share, thereby significantly weakening the overall random probing security. To address this pitfall, whenever a sharing $\llbracket x \rrbracket^{(0)}$ is used more than twice (we cannot technically avoid the use of two copies of a single sharing), we use a refresh gadget to securely duplicate the data. Specifically, the sharing $\llbracket x \rrbracket^{(0)}$ is used directly in the first gadget, after which it is refreshed to produce a new independent sharing $\llbracket x \rrbracket^{(1)}$. Like in the fan-out 2 setting, a permutation is applied before the first gadget or before the refreshing. If no further duplications are required, the refreshed copy can also be used directly in the second and the third gadgets, with a permutation inserted in one of the branch (as in the fan-out 2 setting). Otherwise, the procedure continues recursively: one copy is used directly, and the other is refreshed again to produce $\llbracket x \rrbracket^{(2)}$, with a permutation applied to one of the input, and so on. Figure 8.5 illustrates the procedures for a fan-out 3.

Composition Strategy for the Compiled Circuits. We now describe the composition strategy to evaluate the random probing security of compiled circuits. Ideally, this would involve computing

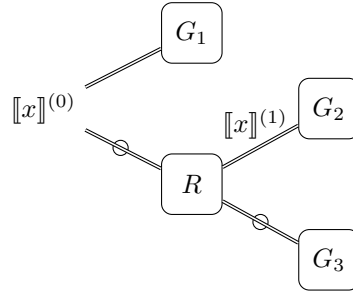


Figure 8.5.: Illustration of the duplication of a sharing $\llbracket x \rrbracket^{(0)}$ across three gadgets G_1 , G_2 , and G_3 . The black circles on the edges indicate the permutations applied to the shares.

the cardinal RPC envelope of the entire compiled circuit. However, this approach is often impractical, particularly in the case of circuits with numerous input and output sharings, each composed of a substantial number of shares. The complexity of such an analysis grows at least as $(n+1)^{\ell+m}$, where ℓ and m denote the number of n -shared inputs and outputs, respectively.

This is why we adopt a *block-wise approach* for computing the random probing security of larger circuits. We start by partitioning our larger circuit into blocks of gadgets for which the global cardinal RPC envelopes can be computed within a reasonable amount of time. Subsequently, we use [BRR25, Lemma 1] to infer the threshold RPC security of each block from its cardinal RPC security, which in turn enables us to estimate the overall random probing security of the compiled circuit.

To ensure efficient computation of the block-wise cardinal RPC security, we impose constraints on the size of each block. Specifically, given ℓ input sharings and m output sharings with n shares, we require that $n^{\ell+m} < 2^\delta$, where δ is a chosen threshold value (we use $\delta = 30$ in the application to AES presented below). In this work, we do not focus on optimizing the block partitioning strategy to maximize the global random probing security. Instead, we rely on manual clustering of the compiled circuits, as exemplified hereafter in the case of AES.

Application to AES. We apply our compiler to an arithmetic circuit over $\mathbb{F}_{256}$ implementing the AES block cipher, with the only modification that two additional refresh gadgets were inserted in the S-boxes to strengthen the overall security. The full circuit and its decomposition into gadget blocks are presented in [BNR25] (Appendix H).

Figure 8.6 illustrates the security-complexity trade-offs of the resulting masked AES in the cardinal RPC framework and compares them with prior random probing secure AES compilers. Specifically, for leakage rates $p \in \{2^{-16}, 2^{-20}\}$, our construction achieves random probing security levels $\varepsilon \in \{2^{-64}, 2^{-80}, 2^{-128}\}$. We report the randomness usage (with light blue for the random bits dedicated to the permutations) and the counts of field additions and multiplications required in each of these settings, and compare with the following compilers:

- **Our Expansion Strategy** [BCP⁺20, BRT21, BRTV21] (Chapter 7): This approach relies on a compiler equipped with dedicated gadgets for addition, copy, and multiplication. When applied to the AES encryption circuit, it yields implementations that remain significantly more expensive than those obtained with the more recent cardinal RPC method. For instance, with the 3-share base gadgets presented in Chapter 7, achieving 80-bit security at a leakage rate of 2^{-20} results in about 2×10^8 computational gates and 5×10^7 random values, while reaching 128-bit security under the same leakage rate requires around 3×10^9 gates and 8×10^8 random values (see Section 7.2, Chapter 7). Due to this substantial difference in complexity, Figure 8.6 does not report the results for the expansion-based strategy.
- **BFO23** [BFO23]: This compiler also uses addition, copy, refresh, and multiplication gadgets. Notably, it inserts a refresh gadget between each operation to ensure composability. The authors

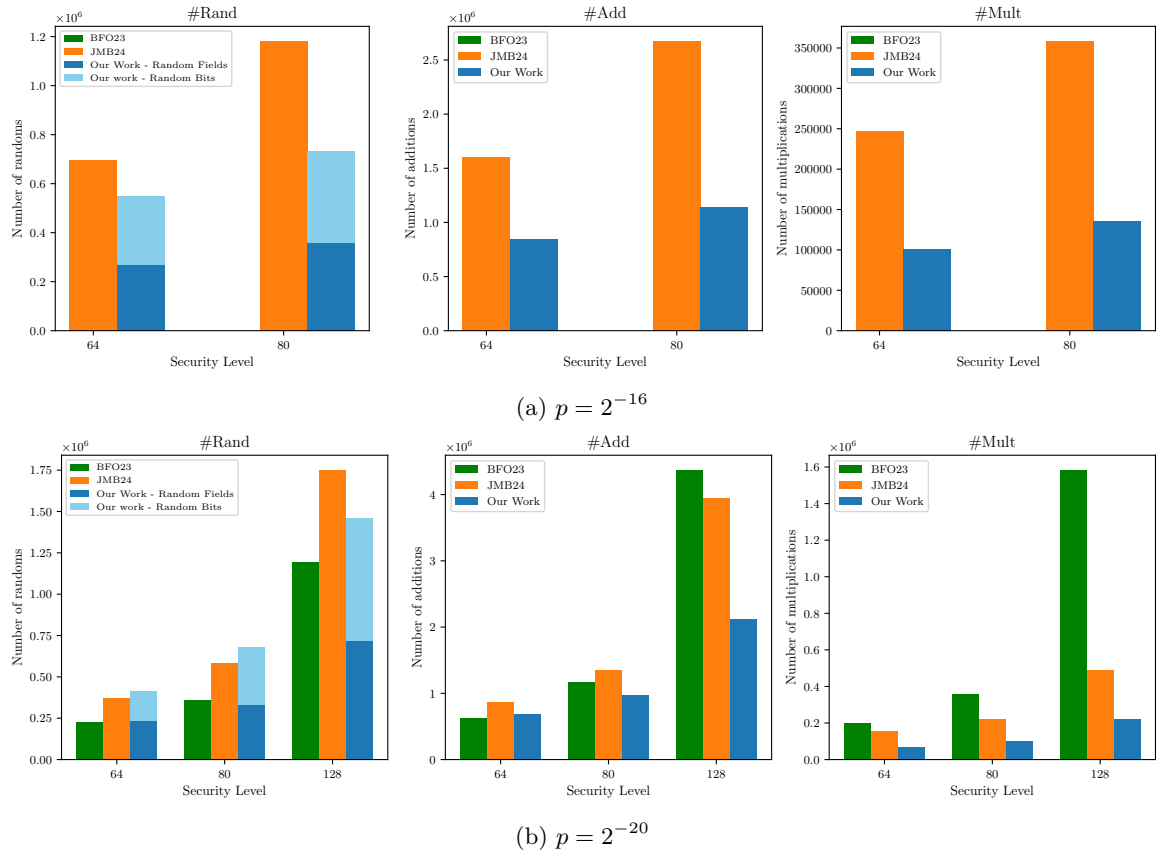


Figure 8.6.: Number of randoms, additions and multiplications for three security levels

also provide an analytical formula to estimate the random probing security of their circuits. We applied this compiler to our masked AES implementation for comparison. For $p = 2^{-15}$, the achieved security level never exceeds 44 bits (*i.e.* the failure probability remains above 2^{-44}) for any number of shares $n \in \mathbb{N}^*$, making meaningful comparisons with our results infeasible. For $p = 2^{-20}$, our compiler achieves lower complexities (or similar for random complexities) for all three target security levels.

- **JMB24** [JMB24]: This work presents a masked AES implementation whose random probing security level is expressed as a function of n and p . As shown in Figure 8.6, our compiler consistently outperforms JMB24 in both computational and randomness usage for all considered leakage probabilities and target security levels. It also offers a strictly stronger guarantee: our construction achieves security level ε in the standard simulation-based random probing model, whereas JMB24 only bounds the mutual information between the leakage and a single key byte. Such a bound only limits average leakage and does not yield simulation-based security, whereas simulation-based security (when the distance between the simulated leakage and the actual leakage is statistical) both implies the corresponding mutual information bound and composes well within larger protocols.

Beyond these results, we also compare our block-wise cardinal RPC setting with the combination of the same gadgets directly under the threshold RPC framework. At $p = 2^{-20}$, our block-wise cardinal RPC compiler already exceeds the 64-bit target (*i.e.* with $\varepsilon = 2^{-67}$) with $n = 8$ shares. In the same setting (p, n) , combining directly the threshold RPC advantages of our gadgets (copies included) would only yield $\varepsilon = 2^{-61}$. This gap highlights the benefit of (block-wise) cardinal RPC.

Part V.

Conclusion

Conclusion

Cryptographic implementations are now expected to be provably secure against side-channel attacks [KJJ99] when deployed on embedded devices. Most of them integrates masking [CJRR99, GP99] for that purpose, where the idea is to split each sensitive variable into shares and to operate directly on them. However, linking these protected implementations to a concrete security level remains challenging. For roughly a decade, masked implementations were proposed without formal security proofs, until the community began to rely on leakage models. The probing model of Ishai, Sahai, and Wagner [ISW03] then became the basis for most analyses and tools, although it does not capture all practical attacks. By contrast, the noisy leakage model, introduced by Chari *et al.* [CJRR99] and later formalized by Prouff and Rivain [PR13], is more realistic but also harder to use directly in proofs. A key development by Duc, Dziembowski, and Faust in 2014 [DDF14] showed that security in an intermediate *random probing model* implies security in the noisy leakage setting. Motivated by this bridge between theory and practice, this thesis contributes concrete compositional tools and constructions in the random probing model, enabling the design of efficient masked implementations with a well-defined practical security level on embedded devices.

As a first step toward this goal, we formalized random probing security in [BCP⁺20] (Chapter 3) and introduced a simple verification method. More recently, we clarified the connection to practical security in [BCM⁺25] (Chapter 4), by describing a methodology that maps an abstract circuit to a practically secure implementation on a target device, relying on the reduction of Duc *et al.* This methodology, which naturally calls for the design of a random probing compiler, provides the conceptual foundation for the other technical contributions presented in this manuscript.

Turning to random probing compilers, some schemes already offered random probing security in 2020, when we started this project, but only a few for an explicit constant leakage rate [AIS18]. Building on these first steps, we introduced a new composability notion, *threshold RPC* [BCP⁺20] (Chapter 5). As with earlier compilers, the idea was simple: if small gadgets satisfy the property individually, then the associated composition rules allow them to be assembled into a full random probing secure circuit. Subsequently, the broader *general RPC* framework of Cassiers *et al.* [CFOS21] yielded tighter bounds but struggled to scale to bigger circuits, which motivated our *cardinal* [BRR25] and *uniformly cardinal* frameworks [BNR25] (Chapter 6). We compared these approaches with alternative (non-simulation-based) methods [BFO23, JMB24], highlighting their respective trade-offs in tightness and complexity.

Alongside these notions, we designed base gadgets that satisfy them: using the expansion technique (building on the approach of Ananth, Ishai, and Sahai [AIS18]) for threshold RPC (Chapter 7), a new refresh gadget for cardinal RPC, and additional permutations for uniformly cardinal RPC (Chapter 8). These building blocks enabled progressively more efficient compilers which, combined with the method above, lead to implementations that can be practically secure on embedded devices.

Naturally, limitations remain. In our methodology to build practically secure implementations, naive whitening is costly, noise levels on current devices are too low for our techniques, and several proofs are not yet tight. On the compiler side, while our successive contributions significantly improve the trade-off between tightness of security bounds, implementation complexity, leakage rate, and verification cost, the resulting implementations are not yet fully competitive for industrial deployment.

Importantly, these limitations also delineate clear and concrete directions for future work. A deeper investigation of gadget designs satisfying the (uniformly) cardinal RPC composition rules appears

particularly promising, as does the derivation of tighter analytical bounds for their security. In parallel, designing cryptographic schemes that are inherently masking-friendly, with random probing security as a design goal, offers a realistic path toward lighter and more competitive implementations. Finally, advancing automated verification techniques remains essential: recent progress shows that combining scalability with tight bounds under realistic leakage models is an achievable objective.

Overall, the framework developed in this thesis establishes a coherent methodology that bridges formal security models and practical implementations. By providing new compositional tools, concrete gadgets, and a clear path from abstract security to embedded-device guarantees, it lays a solid foundation for future advances. Building on these results, I am confident that the next steps will lead to tighter analyses, more efficient compilers, and ultimately to practically secure masked implementations that meet real-world constraints.

Bibliography

- [ADF16] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 586–615. Springer, Berlin, Heidelberg, May 2016.
- [AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 427–455. Springer, Cham, August 2018.
- [Ajt11] Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 715–724. ACM Press, June 2011.
- [BBC⁺19] Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. maskVerif: Automated verification of higher-order masking in presence of physical defaults. In Kazue Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part I*, volume 11735 of *LNCS*, pages 300–318. Springer, Cham, September 2019.
- [BBD⁺15] Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 457–485. Springer, Berlin, Heidelberg, April 2015.
- [BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. *IACR TCHES*, 2022(4):553–588, 2022.
- [BCG⁺23] Julien Béguinot, Wei Cheng, Sylvain Guilley, Yi Liu, Loïc Masure, Olivier Rioul, and François-Xavier Standaert. Removing the field size loss from duc et al.’s conjectured bound for masked encodings. In Elif Bilge Kavun and Michael Pehl, editors, *COSADE 2023*, volume 13979 of *LNCS*, pages 86–104. Springer, Cham, April 2023.
- [BCM⁺25] Sonia Belaïd, Gaëtan Cassiers, Camille Mutschler, Matthieu Rivain, Thomas Roche, François-Xavier Standaert, and Abdul Rahman Taleb. SoK: A methodology to achieve provable side-channel security in real-world implementations. *CiC*, 2(1):4, 2025.
- [BCP⁺20] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 339–368. Springer, Cham, August 2020.
- [BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Berlin, Heidelberg, August 2016.
- [BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ASCAD with a single side-channel trace. Cryptology ePrint Archive, Report 2021/817, 2021.

- [BDF⁺17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 535–566. Springer, Cham, April / May 2017.
- [BFG⁺24] Sonia Belaïd, Jakob Feldtkeller, Tim Güneysu, Anna Guinet, Jan Richter-Brockmann, Matthieu Rivain, Pascal Sasdrich, and Abdul Rahman Taleb. Formal definition and verification for combined random fault and random probing security. In Kai-Min Chung and Yu Sasaki, editors, *ASIACRYPT 2024, Part VII*, volume 15490 of *LNCS*, pages 167–200. Springer, Singapore, December 2024.
- [BFG⁺25] Christof Beierle, Jakob Feldtkeller, Anna Guinet, Tim Güneysu, Gregor Leander, Jan Richter-Brockmann, and Pascal Sasdrich. INDIANA - verifying (random) probing security through indistinguishability analysis. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology - EUROCRYPT 2025 - 44th Annual International Conference on the Theory and Applications of Cryptographic Techniques Madrid, Spain, May 4-8, 2025, Proceedings, Part VIII*, volume 15608 of *Lecture Notes in Computer Science*, pages 33–63. Springer, 2025.
- [BFO23] Francesco Berti, Sebastian Faust, and Maximilian Ortl. Provable secure parallel gadgets. *IACR TCHES*, 2023(4):420–459, 2023.
- [BGG⁺14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.
- [BGG⁺22] Roderick Bloem, Barbara Gigerl, Marc Gourjon, Vedad Hadzic, Stefan Mangard, and Robert Primas. Power contracts: Provably complete power leakage models for processors. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 381–395. ACM Press, November 2022.
- [BMRT22] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. IronMask: Versatile verification of masking security. In *2022 IEEE Symposium on Security and Privacy*, pages 142–160. IEEE Computer Society Press, May 2022.
- [BNR25] Sonia Belaïd, Victor Normand, and Matthieu Rivain. Masked circuit compiler in the cardinal random probing composability framework. *Cryptology ePrint Archive*, Paper 2025/1747, 2025.
- [BRR25] Sonia Belaïd, Matthieu Rivain, and Mélissa Rossi. New techniques for random probing security and application to raccoon signature scheme. In Serge Fehr and Pierre-Alain Fouque, editors, *EUROCRYPT 2025, Part VIII*, volume 15608 of *LNCS*, pages 94–123. Springer, Cham, May 2025.
- [BRT21] Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 313–343. Springer, Cham, October 2021.
- [BRTV21] Sonia Belaïd, Matthieu Rivain, Abdul Rahman Taleb, and Damien Vergnaud. Dynamic random probing expansion with quasi linear asymptotic complexity. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 157–188. Springer, Cham, December 2021.

- [BS21] Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms. *IACR TCHES*, 2021(3):202–234, 2021.
- [BUS22] Davide Bellizia, Balazs Udvarhelyi, and François-Xavier Standaert. Towards a better understanding of side-channel analysis measurements setups. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications*, pages 64–79, Cham, 2022. Springer International Publishing.
- [CDSU23] Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. Efficient regression-based linear discriminant analysis for side-channel security evaluations towards analytical attacks against 32-bit implementations. *IACR TCHES*, 2023(3):270–293, 2023.
- [CFOS21] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards tight random probing security. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 185–214, Virtual Event, August 2021. Springer, Cham.
- [Che52] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on a sum of observations. *Ann. Math. Statist.*, 23(4):493–507, 1952.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 398–412. Springer, Berlin, Heidelberg, August 1999.
- [CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In *CARDIS*, volume 8419 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2013.
- [CM24] Gaëtan Cassiers and Charles Momin. The SMAesH dataset. Cryptology ePrint Archive, Paper 2024/1521, 2024.
- [CPW24] Hao Cheng, Daniel Page, and Weijia Wang. eLIMInate: a leakage-focused ISE for masked implementation. *IACR TCHES*, 2024(2):329–358, 2024.
- [CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, Jr., Çetin Kaya Koç, and Christof Paar, editors, *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, Berlin, Heidelberg, August 2003.
- [CS19] Gaëtan Cassiers and François-Xavier Standaert. Towards globally optimized masking: From low randomness to low noise rate. *IACR TCHES*, 2019(2):162–198, 2019.
- [CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR TCHES*, 2021(2):136–158, 2021.
- [DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Berlin, Heidelberg, May 2014.
- [DFS15] Stefan Dziembowski, Sebastian Faust, and Maciej Skorski. Noisy leakage revisited. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 159–188. Springer, Berlin, Heidelberg, April 2015.
- [dPKPR24] Rafaël del Pino, Shuichi Katsumata, Thomas Prest, and Mélissa Rossi. Raccoon: A masking-friendly signature proven in the probing model. In Leonid Reyzin and Douglas Stebila, editors, *CRYPTO 2024, Part I*, volume 14920 of *LNCS*, pages 409–444. Springer, Cham, August 2024.

- [FGM⁺18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR TCHES*, 2018(3):89–120, 2018.
- [GJR18] Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain. How to securely compute with noisy leakage in quasilinear complexity. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 547–574. Springer, Cham, December 2018.
- [GMPP20] Si Gao, Ben Marshall, Dan Page, and Thinh Pham. FENL: an ISE to mitigate analogue micro-architectural leakage. *IACR TCHES*, 2020(2):73–98, 2020.
- [GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the “duplication” method). In Çetin Kaya Koç and Christof Paar, editors, *CHES’99*, volume 1717 of *LNCS*, pages 158–172. Springer, Berlin, Heidelberg, August 1999.
- [ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003.
- [JMB24] Vahid Jahandideh, Bart Mennink, and Lejla Batina. An algebraic approach for evaluating random probing security with application to AES. *IACR TCHES*, 2024(4):657–689, 2024.
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 388–397. Springer, Berlin, Heidelberg, August 1999.
- [MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR TCHES*, 2020(1):348–375, 2019.
- [MKSM22] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional leakage in theory and practice unveiling security flaws in masked circuits. *IACR TCHES*, 2022(2):266–288, 2022.
- [MOW17] David McCann, Elisabeth Oswald, and Carolyn Whitnall. Towards practical tools for side channel aware software engineering: ‘grey box’ modelling for instruction leakages. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017*, pages 199–216. USENIX Association, August 2017.
- [MPW22] Ben Marshall, Dan Page, and James Webb. MIRACLE: MICRo-ARChitectural leakage evaluation A study of micro-architectural power leakage across many devices. *IACR TCHES*, 2022(1):175–220, 2022.
- [MR04] Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004*, volume 2951 of *LNCS*, pages 278–296. Springer, Berlin, Heidelberg, February 2004.
- [PGMP19] Thomas Prest, Dahmun Goudarzi, Ange Martinelli, and Alain Passelègue. Unifying leakage models on a Rényi day. In Alexandra Boldyreva and Daniele Micciancio, editors, *CRYPTO 2019, Part I*, volume 11692 of *LNCS*, pages 683–712. Springer, Cham, August 2019.
- [PPM⁺23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, 55(11):227:1–227:35, 2023.

- [PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Berlin, Heidelberg, May 2013.
- [SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES 2008*, volume 5154 of *LNCS*, pages 411–425. Springer, Berlin, Heidelberg, August 2008.
- [SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *CHES 2005*, volume 3659 of *LNCS*, pages 30–46. Springer, Berlin, Heidelberg, August / September 2005.
- [WO19] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of ISO 17825 (‘testing methods for the mitigation of non-invasive attack classes against cryptographic modules’). In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part III*, volume 11923 of *LNCS*, pages 256–284. Springer, Cham, December 2019.

Part VI.

Appended publications

Appendix A

Random Probing Security: Verification, Composition, Expansion and New Constructions

Hereafter is appended the full version of our paper [\[BCP⁺20\]](#), joint work with Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb, published at **CRYPTO 2020**.

Random Probing Security: Verification, Composition, Expansion and New Constructions

Sonia Belaïd¹, Jean-Sébastien Coron², Emmanuel Prouff^{3,4}, Matthieu Rivain¹, and
Abdul Rahman Taleb¹

¹ CryptoExperts, France

² University of Luxembourg

³ ANSSI, France

⁴ Sorbonne Universités, UPMC Univ Paris 06, POLSYS, UMR 7606, LIP6, F-75005, Paris, France

{sonia.belaid,matthieu.rivain,abdul.taleb}@cryptoexperts.com

⁵ jean-sebastien.coron@uni.lu

⁶ emmanuel.prouff@ssi.gouv.fr

Abstract. The masking countermeasure is among the most powerful countermeasures to counteract side-channel attacks. Leakage models have been exhibited to theoretically reason on the security of such masked implementations. So far, the most widely used leakage model is the *probing model* defined by Ishai, Sahai, and Wagner at (CRYPTO 2003). While it is advantageously convenient for security proofs, it does not capture an adversary exploiting full leakage traces as, *e.g.*, in horizontal attacks. Those attacks target the multiple manipulations of the same share to reduce noise and recover the corresponding value. To capture a wider class of attacks another model was introduced and is referred to as the *random probing model*. From a leakage parameter p , each wire of the circuit leaks its value with probability p . While this model much better reflects the physical reality of side channels, it requires more complex security proofs and does not yet come with practical constructions.

In this paper, we define the first framework dedicated to the random probing model. We provide an automatic tool, called VRAPS, to quantify the random probing security of a circuit from its leakage probability. We also formalize a composition property for secure random probing gadgets and exhibit its relation to the *strong non-interference* (SNI) notion used in the context of probing security. We then revisit the expansion idea proposed by Ananth, Ishai, and Sahai (CRYPTO 2018) and introduce a compiler that builds a random probing secure circuit from small base gadgets achieving a *random probing expandability* property. We instantiate this compiler with small gadgets for which we verify the expected properties directly from our automatic tool. Our construction can tolerate a leakage probability up to 2^{-8} , against 2^{-25} for the previous construction, with a better asymptotic complexity.

Keywords: Compiler, Masking, Automated verification, Random probing model

1 Introduction

Most cryptographic algorithms are assumed to be secure against *black-box* attacks where the adversary is limited to the knowledge of some inputs and outputs to recover the manipulated secrets. However, as revealed in the late nineties [21], when implemented on physical devices, they become vulnerable to the more powerful *side-channel attacks* which additionally exploit the physical emanations such as temperature, time, power consumption, electromagnetic radiations.

As such attacks may only require cheap equipment and can be easily mounted in a short time interval, the community had to adapt quickly by looking for efficient countermeasures. The most widely deployed approach to counteract side-channel attacks was simultaneously introduced in 1999 by Chari et al. [12] and by Goubin and Patarin [18] and is now called *masking*. Basically, the idea is to split each sensitive variable x of the implementation into n shares such that $n - 1$ of them

are generated uniformly at random and the last one is computed as the combination of x and all the previous shares according to some group law $*$. When $*$ is the (bitwise) addition, we talk about *linear sharing* (aka Boolean masking). The adversary thus needs to get information on all the shares of x to recover information on the sensitive value. This countermeasure is really simple to implement for linear operations which are simply applied on each share separately. However, things are getting trickier for non-linear operations where it is impossible to compute the result without combining shares.

To reason about the security of masked implementations, the community introduced leakage models. One of the most broadly used is the *probing model*, introduced by Ishai, Sahai, and Wagner [20]. In a nutshell, a circuit is claimed to be t -probing secure if the exact values of any set of t intermediate variables do not reveal any information on the secrets. As leakage traces are assumed to reveal noisy functions of the manipulated data, this model is motivated by the difficulty to recover information from the combination of t variables from their noisy functions in masking schemes (as t grows). Nevertheless, the probing model fails to capture the huge amount of information resulting from the leakage of all manipulated data, and in particular from the repeated manipulation of identical values (see horizontal attacks in [7]). Therefore, after a long sequence of works building and analyzing masking schemes with respect to their security in the probing model [25, 15, 9], the community is now looking for security in more practical models.

The *noisy leakage model* was originally considered by Chari et al. in [12] and was later formalized by Prouff and Rivain in [24] as a specialization of the *only computation leaks* model [23] in order to better capture the reality of the physical leakage. Informally, a circuit is secure in the noisy leakage model if the adversary cannot recover the secrets from a noisy function of each intermediate variable of the implementation. While realistic, this model is not convenient for security proofs, and therefore masking schemes continued to be verified in the probing model relying on the *not tight* reduction that was formally established by Duc, Dziembowski, and Faust [17].

The latter reduction actually came with an intermediate leakage model, called *random probing model*, to which the security in the noisy leakage model reduces to. In the random probing model, each intermediate variable leaks with some constant leakage probability p . A circuit is secure in this model if there is a negligible probability that these leaking wires actually reveal information on the secrets. It is worth noting that this notion advantageously captures the horizontal attacks which exploit the repeated manipulations of variables throughout the implementation. Classical probing-secure schemes are also secure in the random probing model but the tolerated leakage probability (a.k.a. leakage rate) might not be constant which is not satisfactory from a practical viewpoint. Indeed, in practice the side-channel noise might not be customizable by the implementer.

Only a few constructions [1, 3, 2] tolerate a constant leakage probability. These three constructions are conceptually involved and their practical instantiation is not straightforward. The first one from Ajtai et al. and its extension [3] are based on expander graphs. The tolerated probability is not made explicit. The third work [2] is based on multi-party computation protocols and an expansion strategy; the tolerated probability is around 2^{-26} and for a circuit with $|C|$ gates, the complexity is $\mathcal{O}(|C| \cdot \text{poly}(\kappa))$ for some parameter κ but the polynomial is not made explicit.

Following the long sequence of works relying on the probing security, formal tools have recently been built to supervise the development of masking implementations proven secure in the probing model. Namely, verification tools are now able to produce a security proof or identify potential attacks from the description of a masked implementation at up to some masking orders (i.e., < 5) [4, 14, 11]. In the same vein, compilers have been built to automatically generate masked

implementations at any order given the high level description of a primitive [5, 11, 10]. Nevertheless, no equivalent framework has yet been proposed to verify the security of implementations in the random probing model.

Our contributions. In this paper, we aim to fill this huge gap by providing a framework to verify, compose, and build random probing secure circuits from simple gadgets. Our contributions are three-fold.

Automatic verification tool. As a first contribution, we define a verification method that we instantiate in a tool to automatically exhibit the random probing security parameters of any small circuit defined with addition and multiplication gates whose wires leak with some probability p . In a nutshell, a circuit is (p, f) -random probing secure if it leaks information on the secret with probability $f(p)$, where $f(p)$ is the *failure probability function*. From these notations, our tool named VRAPS (for Verifier of Random Probing Security), based on top of a set of rules that were previously defined to verify the probing security of implementations [4], takes as input the description of a circuit and outputs an upper bound on the failure probability function. While it is limited to small circuits by complexity, the state-of-the-art shows that verifying those circuits can be particularly useful in practice (see e.g. the `maskVerif` tool [4]), for instance to verify gadgets and then deduce global security through composition properties and/or low-order masked implementations. The source code of VRAPS is publicly available.⁷

Composition and expanding compiler. We introduce a composition security property to make gadgets composable in a global random probing secure circuit. We exhibit the relation between this new *random probing composability* (RPC) notion and the *strong non-interference* (SNI) notion which is widely used in the context of probing security [5]. Then, we revisit the modular approach of Ananth, Ishai, and Sahai [2] which uses an expansion strategy to get random probing security from a multi-party computation protocol. We introduce the *expanding compiler* that builds random probing secure circuits from small base gadgets. We formalize the notion of *random probing expandability* (RPE) and show that a base gadget satisfying this notion can be securely used in the expanding compiler to achieve arbitrary/composable random probing security. As a complementary contribution, our verification tool, VRAPS, is extended to verify the newly introduced RPC and RPE properties.

Instantiation. We instantiate the expanding compiler with new constructions of simple base gadgets that fulfill the desired RPE property, which is verified by VRAPS. For a security level κ , our instantiation achieves a complexity of $\mathcal{O}(\kappa^{7.5})$ and tolerates a constant leakage probability $p \approx 0.0045 > 2^{-8}$. In comparison, and as a side contribution, we provide a precise analysis of the construction from [2] and show that it achieves an $\mathcal{O}(\kappa^{8.2})$ complexity for a much lower tolerated leakage probability ($p \approx 2^{-26}$). Finally, we note that our framework probably enables more efficient constructions based on different base gadgets; we leave such optimizations open for future works.

2 Preliminaries

Along the paper, $\mathbb{K}$ shall denote a finite field. For any $n \in \mathbb{N}$, we shall denote $[n]$ the integer set $[n] = [1, n] \cap \mathbb{Z}$. For any tuple $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{K}^n$ and any set $I \subseteq [n]$, we shall denote

⁷ See <https://github.com/CryptoExperts/VRAPS>

$\mathbf{x}|_I = (x_i)_{i \in I}$. Any two probability distributions D_1 and D_2 are said ε -close, denoted $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε , that is

$$\text{SD}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon ,$$

where $p_{D_1}(\cdot)$ and $p_{D_2}(\cdot)$ denote the probability mass functions of D_1 and D_2 .

2.1 Circuit Compilers

In this paper, an *arithmetic circuit* over a field $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations over $\mathbb{K}$. We consider three types of arithmetic gate:

- an addition gate, of fan-in 2 and fan-out 1, computes an addition over $\mathbb{K}$,
- a multiplication gate, of fan-in 2 and fan-out 1, computes a multiplication over $\mathbb{K}$,
- a copy gate, of fan-in 1 and fan-out 2, outputs two copies of its input.

A *randomized arithmetic circuit* is equipped with an additional type of gate:

- a random gate, of fan-in 0 and fan-out 1, outputs a fresh uniform random value of $\mathbb{K}$.

A (randomized) arithmetic circuit is further formally composed of input gates of fan-in 0 and fan-out 1 and output gates of fan-in 1 and fan-out 0. Evaluating an ℓ -input m -output circuit C consists in writing an input $\mathbf{x} \in \mathbb{K}^\ell$ in the input gates, processing the gates from input gates to output gates, then reading the output $\mathbf{y} \in \mathbb{K}^m$ from the output gates. This is denoted by $\mathbf{y} = C(\mathbf{x})$. During the evaluation process, each wire in the circuit is assigned with a value on $\mathbb{K}$. We call the tuple of all these wire values a *wire assignment* of C (on input $\mathbf{x}$).

Definition 1 (Circuit Compiler). A circuit compiler is a triplet of algorithms $(\text{CC}, \text{Enc}, \text{Dec})$ defined as follows:

- CC (circuit compilation) is a deterministic algorithm that takes as input an arithmetic circuit C and outputs a randomized arithmetic circuit $\hat{C}$.
- Enc (input encoding) is a probabilistic algorithm that maps an input $\mathbf{x} \in \mathbb{K}^\ell$ to an encoded input $\hat{\mathbf{x}} \in \mathbb{K}^{\ell'}$.
- Dec (output decoding) is a deterministic algorithm that maps an encoded output $\hat{\mathbf{y}} \in \mathbb{K}^{m'}$ to a plain output $\mathbf{y} \in \mathbb{K}^m$.

These three algorithms satisfy the following properties:

- **Correctness:** For every arithmetic circuit C of input length ℓ , and for every $\mathbf{x} \in \mathbb{K}^\ell$, we have

$$\Pr(\text{Dec}(\hat{C}(\hat{\mathbf{x}})) = C(\mathbf{x}) \mid \hat{\mathbf{x}} \leftarrow \text{Enc}(\mathbf{x})) = 1 , \text{ where } \hat{C} = \text{CC}(C).$$

- **Efficiency:** For some security parameter $\lambda \in \mathbb{N}$, the running time of $\text{CC}(C)$ is $\text{poly}(\lambda, |C|)$, the running time of $\text{Enc}(\mathbf{x})$ is $\text{poly}(\lambda, |\mathbf{x}|)$ and the running time of $\text{Dec}(\hat{\mathbf{y}})$ is $\text{poly}(\lambda, |\hat{\mathbf{y}}|)$, where $\text{poly}(\lambda, q) = O(\lambda^{k_1} q^{k_2})$ for some constants k_1, k_2 .

2.2 Linear Sharing and Gadgets

In the following, the n -linear decoding mapping, denoted LinDec , refers to the function $\bigcup_n \mathbb{K}^n \rightarrow \mathbb{K}$ defined as

$$\text{LinDec} : (x_1, \dots, x_n) \mapsto x_1 + \dots + x_n ,$$

for every $n \in \mathbb{N}$ and $(x_1, \dots, x_n) \in \mathbb{K}^n$. We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $(\hat{x}_1, \dots, \hat{x}_\ell) \in (\mathbb{K}^n)^\ell$ the n -linear decoding mapping acts as

$$\text{LinDec} : (\hat{x}_1, \dots, \hat{x}_\ell) \mapsto (\text{LinDec}(\hat{x}_1), \dots, \text{LinDec}(\hat{x}_\ell)) .$$

Let us recall that for some tuple $\hat{x} = (x_1, \dots, x_n) \in \mathbb{K}^n$ and for some set $I \subseteq [n]$, the tuple $(x_i)_{i \in I}$ is denoted $\hat{x}|_I$.

Definition 2 (Linear Sharing). Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an n -linear sharing of x is a random vector $\hat{x} \in \mathbb{K}^n$ such that $\text{LinDec}(\hat{x}) = x$. It is said to be uniform if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $\hat{x}|_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. A n -linear encoding is a probabilistic algorithm LinEnc which on input a tuple $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{K}^\ell$ outputs a tuple $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_\ell) \in (\mathbb{K}^n)^\ell$ such that $\hat{x}_i$ is a uniform n -sharing of x_i for every $i \in [\ell]$.

In the following, we shall call an $(n\text{-share}, \ell\text{-to-}m)$ gadget, a randomized arithmetic circuit that maps an input $\hat{\mathbf{x}} \in (\mathbb{K}^n)^\ell$ to an output $\hat{\mathbf{y}} \in (\mathbb{K}^n)^m$ such that $\mathbf{x} = \text{LinDec}(\hat{\mathbf{x}}) \in \mathbb{K}^\ell$ and $\mathbf{y} = \text{LinDec}(\hat{\mathbf{y}}) \in \mathbb{K}^m$ satisfy $\mathbf{y} = g(\mathbf{x})$ for some function g . In this paper, we shall consider gadgets for three types of functions (corresponding to the three types of gates): the addition $g : (x_1, x_2) \mapsto x_1 + x_2$, the multiplication $g : (x_1, x_2) \mapsto x_1 \cdot x_2$ and the copy $g : x \mapsto (x, x)$. We shall generally denote such gadgets G_{add} , G_{mult} and G_{copy} respectively.

Definition 3 (Standard Circuit Compiler). Let $\lambda \in \mathbb{N}$ be some security parameter and let $n = \text{poly}(\lambda)$. Let G_{add} , G_{mult} and G_{copy} be n -share gadgets respectively for the addition, multiplication and copy over $\mathbb{K}$. The standard circuit compiler with sharing order n and base gadgets G_{add} , G_{mult} , G_{copy} is the circuit compiler $(\text{CC}, \text{Enc}, \text{Dec})$ satisfying the following:

1. The input encoding Enc is an n -linear encoding.
2. The output decoding Dec is the n -linear decoding mapping LinDec .
3. The circuit compilation CC consists in replacing each gate in the original circuit by an n -share gadget with corresponding functionality (either G_{add} , G_{mult} or G_{copy}), and each wire by a set of n wires carrying a n -linear sharing of the original wire. If the input circuit is a randomized arithmetic circuit, each of its random gates is replaced by n random gates, which duly produce a n -linear sharing of a random value.

For such a circuit compiler, the correctness and efficiency directly holds from the correctness and efficiency of the gadgets G_{add} , G_{mult} and G_{copy} .

2.3 Random Probing Leakage

Let $p \in [0, 1]$ be some constant leakage probability parameter. This parameter is sometimes called *leakage rate* in the literature. Informally, the p -random probing model states that during the evaluation of a circuit C each wire leaks its value with probability p (and leaks nothing otherwise), where all the wire leakage events are mutually independent.

In order to formally define the random-probing leakage of a circuit, we shall consider two probabilistic algorithms:

- The *leaking-wires sampler* takes as input a randomized arithmetic circuit C and a probability $p \in [0, 1]$, and outputs a set $\mathcal{W}$, denoted as

$$\mathcal{W} \leftarrow \text{LeakingWires}(C, p) ,$$

where $\mathcal{W}$ is constructed by including each wire label from the circuit C with probability p to $\mathcal{W}$ (where all the probabilities are mutually independent).

- The *assign-wires sampler* takes as input a randomized arithmetic circuit C , a set of wire labels $\mathcal{W}$ (subset of the wire labels of C), and an input $\mathbf{x}$, and it outputs a $|\mathcal{W}|$ -tuple $\mathbf{w} \in (\mathbb{K} \cup \{\perp\})^{|\mathcal{W}|}$, denoted as

$$\mathbf{w} \leftarrow \text{AssignWires}(C, \mathcal{W}, \mathbf{x}) ,$$

where $\mathbf{w}$ corresponds to the assignments of the wires of C with label in $\mathcal{W}$ for an evaluation on input $\mathbf{x}$.

We can now formally define the random probing leakage of a circuit.

Definition 4 (Random Probing Leakage). *The p -random probing leakage of a randomized arithmetic circuit C on input $\mathbf{x}$ is the distribution $\mathcal{L}_p(C, \mathbf{x})$ obtained by composing the leaking-wires and assign-wires samplers as*

$$\mathcal{L}_p(C, \mathbf{x}) \stackrel{id}{=} \text{AssignWires}(C, \text{LeakingWires}(C, p), \mathbf{x}) .$$

Remark 1. By convention the output wires of C (i.e. the wires incoming output gates) are excluded by the `LeakingWires` sampler whereas the input wires of C (i.e. the wires connecting input gates to subsequent gates) are included. Namely the output set $\mathcal{W}$ of `LeakingWires`(C, p) does not include any output wire label of C . This is because when composing several circuits (or gadgets), the output wires of a circuit are the input wires in a next circuit. This also relates to the widely admitted *only computation leaks* assumption [23]: the processing of a gate leaks information on its input values (and information on the output can be captured through information on the input).

Definition 5 (Random Probing Security). *A randomized arithmetic circuit C with $\ell \cdot n \in \mathbb{N}$ input gates is (p, ε) -random probing secure with respect to encoding Enc if there exists a simulator Sim such that for every $\mathbf{x} \in \mathbb{K}^\ell$:*

$$\text{Sim}(C) \approx_\varepsilon \mathcal{L}_p(C, \text{Enc}(\mathbf{x})) . \tag{1}$$

A circuit compiler $(\text{CC}, \text{Enc}, \text{Dec})$ is (p, ε) -random probing secure if for every (randomized) arithmetic circuit C the compiled circuit $\hat{C} = \text{CC}(C)$ is $(p, |\hat{C}| \cdot \varepsilon)$ -random probing secure where $|\hat{C}|$ is the size of original circuit.

As in [2] we shall consider a *simulation with abort*. In this approach, the simulator first calls the leaking-wires sampler to get a set $\mathcal{W}$ and then either aborts (or fails) with probability ε or outputs the exact distribution of the wire assignment corresponding to $\mathcal{W}$. Formally, for any leakage probability $p \in [0, 1]$, the simulator Sim is defined as

$$\text{Sim}(\hat{C}) = \text{SimAW}(\hat{C}, \text{LeakingWires}(\hat{C}, p)) \tag{2}$$

where SimAW , the *wire assignment simulator*, either returns $\perp$ (simulation failure) or a perfect simulation of the requested wires. Formally, the experiment

$$\begin{aligned}\mathcal{W} &\leftarrow \text{LeakingWires}(\widehat{C}, p) \\ \text{out} &\leftarrow \text{SimAW}(\widehat{C}, \mathcal{W})\end{aligned}$$

leads to

$$\Pr[\text{out} = \perp] = \varepsilon \quad \text{and} \quad (\text{out} \mid \text{out} \neq \perp) \stackrel{\text{id}}{=} (\text{AssignWires}(\widehat{C}, \mathcal{W}, \text{Enc}(\mathbf{x})) \mid \text{out} \neq \perp) . \quad (3)$$

It is not hard to see that if we can construct such a simulator SimAW for a compiled circuit $\widehat{C}$, then this circuit is (p, ε) -random probing secure.

3 Formal Verification

In this section we show how to compute the simulation failure probability $f(p)$ as a function of the leakage probability p for the base gadgets. Since even for simple gadgets this task would be difficult to perform by hand, we use a formal verification tool to compute $f(p)$.

3.1 Simulation Failure probability

We first derive an upper bound on the simulation failure probability as a function of the leakage probability p . We consider a compiled circuit $\widehat{C}$ composed of s wires labeled from 1 to s and a simulator SimAW as defined in previous section. For any sub-set $\mathcal{W} \subseteq [s]$ we denote by $\delta_{\mathcal{W}}$ the value defined as follows:

$$\delta_{\mathcal{W}} = \begin{cases} 1 & \text{if } \text{SimAW}(\widehat{C}, \mathcal{W}) = \perp, \\ 0 & \text{otherwise.} \end{cases}$$

The simulation failure probability ε in (3) can then be explicitly expressed as a function of p . Namely, we have $\varepsilon = f(p)$ with f defined for every $p \in [0, 1]$ by:

$$f(p) = \sum_{\mathcal{W} \subseteq [s]} \delta_{\mathcal{W}} \cdot p^{|\mathcal{W}|} \cdot (1-p)^{s-|\mathcal{W}|} . \quad (4)$$

Letting c_i be the number of sub-sets $\mathcal{W} \subseteq [s]$ of cardinality i for which $\delta_{\mathcal{W}} = 1$, namely for which the simulation fails, we have $c_i = \sum_{|\mathcal{W}|=i} \delta_{\mathcal{W}}$ and hence (4) simplifies to

$$f(p) = \sum_{i=1}^s c_i \cdot p^i \cdot (1-p)^{s-i} . \quad (5)$$

For any circuit $\widehat{C}$ achieving t -probing security, the values $\delta_{\mathcal{W}}$ with $|\mathcal{W}| \leq t$ are equal to zero, and therefore the corresponding c_i 's are zero, which implies the following simplification:

$$f(p) = \sum_{i=t+1}^s c_i \cdot p^i \cdot (1-p)^{s-i} .$$

Moreover, by definition, the coefficients c_i satisfy:

$$c_i \leq \binom{s}{i} \quad (6)$$

which leads to the following upper-bound for $f(p)$:

$$f(p) \leq \sum_{i=t+1}^s \binom{s}{i} \cdot p^i \cdot (1-p)^{s-i} .$$

Example: evaluating $f(p)$ for the 2-share ISW multiplication gadget (ISW-2). This gadget takes at input the 2-sharings (x_0, x_1) and (y_0, y_1) of x and y respectively, and outputs the 2-sharing

$$(z_0, z_1) = (x_0 \cdot y_0 + r_0, x_1 \cdot y_1 + r_0 + x_0 \cdot y_1 + x_1 \cdot y_0)$$

where r_0 is a random value. The processing is composed of the following intermediate results, where each variable is assigned a wire:

$$\begin{array}{llll} c_0 = x_0 * y_0 & z_0 = c_0 + r_0 & c_1 = x_1 * y_1 & c_2 = c_1 + r_0 \\ c_3 = x_0 * y_1 & c_4 = c_2 + c_3 & c_5 = x_1 * y_0 & z_1 = c_4 + c_5 \end{array}$$

When the same variable is involved as input of several operations, a copy gadget (with 1 input wire and 2 output wires) is applied to duplicate it. Consequently, each new use of the same variable as input of an operation adds 2 wires to the final count of overall wires. It may be checked that the circuit corresponding to ISW-2 is composed of 21 wires, excluding the 2 output wires. Since it is 1-SNI but not 2-SNI, every set with a single wire can be simulated, which is not the case for all pairs of wires. Actually, 51 among the latter pairs cannot be simulated. If we continue the test for the sets of cardinality from 3 to 21, we get the following list of coefficients c_i , $1 \leq i \leq 21$, computed with the verification tool described in the next section: 0, 51, 754, 4827, 18875, 52994, 115520, 203176, 293844, 352702, 352715, 293930, 203490, 116280, 54264, 20349, 5985, 1330, 210, 21, 1. Directly injecting these coefficients in (5) gives the expression of $f(p)$ for ISW-2.

3.2 Verification method

For any compiled circuit $\widehat{C}$ and any simulator defined as in Section 2.3, the computation of the function $f(p)$ for any probability p essentially amounts to computing the values of the coefficients c_i 's appearing in (5). If no assumption is made on the circuit, this task seems difficult to carry out by hand. Actually, it may be checked that an exhaustive testing of all the possible tuples of wires for a gadget with s wires has complexity lower bounded by 2^s , which gives 2^{21} for a simple gadget like the ISW multiplication gadget with two shares per input. Here, we introduce a verification tool, that we call VRAPS, enabling to automatically test the perfect simulation for any set of wires of size lower than or equal to some threshold β . The role of the latter threshold is simply to control the verification duration (which can be long if the circuit to test is complex). Our tool implicitly defines a simulator that may fail with a probability $\varepsilon = f(p)$ satisfying (5).

The verification tool takes as input the representation of a compiled circuit $\widehat{C}$ and a test parameter β , and outputs the list of coefficients $c_1, \dots, c_\beta$. It is assumed that $\widehat{C}$ takes as input the n -linear encoding $\text{Enc}(\mathbf{x})$ of vector $\mathbf{x} = (x_1, \dots, x_\ell)$ defined in $\mathbb{K}^\ell$. It is moreover assumed that $\widehat{C}$ is composed of s wires respectively denoted by $w_1, \dots, w_s$. In the following, we consider s -tuples in the form of $u = (u_1, \dots, u_s) \in \{0, 1\}^s$ together with the common rule $u' \subset u$ iff for every $i \in [s]$, $u'_i = 1 \Rightarrow u_i = 1$ (in this case u' will be said to be included in u). An s -tuple u for which there exists an assignment of the wires in $\mathcal{W} = \{w_i; i \in [s], u_i = 1\}$ such that the simulation fails shall be called a *failure tuple*. Such a tuple shall be said to be *incompressible* if no tuple $t' \subset t$ is a failure tuple. The main idea of the proposed verification tool is to test the simulation failure only on incompressible failure tuples whose Hamming weight ranges from 1 to β . The steps are described in Algorithm 1.

Algorithm 1 Verification tool

Input: a compiled circuit $\widehat{C}$ with s wires and a threshold $\beta \leq s$
Output: a list of β coefficients $c_1, \dots, c_\beta$

```

1:  $\ell_p \leftarrow []$  ▷ will be used to store a list of failure tuples
2:  $\mathbf{c} \leftarrow (0, \dots, 0)$  ▷ will be used to store the output coefficients
3: for  $h = 1$  to  $\beta$  do
4:    $\ell_h \leftarrow \text{listTuples}(s, h)$  ▷ list of  $s$ -tuples of Hamming weight  $h$ 
5:    $(\ell_h^p, \ell_h^{f1}) \leftarrow \text{eliminateFromSmaller}(\ell_h, \ell_p)$  ▷ select tuples including an incompressible failure tuple
6:    $\ell_h^{f2} \leftarrow \text{failureTest}(\widehat{C}, \ell_h^p)$  ▷ identify failure tuples in  $\ell_h^p$ 
7:    $\ell_p \leftarrow \ell_p \cup \ell_h^{f2}$  ▷ update list of incompressible failure tuples
8:    $\mathbf{c} \leftarrow \text{updateCoeffs}(\mathbf{c}, \ell_h^{f1} \cup \ell_h^{f2})$  ▷ update coefficients
9: end for
10: return  $\mathbf{c}$ 

```

The function `listTuples` outputs the list of all s -tuples with Hamming weight h with $h \in [s]$. The function `eliminateFromSmaller` takes as input the list ℓ_h of current tuples of Hamming weight h and the list of incompressible failure tuples ℓ_p . It returns two lists:

- ℓ_h^{f1} : the elements of ℓ_h which are not incompressible (*i.e.* which include at least one element from ℓ_p)
- ℓ_h^p : the elements of ℓ_h which are incompressible (*i.e.* $\ell_h \setminus \ell_h^{f1}$)

The function `failureTest` takes as input the second list ℓ_h^p and checks if a perfect simulation can be achieved for each wire family $\mathcal{W}$ corresponding to a tuple in ℓ_h^p . Basically, for each wire family, a sequence of rules taken from `maskVerif` [4] is tested to determine whether $\mathcal{W}$ can be perfectly simulated. It outputs ℓ_h^{f2} , the list of incompressible failure s -tuples of Hamming weight h . In a nutshell, each wire w_i in $\mathcal{W}$ is considered together with the algebraic expression $\varphi_i(\cdot)$ describing its assignment by $\widehat{C}$ as a function of the circuit inputs and the random values returned by the random gates, then the three following rules are successively and repeatedly applied on all the wires families $\mathcal{W}$ (see [4] for further details):

- rule 1:** check whether all the expressions $\varphi_i(\cdot)$ corresponding to wires w_i in $\mathcal{W}$ contain all the shares of at least one of the coordinates of $\mathbf{x}$;
- rule 2:** for every $\varphi_i(\cdot)$, check whether a random r (*i.e.* an output of a random gate) additively masks a sub-expression e (which does not involve r) and appears nowhere else in the other $\varphi_j(\cdot)$

with $j \neq i$; in this case replace the sum of the so-called sub-expression and r by r , namely $e + r \leftarrow r$;

rule 3: apply mathematical simplifications on the tuple.

Function `updateCoeffs` takes as input the current array of β coefficients c_i for $1 \leq i \leq \beta$ and the concatenation of both lists of potential failure tuples $\ell_h^{f_1}$ and $\ell_h^{f_2}$. For each failure tuple, these coefficients are updated.

Link with the tool `maskVerif`. This tool was introduced in [4] to automatically and formally verify higher-order masking implementations, and has further been improved to verify the t -NI and t -SNI security properties. Essentially, this tool verifies each property by analyzing the dependency of sets of fixed number of wires (say t) with a specific number of input shares. In our case, the size of the wires' sets which must be tested (to decide whether the corresponding coefficient c_i must be incremented or not) is *a priori* not bounded, or (for efficiency reasons) is bounded by a threshold β that is not a security parameter but an efficiency one. Moreover, our testing must take intermediate failures into account. Although `maskVerif` does not directly allows to answer our specific needs, we could have exploited its rules directly in our tool with dedicated add-ons. However we wanted to provide an easy-to-understand global tool and we therefore re-implemented the common parts (essentially those enabling to decided whether a given set of wires can be simulated or not).

Optimization 1 (grouping the wires). In most of the compiled circuits that we usually considered, several wires are always assigned the same value. Grouping those wires altogether allows us to significantly reduce the number of wires to be considered by the verification tool. Let us denote by s^* the number of groups, by α_i the size of the i -th group and by w_i a representative of the i -th group. Then, Algorithm 1 can be almost directly applied to the shortened list of s^* wires (instead of s). The single main difference is that the `updateCoeffs` procedure also takes into account the weights α_i when updating the coefficients c_i . For instance, considering $h = 3$, and the tuple $(1, 1, 1, 0, \dots, 0)$ with respective weights $\alpha_1 = 2$ (for w_1), $\alpha_2 = 1$ (for w_2) and $\alpha_3 = 3$ (for w_3), the function would increase c_3 with 6, c_4 with 6, c_5 with 4 and c_6 with 1. The latter evaluation is performed using a recursive function which evaluates the number of partitions of an integer j into h parts with the constraints that each part should be at least one. When this optimization is applied, it may be observed that the `updateCoeffs` procedure also starts to update some coefficients c_i for $i > \beta$. These updated coefficients can be used as lower bounds of the final c_i values. They will be called c_i^{inf} in the rest of this paper. c_i^{sup} will be used to denote the maximal possible value for c_i , namely the binomial coefficient $\binom{s}{i}$.

Optimization 2 (using the ‘longest failure tuple’). To build all the potential failure tuples, a strategy consists in exhaustively testing all the s -tuples with Hamming weight below the Hamming weight of the longest *incompressible attack tuple*. Once this set, let say $\mathcal{U}_{\text{inc}}$, has been built, the set of all potential failure tuples can be deduced by executing the following procedure:

- for one $u_{\text{inc}} \in \mathcal{U}_{\text{inc}}$ define $\mathcal{U}_{\text{failure}} = \{u \in \{0, 1\}^s; u_{\text{inc}} \subset u\}$.
- for every new $u_{\text{inc}} \in \mathcal{U}_{\text{inc}}$, update $\mathcal{U}_{\text{failure}} = \mathcal{U}_{\text{failure}} \cup \{u \in \{0, 1\}^s; u_{\text{inc}} \subset u\}$

Implementation. An implementation of Algorithm 1 has been developed in Python. This tool, named VRAPS, has been open sourced at:

<https://github.com/CryptoExperts/VRAPS>

Small examples. In order to illustrate our automatic verification of gadgets in the random probing model, we give the list of coefficients and the subsequent failure functions obtained for three known gadgets from the literature, namely the 2-share and 3-share multiplication gadgets introduced by Ishai, Sahai, and Wagner in [20] and a 3-share multiplication gadget from [8] with an optimal number of random variables to achieve security in the 2-probing model. Descriptions for these three gadgets are given below together with an approximation of the corresponding failure function f produced by our tool. Operations are performed according to the standard priority rules. Sharings $\mathbf{x}$ and $\mathbf{y}$ denote the inputs, sharings $\mathbf{z}$ denote the outputs, and r_i are random variables. Copy gates are implicit when variables are used more than once. Hereafter $\mathcal{O}(p^5)$ is to be interpreted as p tends to 0.

2-share ISW multiplication gadget (ISW-2):

$$\begin{cases} z_0 = x_0 \cdot y_0 + r_0 \\ z_1 = x_1 \cdot y_1 + r_0 + x_0 \cdot y_1 + x_1 \cdot y_0 \end{cases} \Rightarrow f(p) = 51p^2 + 754p^3 + 4827p^4 + \mathcal{O}(p^5)$$

3-share multiplication gadget from [8] (EC16-3):

$$\begin{cases} z_0 = x_0 \cdot y_0 + r_0 + x_0 \cdot y_2 + x_2 \cdot y_0 \\ z_1 = x_1 \cdot y_1 + r_1 + x_0 \cdot y_1 + x_1 \cdot y_0 \\ z_2 = x_2 \cdot y_2 + r_0 + r_1 + x_1 \cdot y_2 + x_2 \cdot y_1 \end{cases} \Rightarrow f(p) = 1116p^3 + 44909p^4 + \mathcal{O}(p^5)$$

3-share ISW multiplication gadget (ISW-3):

$$\begin{cases} z_0 = x_0 \cdot y_0 + r_0 + r_1 \\ z_1 = x_1 \cdot y_0 + (x_0 \cdot y_1 + r_0) + x_1 \cdot y_1 + r_2 \\ z_2 = x_2 \cdot y_0 + (x_0 \cdot y_2 + r_1) + \\ \quad (x_2 \cdot y_1 + (x_1 \cdot y_2 + r_2)) + x_2 \cdot y_2 \end{cases} \Rightarrow f(p) = 1219p^3 + 55756p^4 + \mathcal{O}(p^5)$$

For our three examples, our verification tool (Algorithm 1) has been launched respectively with $\beta = s = 21$ for ISW-2, with $\beta = 9 < s = 57$ for ISW-3 and with $\beta = 13 < s = 52$ for EC16-3. In the two later cases, the missing coefficients c_i with $i > \beta$ have been either set to 0 or to $\binom{s}{i}$. This allowed us to define a lower bound $f_{\inf}$ and an upper bound $f_{\sup}$ for the functions f corresponding to ISW-3 and EC16-3. The behavior of these functions is plotted in Figures 1 to 3.

4 Composition

This section aims to provide composition properties for random-probing secure gadgets. In a nutshell, we aim to show how to build random probing secure larger circuits from specific random probing secure building blocks.

4.1 Random Probing Composability

We introduce hereafter the *random probing composability* notion for a gadget. In the following definition, for an n -share, ℓ -to- m gadget, we denote by $\mathbf{I}$ a collection of sets $\mathbf{I} = (I_1, \dots, I_\ell)$ with $I_1 \subseteq [n], \dots, I_\ell \subseteq [n]$ where $n \in \mathbb{N}$ refers to the number of shares. For some $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_\ell) \in (\mathbb{K}^n)^\ell$, we then denote $\hat{\mathbf{x}}|_{\mathbf{I}} = (\hat{x}_1|_{I_1}, \dots, \hat{x}_\ell|_{I_\ell})$ where $\hat{x}_i|_{I_i} \in \mathbb{K}^{|I_i|}$ is the tuple composed of the coordinates of the sharing $\hat{x}_i$ of indexes included in I_i .

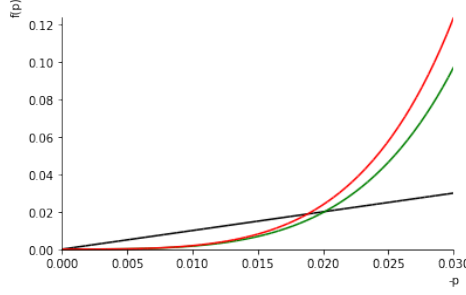


Fig. 1: Values taken by $f_{\text{inf}}(p)$ for ISW-3 (red) and EC16-3 (green) compared to the function $p \mapsto p$ (black).

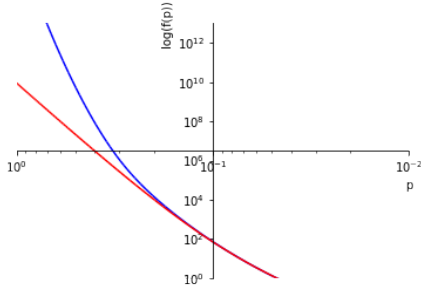


Fig. 2: Values of $\log(f_{\text{inf}}(p))$ (red) and $\log(f_{\text{sup}}(p))$ (blue) for ISW-3.

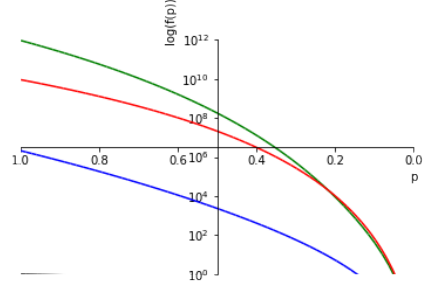


Fig. 3: For values p ranging from 1 to 0, values of $\log(f_{\text{inf}}(p))$ for ISW-3 (red) and EC16-3 (green) together with the values of $\log(f(p))$ for ISW-2 (blue).

Definition 6 (Random Probing Composability). Let $n, \ell, m \in \mathbb{N}$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is (t, p, ε) -random probing composable (RPC) for some $t \in \mathbb{N}$ and $p, \varepsilon \in [0, 1]$ if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $\hat{\mathbf{x}} \in (\mathbb{K}^n)^\ell$ and for every set collection $J_1 \subseteq [n], \dots, J_m \subseteq [n]$ of cardinals $|J_1| \leq t, \dots, |J_m| \leq t$, the random experiment

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ \mathbf{I} &\leftarrow \text{Sim}_1^G(\mathcal{W}, \mathbf{J}) \\ \text{out} &\leftarrow \text{Sim}_2^G(\hat{\mathbf{x}}|_{\mathbf{I}}) \end{aligned}$$

yields

$$\Pr((|I_1| > t) \vee \dots \vee (|I_\ell| > t)) \leq \varepsilon \quad (7)$$

and

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, \hat{\mathbf{x}}), \hat{\mathbf{y}}|_{\mathbf{J}})$$

where $\mathbf{J} = (J_1, \dots, J_m)$ and $\hat{\mathbf{y}} = G(\hat{\mathbf{x}})$. Let $f : \mathbb{R} \rightarrow \mathbb{R}$. The gadget G is (t, f) -RPC if it is $(t, p, f(p))$ -RPC for every $p \in [0, 1]$.

In the above definition, the first-pass simulator Sim_1^G determines the necessary input shares (through the returned collection of sets $\mathbf{I}$) for the second-pass simulator Sim_2^G to produce a perfect simulation of the leaking wires defined by the set $\mathcal{W}$ together with the output shares defined by the

collection of sets $\mathbf{J}$. Note that there always exists such a collection of sets $\mathbf{I}$ since $\mathbf{I} = ([n], \dots, [n])$ trivially allows a perfect simulation whatever $\mathcal{W}$ and $\mathbf{J}$. However, the goal of Sim_1^G is to return a collection of sets $\mathbf{I}$ with cardinals at most t . The idea behind this constraint is to keep the following composition invariant: for each gadget we can achieve a perfect simulation of the leaking wires plus t shares of each output sharing from t shares of each input sharing. We shall call *failure event* the event that at least one of the sets $I_1, \dots, I_\ell$ output of Sim_1^G has cardinality greater than t . When (t, p, ε) -RPC is achieved, the failure event probability is upper bounded by ε according to (7). A failure event occurs whenever Sim_2^G requires more than t shares of one input sharing to be able to produce a perfect simulation of the leaking wires (*i.e.* the wires with label in $\mathcal{W}$) together with the output shares in $\hat{\mathbf{y}}|_{\mathbf{J}}$. Whenever such a failure occurs, the composition invariant is broken. In the absence of failure event, the RPC notion implies that a perfect simulation can be achieved for the full circuit composed of RPC gadgets. This is formally stated in the next theorem.

4.2 Composition Security

Theorem 1 (Composition). *Let $t \in \mathbb{N}$, $p, \varepsilon \in [0, 1]$, and CC be a standard circuit compiler with (t, p, ε) -RPC base gadgets. For every (randomized) arithmetic circuit C composed of $|C|$ gadgets, the compiled circuit $\text{CC}(C)$ is $(p, |C| \cdot \varepsilon)$ -random probing secure. Equivalently, the standard circuit compiler CC is (p, ε) -random probing secure.*

Proof. Let $\mathcal{W}$ denote the leaking wires of the randomized circuit $\text{CC}(C)$ with probability p for each wire. We now build a simulator Sim taking as inputs $\text{CC}(C)$ and $\mathcal{W}$ and that perfectly simulates $\mathcal{W}$ with probability at least $(1 - |C| \cdot \varepsilon)$ from the simulators of the (t, p, ε) -RPC base gadgets.

We start with splitting set $\mathcal{W}$ into $|C|$ distinct subsets $\mathcal{W}_i$ for $i \in \{1, \dots, |C|\}$ such that each $\mathcal{W}_i$ stands for the output of LeakingWires when applied to the i 'th gadget G_i of $\text{CC}(C)$ with probability p . Then, we start from end gadgets whose outputs coincide with the circuit's outputs. We execute their $\text{Sim}_1^{G_i}$ with $\mathcal{W}_i$ and $J = \emptyset$, to get the sets I of required inputs. Then, we target their parents, that are gadgets whose outputs are inputs of end gadgets. For each such gadget G_i , we execute $\text{Sim}_1^{G_i}$ with $\mathcal{W}_i$ and J as defined by children sets I , to get the new sets I of required inputs. The simulation goes through the circuit from bottom to top by applying the Sim_1^G simulators to get the $\mathcal{W}_i$ and I/J sets. The simulation fails if at least one set I is of cardinal greater than t . For $|C|$ gadgets, this happens with probability $1 - (1 - \varepsilon)^{|C|} \leq |C| \cdot \varepsilon$. Otherwise, the simulation runs the Sim_2^G simulators from top to bottom by randomly picking the initial $(x_i)_I$, which completes the construction of our global simulator Sim . $\square$

4.3 Relation with Standard Probing Composition Notions

We first reformulate the Strong Non-Interference notion introduced in [5] with the formalism used for our definition of the Random Probing Composability.

Definition 7 (Strong Non-Interference (SNI)). *Let n, ℓ and t be positive integers. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow \mathbb{K}^n$ is t -SNI if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every set $J \subseteq [n]$ and subset $\mathcal{W}$ of wire labels from G satisfying $|\mathcal{W}| + |J| \leq t$, the following random experiment with any $\hat{\mathbf{x}} \in (\mathbb{K}^n)^\ell$*

$$\begin{aligned} \mathbf{I} &\leftarrow \text{Sim}_1^G(\mathcal{W}, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(\hat{\mathbf{x}}|_{\mathbf{I}}) \end{aligned}$$

yields

$$|I_1| \leq |\mathcal{W}|, \dots, |I_\ell| \leq |\mathcal{W}| \quad (8)$$

and

$$out \stackrel{id}{=} (\text{AssignWires}(G, \mathcal{W}, \widehat{\mathbf{x}}), \widehat{\mathbf{y}}|_J) \quad (9)$$

where $\mathbf{I} = (I_1, \dots, I_\ell)$ and $\widehat{\mathbf{y}} = G(\widehat{\mathbf{x}})$.

Then, we demonstrate that gadgets satisfying the t -SNI notion are also random probing composable for specific values that we explicit in the following proposition, whose proof is available in Appendix A.

Proposition 1. *Let n, ℓ and t be positive integers and let G be a gadget from $(\mathbb{K}^n)^\ell$ to $\mathbb{K}^n$. If G is t -SNI, then it is also $(t/2, p, \varepsilon)$ -RPC for any probability p and ε satisfying:*

$$\varepsilon = \sum_{i=\lfloor \frac{t}{2} + 1 \rfloor}^s \binom{s}{i} p^i (1-p)^{s-i}, \quad (10)$$

where s is the number of wires in G .

4.4 Verification of Gadget Composability

Our random probing verification tool (Algorithm 1) can be easily extended to define a simulator for the (t, p, ε) -random probing composability of a gadget for some t and some p . This essentially amounts to extend Algorithm 1 inputs with a multi-set $\mathcal{O}$ and to modify the `failureTest` procedure in order to test the simulation for each tuple in the input list ℓ_n^p augmented with the outputs coordinates with indices in $\mathcal{O}$. Then, our extended algorithm is called for every set $\mathcal{O}$ composed of at most t indices in each of the sets $J_1, \dots, J_m$. The output for the call with input set $\mathcal{O}$ is denoted by $\mathbf{c}_{\mathcal{O}} = (c_1^{\mathcal{O}}, \dots, c_\beta^{\mathcal{O}})$. For our simulator construction, the probability ε satisfies

$$\varepsilon = \sum_{i=1}^s c_i \cdot p^i \cdot (1-p)^{s-i},$$

where s denotes the number of wires in the tested gadget. Moreover, the c_i 's satisfy $c_i = \max_{\mathcal{O}} c_i^{\mathcal{O}}$.

Example. As an illustration of the proposition, let us consider the well deployed 3-share ISW multiplication gadget $G_{\text{ISW-3}} : (\mathbb{K}^3)^2 \rightarrow (\mathbb{K}^3)$ displayed in Section 3 and satisfying 2-SNI from [5]. Considering implicit copy gadgets that are mandatory in the circuit definition when a variable is reused, the corresponding circuit contains $s = 57$ wires. From Proposition 1, this gadget is also $(1, p, \varepsilon_{\text{ISW}})$ -RPC for any probability p and ε_{ISW} such that

$$\varepsilon_{\text{ISW}} = \sum_{i=2}^{57} \binom{s}{i} p^i (1-p)^{57-i}.$$

Figure 4 displays for $p \in [0, 1]$ the values taken by ε_{ISW} (in red). It also displays (in green) the values $\varepsilon'_{\text{ISW}}$ obtained by calling our verification tool on the same gadget $G_{\text{ISW-3}}$ with $\beta = 5$ (see Algorithm 1) and by replacing the missing coefficients c_i with $i > \beta$ by their upper bound $\binom{s}{i}$ (see (6)). It may be checked for small values of p the failure probability $\varepsilon'_{\text{ISW}}$ is smaller than ε_{ISW} which directly implies that the simulation induced by our verification tool is tighter than that deduced from Proposition 1.

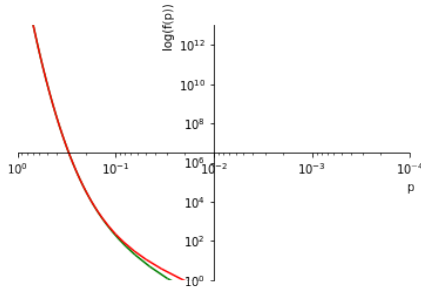


Fig. 4: Values taken by ε_{ISW} and $\varepsilon'_{\text{ISW}}$ as a function of $p \in [0, 1]$.

5 Expansion

Constructing random-probing-secure circuit compilers with a gadget expansion strategy has been proposed by Ananth, Ishai and Sahai in [2]. Such strategy was previously used in the field of multi-party computation (MPC) with different but close security goals [13, 19]. Note that such approach is called *composition* in [2] since it roughly consists in composing a base circuit compiler several times. We prefer the terminology of *expansion* here to avoid any confusion with the notion of composition for gadgets as considered in Section 4 and usual in the literature – see for instance [5, 9, 11].

We recall hereafter the general principle of the gadget expansion strategy and provide an asymptotic analysis of the so-called *expanding circuit compiler*. Then we propose an implementation of this strategy which relies on the new notion of *gadget expandability*. In contrast, the construction of [2] relies on a t -out- n secure MPC protocol in the passive security model. The advantage of our notion is that it can be achieved and/or verified by simple atomic gadgets leading to simple and efficient constructions. After introducing the gadget expandability notion, we show that it allows to achieve random-probing security with the expansion strategy. We finally explain how to adapt the verification tool described in Section 3 to this expandability notion.

5.1 Expansion Strategy

The basic principle of the gadget expansion strategy is as follows. Assume we have three n -share gadgets G_{add} , G_{mult} , G_{copy} and denote CC the standard circuit compiler for these base gadgets. We can derive three new n^2 -share gadgets by simply applying CC to each gadget: $G_{\text{add}}^{(2)} = \text{CC}(G_{\text{add}})$, $G_{\text{mult}}^{(2)} = \text{CC}(G_{\text{mult}})$ and $G_{\text{copy}}^{(2)} = \text{CC}(G_{\text{copy}})$. Let us recall that this process simply consists in replacing each addition gate in the original gadget by G_{add} , each multiplication gate by G_{mult} and each copy gate by G_{copy} , and by replacing each wire by n wires carrying a sharing of the original wire. Doing so, we obtain n^2 -share gadgets for the addition, multiplication and copy on $\mathbb{K}$. This process can be iterated an arbitrary number of times, say k , to an input circuit C :

$$C \xrightarrow{\text{CC}} \hat{C}_1 \xrightarrow{\text{CC}} \dots \xrightarrow{\text{CC}} \hat{C}_k.$$

The first output circuit $\hat{C}_1$ is the original circuit in which each gate is replaced by a base gadget G_{add} , G_{mult} or G_{copy} . The second output circuit $\hat{C}_2$ is the original circuit C in which each gate is replaced by an n^2 -share gadget $G_{\text{add}}^{(2)}$, $G_{\text{mult}}^{(2)}$ or $G_{\text{copy}}^{(2)}$ as defined above. Equivalently, $\hat{C}_2$ is the circuit $\hat{C}_1$ in which each gate is replaced by a base gadget. In the end, the output circuit $\hat{C}_k$ is hence

the original circuit C in which each gate has been replaced by a k -expanded gadget and each wire as been replaced by n^k wires carrying an (n^k) -linear sharing of the original wire. The underlying compiler is called *expanding circuit compiler* which is formally defined hereafter.

Definition 8 (Expanding Circuit Compiler). *Let CC be the standard circuit compiler with sharing order n and base gadgets $G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}$. The expanding circuit compiler with expansion level k and base compiler CC is the circuit compiler $(\text{CC}^{(k)}, \text{Enc}^{(k)}, \text{Dec}^{(k)})$ satisfying the following:*

1. *The input encoding $\text{Enc}^{(k)}$ is an (n^k) -linear encoding.*
2. *The output decoding Dec is the (n^k) -linear decoding mapping.*
3. *The circuit compilation is defined as*

$$\text{CC}^{(k)}(\cdot) = \underbrace{\text{CC} \circ \text{CC} \circ \dots \circ \text{CC}}_{k \text{ times}}(\cdot)$$

The goal of the expansion strategy in the context of random probing security is to replace the leakage probability p of a wire in the original circuit by the failure event probability ε in the subsequent gadget simulation. If this simulation fails then one needs the full input sharing for the gadget simulation, which corresponds to leaking the corresponding wire value in the base case. The security is thus amplified by replacing the probability p in the base case by the probability ε (assuming that we have $\varepsilon < p$). If the failure event probability ε can be upper bounded by some function of the leakage probability: $\varepsilon < f(p)$ for every leakage probability $p \in [0, p_{\max}]$ for some $p_{\max} < 1$, then the expanding circuit compiler with expansion level k shall result in a security amplification as

$$p = \varepsilon_0 \xrightarrow{f} \varepsilon_1 \xrightarrow{f} \dots \xrightarrow{f} \varepsilon_k = f^{(k)}(p) ,$$

which for an adequate function f (e.g. $f : p \mapsto p^2$) provides exponential security. In order to get such a security expansion, the gadgets must satisfy a stronger notion than the composability notion introduced in Section 4 which we call *random probing expandability*; see Section 5.3 below.

5.2 Asymptotic Analysis of the Expanding Compiler

In this section we show that the asymptotic complexity of a compiled circuit $\widehat{C} = \text{CC}^{(k)}(C)$ is $|\widehat{C}| = \mathcal{O}(|C| \cdot \kappa^e)$ for security parameter κ , for some constant e that we make explicit.

Let us denote by $\mathbf{N} = (N_a, N_c, N_m, N_r)^\top$ the column vector of gate counts for some base gadget G , where N_a, N_c, N_m, N_r stands for the number of addition gates, copy gates, multiplication gates and random gates respectively. We have three different such vectors

$$\begin{aligned} \mathbf{N}_{\text{add}} &\doteq (N_{\text{add},a}, N_{\text{add},c}, N_{\text{add},m}, N_{\text{add},r})^\top \\ \mathbf{N}_{\text{mult}} &\doteq (N_{\text{mult},a}, N_{\text{mult},c}, N_{\text{mult},m}, N_{\text{mult},r})^\top \\ \mathbf{N}_{\text{copy}} &\doteq (N_{\text{copy},a}, N_{\text{copy},c}, N_{\text{copy},m}, N_{\text{copy},r})^\top \end{aligned}$$

for the gate counts respectively in the base addition gadget G_{add} , in the base multiplication gadget G_{mult} and in the base copy gadgets G_{copy} . Let us define the 4×4 square matrix $\mathbf{M}$ as

$$\mathbf{M} = (\mathbf{N}_{\text{add}} \mid \mathbf{N}_{\text{copy}} \mid \mathbf{N}_{\text{mult}} \mid \mathbf{N}_{\text{rand}}) \quad \text{with} \quad \mathbf{N}_{\text{rand}} = (0, 0, 0, n)^\top ,$$

where the definition $\mathbf{N}_{\text{rand}}$ holds from the fact that the standard circuit compiler replaces each random gate by n random gates.

It can be checked that applying the standard circuit compiler with base gadgets G_{add} , G_{mult} and G_{copy} to some circuit C with gate-count vector $\mathbf{N}_C$ gives a circuit $\widehat{C}$ with gate-count vector $\mathbf{N}_{\widehat{C}} = \mathbf{M} \cdot \mathbf{N}_C$. It follows that the k th power of the matrix $\mathbf{M}$ gives the gate counts for the level- k gadgets as:

$$\mathbf{M}^k = \underbrace{\mathbf{M} \cdot \mathbf{M} \cdots \mathbf{M}}_{k \text{ times}} = (\mathbf{N}_{\text{add}}^{(k)} \mid \mathbf{N}_{\text{copy}}^{(k)} \mid \mathbf{N}_{\text{mult}}^{(k)} \mid \mathbf{N}_{\text{rand}}^{(k)}) \quad \text{with} \quad \mathbf{N}_{\text{rand}}^{(k)} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ n^k \end{pmatrix}$$

where $\mathbf{N}_{\text{add}}^{(k)}$, $\mathbf{N}_{\text{mult}}^{(k)}$ and $\mathbf{N}_{\text{copy}}^{(k)}$ are the gate-count vectors for the level- k gadgets $G_{\text{add}}^{(k)}$, $G_{\text{mult}}^{(k)}$ and $G_{\text{copy}}^{(k)}$ respectively. Let us denote the eigen decomposition of $\mathbf{M}$ as $\mathbf{M} = \mathbf{Q} \cdot \mathbf{\Lambda} \cdot \mathbf{Q}^{-1}$, we get

$$\mathbf{M}^k = \mathbf{Q} \cdot \mathbf{\Lambda}^k \cdot \mathbf{Q}^{-1} \quad \text{with} \quad \mathbf{\Lambda}^k = \begin{pmatrix} \lambda_1^k & & & \\ & \lambda_2^k & & \\ & & \lambda_3^k & \\ & & & \lambda_4^k \end{pmatrix}$$

where $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ are the eigenvalues of $\mathbf{M}$. We then obtain an asymptotic complexity of

$$|\widehat{C}| = \mathcal{O}(|C| \cdot (\lambda_1^k + \lambda_2^k + \lambda_3^k + \lambda_4^k)) = \mathcal{O}(|C| \cdot \max(\lambda_1, \lambda_2, \lambda_3, \lambda_4)^k)$$

for a compiled circuit $\widehat{C} = \text{CC}^{(k)}(C)$ (where the constant in the $\mathcal{O}(\cdot)$ depends on $\mathbf{Q}$ and shall be fairly small).

Interestingly, if multiplication gates are solely used in the multiplication gadget (*i.e.* $N_{\text{add},m} = N_{\text{copy},m} = 0$) which is the case in the constructions we consider in this paper, it can be checked that (up to some permutation) the eigenvalues satisfy

$$(\lambda_1, \lambda_2) = \text{eigenvalues}(\mathbf{M}_{ac}), \quad \lambda_3 = N_{\text{mult},m}^k \quad \text{and} \quad \lambda_4 = n^k$$

where $\mathbf{M}_{ac}$ is the top left 2×2 block matrix of $\mathbf{M}$ *i.e.*

$$\mathbf{M}_{ac} = \begin{pmatrix} N_{\text{add},a} & N_{\text{copy},a} \\ N_{\text{add},c} & N_{\text{copy},c} \end{pmatrix}.$$

We finally get

$$|\widehat{C}| = \mathcal{O}(|C| \cdot N_{\text{max}}^k) \quad \text{with} \quad N_{\text{max}} = \max(\text{eigenvalues}(\mathbf{M}_{ac}), N_{\text{mult},m}) . \quad (11)$$

In order to reach some security level $\varepsilon = 2^{-\kappa}$ for some target security parameter κ and assuming that we have a security expansion $p \rightarrow f^{(k)}(p)$, the expansion level k must be chosen so that $f^{(k)}(p) \leq 2^{-\kappa}$. In practice, the function f is of the form

$$f : p \mapsto \sum_{i \geq d} c_i p^i \leq (c_d + \mathcal{O}(p)) p^d .$$

where $\mathcal{O}(p)$ is to be interpreted as p tends to 0. In the rest of this paper, we shall say that such a function has *amplification order* d .

The upper bound $f(p) \leq c'_d p^d$ with $c'_d = c_d + \mathcal{O}(p)$ implies $f^{(k)}(p) < (c'_d p)^{d^k}$. Hence, to satisfy the required security $f^{(k)}(p) \leq 2^{-\kappa}$ while assuming $c'_d p < 1$, the number k of expansions must satisfy:

$$k \geq \log_d(\kappa) - \log_d(-\log_2(c'_d p)) .$$

We can then rewrite (11) as

$$|\widehat{C}| = \mathcal{O}(|C| \cdot \kappa^e) \quad \text{with} \quad e = \frac{\log N_{\max}}{\log d} . \quad (12)$$

5.3 Random Probing Expandability

In the evaluation of random probing composability, let us recall that the failure event in the simulation of a gadget means that more than t shares from one of its inputs are necessary to complete a perfect simulation. For a gadget to be expandable we need slightly stronger notions than random probing composability. As first requirement, a two-input gadget should have a failure probability which is independent for each input. This is because in the base case, each wire as input of a gate leaks independently. On the other hand, in case of failure event in the child gadget, the overall simulator should be able to produce a perfect simulation of the full output (that is the full input for which the failure occurs). To do so, the overall simulator is given the clear output (which is obtained from the simulation of the base case) plus any set of $n - 1$ output shares. This means that whenever the set J is of cardinal greater than t , the gadget simulator can replace it by any set J' of cardinal $n - 1$.

Definition 9 (Random Probing Expandability). *Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -random probing expandable (RPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\widehat{x}, \widehat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2, J') &\leftarrow \text{Sim}_1^G(\mathcal{W}, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J', \widehat{x}|_{I_1}, \widehat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (|I_1| > t)$ and $\mathcal{F}_2 \equiv (|I_2| > t)$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \quad (13)$$

- with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),
2. J' is such that $J' = J$ if $|J| \leq t$ and $J' \subseteq [n]$ with $|J'| = n - 1$ otherwise,
3. the output distribution satisfies

$$\text{out} \stackrel{id}{=} (\text{AssignWires}(G, \mathcal{W}, (\widehat{x}, \widehat{y})), \widehat{z}|_{J'}) \quad (14)$$

where $\widehat{z} = G(\widehat{x}, \widehat{y})$.

The RPE notion can be simply extended to gadgets with 2 outputs: the Sim_1^G simulator takes two sets $J_1 \subseteq [n]$ and $J_2 \subseteq [n]$ as input and produces two sets J'_1 and J'_2 satisfying the same property as J' in the above definition (w.r.t. J_1 and J_2). The Sim_2^G simulator must then produce an output including $\hat{z}_1|_{J'_1}$ and $\hat{z}_2|_{J'_1}$ where $\hat{z}_1$ and $\hat{z}_2$ are the output sharings. The RPE notion can also be simply extended to gadgets with a single input: the Sim_1^G simulator produces a single set I so that the failure event ($|I| > t$) occurs with probability lower than ε (and the Sim_2^G simulator is then simply given $\hat{x}|_I$ where $\hat{x}$ is the single input sharing). For the sake of completeness, and since we only focus in $2 \rightarrow 1$ and $1 \rightarrow 2$ gadgets in this paper, the RPE definition for the $1 \rightarrow 2$ case is given in Appendix B.

It is not hard to check that the above expandability notion is stronger than the composability notion introduced in Section 4. Formally, we have the following reduction:

Proposition 2. *Let $f = \mathbb{R} \rightarrow \mathbb{R}$ and $n \in \mathbb{N}$. Let G be an n -share gadget. If G is (t, f) -RPE then G is (t, f') -RPC, with $f'(\cdot) = 2 \cdot f(\cdot)$.*

Proof. We consider a (t, f) -RPE n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$. The $(t, 2 \cdot f)$ -random composability property is directly implied by the (t, f) -random probing expandability by making use of the exact same simulators and observing that

$$\Pr(|I_1| > t \vee |I_2| > t) \leq \Pr(|I_1| > t) + \Pr(|I_2| > t) = 2 \cdot \varepsilon.$$

The case of $1 \rightarrow 2$ gadgets is even more direct. □

5.4 Expansion Security

Definition 9 of random probing expandability is valid for base gadgets. For level- k gadgets $G^{(k)} = \text{CC}^{(k-1)}(G)$ where $G \in \{G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}\}$ is a base gadget, we provide a generalized definition of random probing expandability.

Adequate subsets of $[n^k]$. We first define the notion of “adequate” subsets of $[n^k]$, instead of only bounded subsets. For this we define recursively a family $S_k \in \mathcal{P}([n^k])$, where $\mathcal{P}([n^k])$ denotes the set of all subsets of $[n^k]$, as follows:

$$\begin{aligned} S_1 &= \{I \in [n], |I| \leq t\} \\ S_k &= \{(I_1, \dots, I_n) \in (S_{k-1} \cup [n^{k-1}])^n, I_j \in S_{k-1} \ \forall j \in [1, n] \text{ except at most } t\} \end{aligned}$$

In other words, a subset I belongs to S_k if among the n subset parts of I , at most t of them are full, while the other ones recursively belong to S_{k-1} ; see Figure 9 in Appendix C.1 for an illustration with $n = 3$ and $t = 1$.

Generalized definition of Random Probing Expandability. We generalize Definition 9 as follows. At level k the input sets I_1 and I_2 must belong to S_k , otherwise we have a failure event. As in Definition 9, the simulation is performed for an output subset J' with $J' = J$ if $J \in S_k$, otherwise $J' = [n^k] \setminus \{j^*\}$ for some $j^* \in [n^k]$.

Definition 10 (Random Probing Expandability with $\{S_k\}_{k \in \mathbb{N}}$). Let $f : \mathbb{R} \rightarrow \mathbb{R}$ and $k \in \mathbb{N}$. An n^k -share gadget $G : \mathbb{K}^{n^k} \times \mathbb{K}^{n^k} \rightarrow \mathbb{K}^{n^k}$ is (S_k, f) -random probing expandable (RPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^{n^k} \times \mathbb{K}^{n^k}$, for every set $J \in S_k \cup [n^k]$ and for every $p \in [0, 1]$, the random experiment

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2, J') &\leftarrow \text{Sim}_1^G(\mathcal{W}, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J', \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (I_1 \notin S_k)$ and $\mathcal{F}_2 \equiv (I_2 \notin S_k)$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \quad (15)$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. the set J' is such that $J' = J$ if $J \in S_k$, and $J' = [n^k] \setminus \{j^*\}$ for some $j^* \in [n^k]$ otherwise,
3. the output distribution satisfies

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, (\hat{x}, \hat{y})), \hat{z}|_{J'}) \quad (16)$$

where $\hat{z} = G(\hat{x}, \hat{y})$.

The notion of random probing expandability from Definition 10 naturally leads to the statement of our main theorem; the proof is given in Appendix C.1.

Theorem 2. Let $n \in \mathbb{N}$ and $f : \mathbb{R} \rightarrow \mathbb{R}$. Let $G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}$ be n -share gadgets for the addition, multiplication and copy on $\mathbb{K}$. Let CC be the standard circuit compiler with sharing order n and base gadgets $G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}$. Let $\text{CC}^{(k)}$ be the expanding circuit compiler with base compiler CC . If the base gadgets $G_{\text{add}}, G_{\text{mult}}$ and G_{copy} are (t, f) -RPE then, $G_{\text{add}}^{(k)} = \text{CC}^{(k-1)}(G_{\text{add}})$, $G_{\text{mult}}^{(k)} = \text{CC}^{(k-1)}(G_{\text{mult}})$, $G_{\text{copy}}^{(k)} = \text{CC}^{(k-1)}(G_{\text{copy}})$ are $(S_k, f^{(k)})$ -RPE, n^k -share gadgets for the addition, multiplication and copy on $\mathbb{K}$.

The random probing security of the expanding circuit compiler can then be deduced as a corollary of the above theorem together with Proposition 2 (RPE $\Rightarrow$ RPC reduction) and Theorem 1 (composition theorem).

Corollary 1. Let $n \in \mathbb{N}$ and $f : \mathbb{R} \rightarrow \mathbb{R}$. Let $G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}$ be n -share gadgets for the addition, multiplication and copy on $\mathbb{K}$. Let CC be the standard circuit compiler with sharing order n and base gadgets $G_{\text{add}}, G_{\text{mult}}, G_{\text{copy}}$. Let $\text{CC}^{(k)}$ be the expanding circuit compiler with base compiler CC . If the base gadgets $G_{\text{add}}, G_{\text{mult}}$ and G_{copy} are (t, f) -RPE then $\text{CC}^{(k)}$ is $(p, 2 \cdot f^{(k)}(p))$ -random probing secure.

5.5 Relaxing the Expandability Notion

The requirement of the RPE property that the failure events $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent might seem too strong. In practice it might be easier to show or verify that some gadgets satisfy a weaker notion. We say that a gadget is (t, f) -weak random probing expandable (wRPE) if the failure events verify $\Pr(\mathcal{F}_1) \leq \varepsilon$, $\Pr(\mathcal{F}_2) \leq \varepsilon$ and $\Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) \leq \varepsilon^2$ instead of (22) in Definition 9. Although being easier to achieve and to verify this notion is actually not much weaker as the original RPE. We have the following reduction of RPE to wRPE; see Appendix C.3 for the proof.

Proposition 3. *Let $f = \mathbb{R} \rightarrow [0, 0.14]$. Let $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share gadget. If G is (t, f) -wRPE then G is (t, f') -RPE with $f'(\cdot) = f(\cdot) + \frac{3}{2}f(\cdot)^2$.*

Assume that we can show or verify that a gadget is wRPE with the following failure event probabilities

$$\Pr(\mathcal{F}_1) = f_1(p), \quad \Pr(\mathcal{F}_2) = f_2(p) \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = f_{12}(p),$$

for every $p \in [0, 1]$. Then the above proposition implies that the gadget is (p, f) -RPE with

$$f : p \mapsto f_{\max}(p) + \frac{3}{2}f_{\max}(p)^2 \quad \text{with} \quad f_{\max} = \max(f_1, f_2, \sqrt{f_{12}}).$$

We shall base our verification of the RPE property on the above equation as we describe hereafter.

5.6 Verification of Gadget Expandability

We can easily adapt our automatic tool to verify the weak random probing expandability for base gadgets (Definition 9). Basically, the verification is split into two steps that we first describe for the case of addition and multiplication gadgets with fan-in 2 and fan-out 1.

In a first step, our tool computes the function f to check the (t, f) -wRPE property for output sets of shares of cardinal at most t . For 2-input gadgets, this step leads to the computation of coefficients c_i corresponding to three failure events $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_1 \wedge \mathcal{F}_2$ as defined above but restricted to output sets of shares of cardinal less than t . The process is very similar to the verification of random probing composability but requires to separate the failure events counter into failure events for the first input ($|\mathcal{I}_1| > t$), for the second input ($|\mathcal{I}_2| > t$) or for both ($(|\mathcal{I}_1| > t) \wedge (|\mathcal{I}_2| > t)$). In the following, we denote the three functions formed from the corresponding coefficients as $f_1^{(1)}$, $f_2^{(1)}$, and $f_{12}^{(1)}$.

Then, in a second step, our tool verifies that there exists at least one set of $n - 1$ shares for each output, such that the simulation failure is limited by $f(p)$ for some probability $p \in [0, 1]$. In that case, it still loops on the possible output sets of shares (of cardinal $n - 1$) but instead of computing the maximum coefficients, it determines whether the simulation succeeds for at least one of such sets. A failure event is recorded for a given tuple if no output sets of cardinal $n - 1$ can be simulated together with this tuple from at most t shares of each input. As for the first verification step, we record the resulting coefficients for the three failure events to obtain functions $f_1^{(2)}$, $f_2^{(2)}$, and $f_{12}^{(2)}$.

From these two steps, we can deduce f such that the gadget is (t, f) -wRPE:

$$\forall p \in [0, 1], f(p) = \max(f_1(p), f_2(p), \sqrt{f_{12}(p)})$$

with

$$f_\alpha(p) = \max(f_\alpha^{(1)}(p), f_\alpha^{(2)}(p)) \quad \text{for} \quad \alpha \in \{1, 2, 12\}$$

The computation of f for a gadget to satisfy (t, f) -weak random probing expandability is a bit trickier for copy gadgets which produce two outputs. Instead of two verification steps considering both possible ranges of cardinals for the output set of shares J , we need to consider four scenarios for the two possible features for output sets of shares J_1 and J_2 . In a nutshell, the idea is to follow the first verification step described above when both J_1 and J_2 have cardinal equal or less than t and to follow the second verification step described above when both J_1 and J_2 have greater cardinals. This leads to functions $f^{(1)}$ and $f^{(2)}$. Then, two extra cases are to be considered, namely

when $(|J_1| \leq t)$ and $(|J_2| > t)$ and the reverse when $(|J_1| > t)$ and $(|J_2| \leq t)$. To handle these scenarios, our tool loops over the output sets of shares of cardinal equal or less than t for the first output, and it determines whether there exists a set of $n - 1$ shares of the second output that a simulator can perfectly simulate with the leaking wires and the former set. This leads to function $f^{(12)}$ and reversely to function $f^{(21)}$. From these four verification steps, we can deduce f such that the copy gadget is (t, f) -wRPE:

$$\forall p \in [0, 1], f(p) = \max(f^{(1)}(p), f^{(2)}(p), f^{(12)}(p), f^{(21)}(p)).$$

Once gadgets have been proven (t, f) -weak RPE, they are also proven to be (t, f') -RPE from Proposition 3 with $f' : p \mapsto f(p) + \frac{3}{2}f(p)^2$. Examples of such computations for 3-share gadgets are provided in Section 6.

6 New Constructions

In this section, we exhibit and analyze $(1, f)$ -wRPE gadgets for the addition, multiplication, and copy (on any base field $\mathbb{K}$) to instantiate the expanding circuit compiler. These gadgets are sound in the sense that their function f has amplification order strictly greater than one. As explained in previous sections, an amplification order strictly greater than one guarantees that there exists a probability $p_{max} \in [0, 1]$ such that $\forall p \leq p_{max}, f(p) \leq p$, which is necessary to benefit from the expansion. For 2-input gadgets, f is defined as the maximum between f_1 , f_2 , and $\sqrt{f_{12}}$. Therefore, the constraint on the amplification order also applies to the functions f_1 , f_2 , and $\sqrt{f_{12}}$. For the function f_{12} , this means that the amplification order should be strictly greater than two.

We start hereafter with an impossibility result, namely there are no (2-share, 2-to-1) $(1, f)$ -RPE gadgets such that f has an amplification order greater than one. Then, we provide concrete instantiations of addition, multiplication, and copy gadgets based on 3 shares which successfully achieve $(1, f)$ -RPE for amplification order greater than one and can be used in the expansion compiler.

6.1 About 2-Share Gadgets

Consider a gadget G with a 2-share single output $\mathbf{z} = (z_0, z_1)$ and two 2-share inputs $\mathbf{x} = (x_0, x_1)$ and $\mathbf{y} = (y_0, y_1)$. We reasonably assume that the latter are the outputs of gates with fan-in at most two (and not direct input shares). For G to be $(1, f)$ -RPE with f of amplification order strictly greater than one, then f_{12} must be of amplification strictly greater than two. In other words, we should be able to exhibit a simulator such that one share of each input is enough to simulate anyone of the output shares and an arbitrary couple of leaking wires. But the output wire z_0 and both input gates of the second output share z_1 represent the full output and require the knowledge of both inputs to be simulated. Therefore, f_{12} has a non-zero coefficient in p and is thus not of amplification order strictly greater than two. We thus restrict our investigation to n -share gadgets, with $n \geq 3$ to instantiate our compiler.

In the upcoming gadget descriptions, notice that variables r_i are fresh random values, operations are processed with the usual priority rules, and the number of implicit copy gates can be deduced from the occurrences of each intermediate variable such that n occurrences require $n - 1$ implicit copy gates. Also, the function expression below each gadget corresponds to the function obtained from our verification tool when verifying weak random probing expandability. It implies that the gadget is (t, f) -wRPE for t usually equal to one except when defined otherwise. A more complete description of each function (with more coefficients) is available in Appendix D.1.

6.2 Addition Gadgets

The most classical masked addition schemes are sharewise additions which satisfy the simpler probing security property. Basically, given two input n -sharings $\mathbf{x}$ and $\mathbf{y}$, such an addition computes the output n -sharing $\mathbf{z}$ as $z_1 \leftarrow x_1 + y_1, z_2 \leftarrow x_2 + y_2, \dots, z_n \leftarrow x_n + y_n$. Unfortunately, such elementary gadgets do not work in our setting. Namely consider an output set of shares J of cardinality t . Then, for any n , there exists sets $\mathcal{W}$ of leaking wires of cardinality one such that no set I of cardinality $\leq t$ can point to input shares that are enough to simulate both the leaking wire and the output shares of indexes in J . For instance, given a set $J = \{1, \dots, t\}$, if $\mathcal{W}$ contains x_{t+1} , then no set I of cardinal $\leq t$ can define a set of input shares from which we can simulate both the leaking wire and $z_1, \dots, z_t$. Indeed, each z_i for $1 \leq i \leq t$ requires both input shares x_i and y_i for its simulation. Thus, a simulation set I would contain at least $\{1, \dots, t\}$ and $t+1$ for the simulation of the leaking wire. I would thus be of cardinal $t+1$ which represents a failure event in the random probing expandability definition. As a consequence, such a n -share addition gadget could only be (t, f) -RPE with f with a first coefficient c_1 as defined in Section 3 strictly positive. In other words, f would be of amplification order one such that $\forall p \in [0, 1], f(p) \geq p$.

In the following, we introduce two 3-share addition gadgets. From our automatic tool, both are $(1, f)$ -wRPE with f of amplification order strictly greater than one. Basically, in our first addition gadget G_{add}^1 , both inputs are first refreshed with a circular refreshing gadget as originally introduced in [6]:

$$\begin{aligned} G_{\text{add}}^1 : z_0 &\leftarrow x_0 + r_0 + r_1 + y_0 + r_3 + r_4 \\ z_1 &\leftarrow x_1 + r_1 + r_2 + y_1 + r_4 + r_5 & f_{\max}(p) = \sqrt{10}p^{3/2} + \mathcal{O}(p^2) \\ z_2 &\leftarrow x_2 + r_2 + r_0 + y_2 + r_5 + r_3 \end{aligned}$$

The second addition gadget G_{add}^2 simply rearranges the order of the refreshing variables:

$$\begin{aligned} G_{\text{add}}^2 : z_0 &\leftarrow x_0 + r_0 + r_4 + y_0 + r_1 + r_3 \\ z_1 &\leftarrow x_1 + r_1 + r_5 + y_1 + r_2 + r_4 & f_{\max}(p) = \sqrt{69}p^2 + \mathcal{O}(p^3) \\ z_2 &\leftarrow x_2 + r_2 + r_3 + y_2 + r_0 + r_5 \end{aligned}$$

In each gadget, $\mathbf{x}$ and $\mathbf{y}$ are the input sharings and $\mathbf{z}$ the output sharing; $f_{\max}$ additionally reports the maximum of the first non zero coefficient (as defined in Section 3) of the three functions f_1, f_2 , and f_{12} , as defined in the previous section, obtained for the random probing expandability automatic verifications. A further definition of these functions can be found in Appendix D.1. Note that both gadgets G_{add}^1 and G_{add}^2 are built with 15 addition gates and 6 implicit copy gates.

6.3 Multiplication Gadget

We start by proving an impossibility result: no 3-share multiplication gadget composed of direct products between input shares satisfies $(1, f)$ -RPE with amplification order strictly greater than one. Consider such a gadget G with two 3-input sharings $\mathbf{x}$ and $\mathbf{y}$ whose shares are directly multiplied together. Let $(x_i \cdot y_j)$ and $(x_k \cdot y_\ell)$ be two such products such that $i, j, k, \ell \in [3]$ and $i \neq k$ and $j \neq \ell$. If both results are leaking, then the leakage can only be simulated using the four input shares. Namely, $\{i, k\} \subseteq I_1$ and $\{j, \ell\} \subseteq I_2$. This scenario represents a failure since cardinals of I_1 and I_2 are both strictly greater than one. As a consequence, function f_{12} which records the

failures for both inputs is defined with a coefficient c_2 at least equal to one. Hence f_{12} is not of amplification greater than two and f cannot be of amplification order greater than one. Regular 3-share multiplication gadgets consequently cannot be used as base gadgets of our compiler.

To circumvent this issue, we build a 3-share multiplication gadget G_{mult}^1 whose both inputs are first refreshed, before any multiplication is performed:

$$\begin{aligned} u_0 &\leftarrow x_0 + r_5 + r_6; & u_1 &\leftarrow x_1 + r_6 + r_7; & u_2 &\leftarrow x_2 + r_7 + r_5 \\ v_0 &\leftarrow y_0 + r_8 + r_9; & v_1 &\leftarrow y_1 + r_9 + r_{10}; & v_2 &\leftarrow y_2 + r_{10} + r_8 \end{aligned}$$

$$\begin{aligned} z_0 &\leftarrow (u_0 \cdot v_0 + r_0) + (u_0 \cdot v_1 + r_1) + (u_0 \cdot v_2 + r_2) \\ z_1 &\leftarrow (u_1 \cdot v_0 + r_1) + (u_1 \cdot v_1 + r_4) + (u_1 \cdot v_2 + r_3) \\ z_2 &\leftarrow (u_2 \cdot v_0 + r_2) + (u_2 \cdot v_1 + r_3) + (u_2 \cdot v_2 + r_0) + r_4 \end{aligned}$$

$$f_{\max}(p) = \sqrt{83}p^{3/2} + \mathcal{O}(p^2)$$

6.4 Copy Gadget

We exhibit a 3-share $(1, f)$ -wRPE copy gadget G_{copy}^1 with f of amplification order strictly greater than one:

$$\begin{aligned} v_0 &\leftarrow u_0 + r_0 + r_1; & w_0 &\leftarrow u_0 + r_3 + r_4 \\ v_1 &\leftarrow u_1 + r_1 + r_2; & w_1 &\leftarrow u_1 + r_4 + r_5 & f_{\max}(p) &= 33p^2 + \mathcal{O}(p^3) \\ v_2 &\leftarrow u_2 + r_2 + r_0; & w_2 &\leftarrow u_2 + r_5 + r_3 \end{aligned}$$

It simply relies on two calls of the circular refreshing from [6] on the input. This last gadget is made of 6 addition gates and 9 implicit copy gates.

6.5 Complexity and Tolerated Probability

Following the asymptotic analysis of Section 5.2, our construction yields the following instantiation of the matrix $\mathbf{M}$

$$\mathbf{M} = \begin{pmatrix} 15 & 12 & 28 & 0 \\ 6 & 9 & 23 & 0 \\ 0 & 0 & 9 & 0 \\ 6 & 6 & 11 & 3 \end{pmatrix} \quad (17)$$

with

$$\mathbf{M}_{ac} = \begin{pmatrix} 15 & 12 \\ 6 & 9 \end{pmatrix} \quad \text{and} \quad N_{\text{mult},m} = 9 .$$

The eigenvalues of $\mathbf{M}_{ac}$ are 3 and 21, which gives $N_{\max} = 21$. We also have a random probing expandability with function f of amplification order $d = \frac{3}{2}$. Hence we get

$$e = \frac{\log N_{\max}}{\log d} = \frac{\log 21}{\log 1.5} \approx 7.5$$

which gives a complexity of $|\hat{C}| = \mathcal{O}(|C| \cdot \kappa^{7.5})$. Finally, it can be checked from the coefficients of the RPE functions given in Appendix D that our construction tolerates a leakage probability up to

$$p_{\max} \approx 0.0045 > 2^{-8}.$$

This corresponds to the maximum value p for which we have $f(p) < p$ which is a necessary and sufficient condition for the expansion strategy to apply with (t, f) -RPE gadgets.

As explained in Sec. 5.2, we can compute the new gate count vectors for each of the compiled gadgets $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$ by computing the matrix $\mathbf{M}^k$. In Fig. 5, we plot the total number of gates ($N_a + N_c + N_m + N_r$) in each of the compiled gadgets as a function of the level k . For instance, for level $k = 9$ the number of gates in the compiled gadgets is around 10^{12} . For the latter level and assuming a leakage probability of $p = 0.0045$ (which is the maximum we can tolerate), we achieve a security of $\varepsilon \approx 2^{-76}$. On its right side, Fig. 6 plots the values taken by the function f such that the gadgets G_{add}^1 , G_{add}^2 , G_{mult}^1 and G_{copy}^1 are (t, f) -RPE.

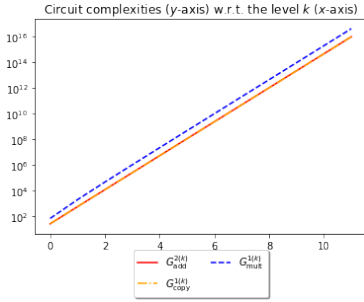


Fig. 5: Number of gates for $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$ circuits with respect to the level k .

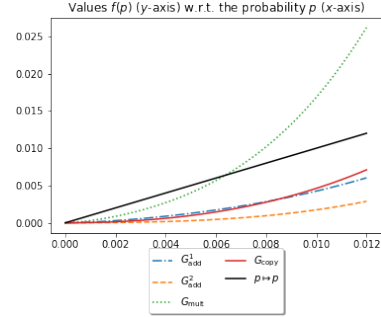


Fig. 6: Values taken by the function f for (t, f) -RPE

7 Comparison with Previous Constructions

In this section, we compare our scheme to previous constructions. Specifically, we first compare it to the well-known Ishai-Sahai-Wagner (ISW) construction and discuss the instantiation of our scheme from the ISW multiplication gadget. Then we exhibit the asymptotic complexity (and tolerated leakage probability) of the Ananth-Ishai-Sahai compiler and compare their results to our instantiation.

7.1 Comparison with ISW

The classical ISW construction [20] is secure in the t -probing model when the adversary can learn any set of t intermediate variables in the circuit, for $n = 2t + 1$ shares. This can be extended to t probes per gadget, where each gadget corresponds to a AND or XOR gate in the original circuit. Using Chernoff bound, security in the t -probing model per gadget implies security in the p -random probing model, where each wire leaks with probability p , with $p = \mathcal{O}(t/|G|)$, where $|G|$ is the gadget size. Since in ISW each gadget has complexity $\mathcal{O}(t^2)$, this gives $p = \mathcal{O}(1/t)$. Therefore, in

the p -random probing model, the ISW construction is only secure against a leakage probability $p = \mathcal{O}(1/n)$, where the number of shares n must grow linearly with the security parameter κ in order to achieve security $2^{-\kappa}$. This means that ISW does not achieve security under a constant leakage probability p ; this explains why ISW is actually vulnerable to horizontal attacks [7], in which the adversary can combine information from a constant fraction of the wires.

ISW-based instantiation of the expanding compiler. In our instantiation, we choose to construct a new 3-share multiplication gadget instead of using the ISW multiplication gadget from [20]. In fact, ISW first performs a direct product of the secret shares before adding some randomness, while we proved in Section 6 that no such 3-share multiplication gadget made of direct products could satisfy $(1, f)$ -RPE with amplification order strictly greater than one. Therefore the ISW gadget is not adapted for our construction with 3 shares.

Table 1 displays the output of our tool when run on the ISW gadget for up to 7 shares with different values for t . It can be seen that an amplification order strictly greater than one is only achieved for $t > 1$, with 4 or more shares. And an order of $3/2$ is only achieved with a minimum of 4 shares for $t = 2$, whereas we already reached this order with our 3-share construction for $t = 1$. If we use the 4-share ISW gadget with appropriate 4-share addition and copy gadgets instead of our instantiation, the overall complexity of the compiler would be greater, while the amplification order would remain the same, and the tolerated leakage probability would be worse (recall that our instantiation tolerates a maximum leakage probability $p \approx 2^{-8}$, while 4-share ISW tolerates $p \approx 2^{-9.83}$). Clearly, the complexity of the 4-share ISW gadget $(N_a, N_c, N_m, N_r) = (24, 30, 16, 6)$ is higher than that of our 3-share multiplication gadget $(N_a, N_c, N_m, N_r) = (28, 23, 9, 11)$. In addition, using 3-share addition and copy gadgets (as in our case) provides better complexity than 4-share gadgets. Hence to reach an amplification order of $3/2$, a 4-share construction with the ISW gadget would be more complex and would offer a lower tolerated leakage probability.

For higher amplification orders, the ISW gadgets with more than 4 shares or other gadgets can be studied. This is an open construction problem as many gadgets can achieve different amplification orders and be globally compared.

7.2 Complexity of the Ananth-Ishai-Sahai Compiler

The work from [2] provides a construction of circuit compiler (the AIS compiler) based on the expansion strategy described in Section 5 with a (p, ε) -composable security property, analogous to our (t, f) -RPE property. To this purpose, the authors use an (m, c) -multi-party computation (MPC) protocol Π . Such a protocol allows to securely compute a functionality shared among m parties and tolerating at most c corruptions. In a nutshell, their composable circuit compiler consists of multiple layers: the bottom layer replaces each gate in the circuit by a circuit computing the (m, c) -MPC protocol for the corresponding functionality (either Boolean addition, Boolean multiplication, or copy). The next $k - 1$ above layers apply the same strategy recursively to each of the resulting gates. As this application can eventually have exponential complexity if applied to a whole circuit C directly, the top layer of compilation actually applies the k bottom layers to each of the gates of C independently and then stitches the inputs and outputs using the correctness of the XOR-encoding property. Hence the complexity is in

$$\mathcal{O}(|C| \cdot N_g^k), \quad (18)$$

where $|C|$ is the number of gates in the original circuit and N_g is the number of gates in the circuit computing Π . The authors of [2] prove that such compiler satisfies (p, ε) -composition security

Table 1: Complexity, amplification order and maximum tolerated leakage probability of the ISW multiplication gadgets. Some leakage probabilities were not computed accurately by VRAPS for performances reasons. An interval on these probabilities is instead given by evaluating lower and upper bound functions $f_{\inf}$ and $f_{\sup}$ of $f(p)$.

# shares	Complexity (N_a, N_c, N_m, N_r)	t	Amplification order	$\log_2$ of maximum tolerated leakage probability
3	(12, 15, 9, 3)	1	1	—
4	(24, 30, 16, 6)	1	1	—
		2	3/2	−9.83
5	(40, 50, 25, 10)	1	1	—
		2	3/2	−11.00
		3	2	−8.05
6	(60, 75, 36, 15)	1	1	—
		2	3/2	−13.00
		3	2	[−9.83, −7.87]
		4	2	[−9.83, −5.92]
7	(84, 105, 49, 21)	1	1	—
		2	3/2	[−16.00, −14.00]
		3	2	[−12.00, −7.87]
		4	5/2	[−12.00, −2.27]
		5	2	[−12.00, −3.12]

property, where p is the tolerated leakage probability and ε is the simulation failure probability. Precisely:

$$\varepsilon = N_g^{c+1} \cdot p^{c+1} \quad (19)$$

Equations (18) and (19) can be directly plugged into our asymptotic analysis of Sec. 5.2, with N_g replacing our $N_{\max}$ and where $c + 1$ stands for our amplification order d . The obtained asymptotic complexity for the AIS compiler is

$$\mathcal{O}(|C| \cdot \kappa^e) \quad \text{with} \quad e = \frac{\log N_g}{\log c + 1} . \quad (20)$$

This is to be compared to $e = \frac{\log N_{\max}}{\log d}$ in our scheme. Moreover, this compiler can tolerate a leakage probability

$$p = \frac{1}{N_g^2} .$$

The authors provide an instantiation of their construction using an existing MPC protocol due to Maurer [22]. From their analysis, this protocol can be implemented with a circuit of $N_g = (4m - c) \cdot \left(\binom{m-1}{c}^2 + 2m \binom{m}{c} \right)$ gates. They instantiate their compiler with this protocol for parameters $m = 5$ parties and $c = 2$ corruptions, from which they get $N_g = 5712$. From this number of gates, they claim to tolerate a leakage probability $p = \frac{1}{5712^2} \approx 2^{-25}$ and our asymptotic analysis gives a complexity of $\mathcal{O}(|C| \cdot \kappa^e)$ with $e \approx 7.87$ according to (20). In Appendix E, we give a detailed analysis of the Maurer protocol [22] in the context of the AIS compiler instantiation. From our analysis, we get the following number of gates for the associated circuit:

$$N_g = (6m - 5) \cdot \left(\binom{m-1}{c}^2 + m(2k - 2) + 2k^2 \right) \quad \text{where} \quad k = \binom{m}{c} .$$

Using the parameters $m = 5$ and $c = 2$ from the AIS compiler instantiation [2], we get $N_g = 8150$. This yields a tolerated leakage probability of $p \approx 2^{-26}$ and an exponent $e = \log 8150 / \log 3 \approx 8.19$ in the asymptotic complexity $\mathcal{O}(|C| \cdot \kappa^e)$ of the AIS compiler.

These results are to be compared to the $p \approx 2^{-8}$ and $e \approx 7.5$ achieved by our construction. In either case ($N_g = 5712$ as claimed in [2] or $N_g = 8150$ according to our analysis), our construction achieves a slightly better complexity while tolerating a much higher leakage probability. We stress that further instantiations of the AIS scheme (based on different MPC protocols) or of our scheme (based on different gadgets) could lead to better asymptotic complexities and/or tolerated leakage probabilities. This is an interesting direction for further research.

8 Implementation Results

In this section, we describe and report the performances of a proof-of-concept implementation of the expanding compiler with our base gadgets as well as a protected AES implementation. The source code of these implementations are publicly available at:

<https://github.com/CryptoExperts/poc-expanding-compiler>

All implementations were run on a laptop computer (Intel(R) Core(TM) i7-8550U CPU, 1.80GHz with 4 cores) using Ubuntu operating system and various C, python and sage libraries.

8.1 Circuit Compiler

First, we developed an implementation in python of a compiler CC, that given three n -share gadgets G_{add} , G_{mult} , G_{copy} and an expansion level k , outputs the compiled gadgets $G_{\text{add}}^{(k)}$, $G_{\text{copy}}^{(k)}$, $G_{\text{mult}}^{(k)}$, each as a C function. The variables' type is given as a command line argument. Table 2 shows the complexity of the compiled gadgets from Section 6 using the compiler with several expansion levels k , as well as their execution time in milliseconds when run in C on randomly generated 8-bit integers. For the generation of random variables, we consider that an efficient external random number generator is available in practice, and so we simply use the values of an incremented counter variable to simulate random gates.

Table 2: Complexity and execution time (in ms, on an Intel i7-8550U CPU) for compiled gadgets $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$ from Section 6 implemented in C.

k	# shares	Gadget	Complexity (N_a, N_c, N_m, N_r)	Execution time
1	3	$G_{\text{add}}^{2(1)}$	(15, 6, 0, 6)	$1, 69.10^{-4}$
		$G_{\text{copy}}^{1(1)}$	(12, 9, 0, 6)	$1, 67.10^{-4}$
		$G_{\text{mult}}^{1(1)}$	(28, 23, 9, 11)	$5, 67.10^{-4}$
2	9	$G_{\text{add}}^{2(2)}$	(297, 144, 0, 144)	$2, 21.10^{-3}$
		$G_{\text{copy}}^{1(2)}$	(288, 153, 0, 144)	$2, 07.10^{-3}$
		$G_{\text{mult}}^{1(2)}$	(948, 582, 81, 438)	$9, 91.10^{-3}$
3	27	$G_{\text{add}}^{2(3)}$	(6183, 3078, 0, 3078)	$9, 29.10^{-2}$
		$G_{\text{copy}}^{1(3)}$	(6156, 3105, 0, 3078)	$9, 84.10^{-2}$
		$G_{\text{mult}}^{1(3)}$	(23472, 12789, 729, 11385)	$3, 67.10^{-1}$

It can be observed that both the complexity and running time grow by almost the same factor with the expansion level, with multiplication gadgets being the slowest as expected. Base gadgets with $k = 1$ roughly take 10^{-4} ms, while these gadgets expanded 2 times ($k = 3$) take between 10^{-2} and 10^{-1} ms. The difference between the linear cost of addition and copy gadgets, and the quadratic cost of multiplication gadgets can also be observed through the gadgets' complexities.

8.2 AES Implementation

We describe hereafter a proof-of-concept AES implementation protected with our instantiation of the expanding compiler. We start by describing the underlying AES circuit (over $\mathbb{K} = \text{GF}(256)$), followed by an analysis of the implementation in C of the complete algorithm.

AES circuit. We first describe the non-linear part of the AES, namely the sbox computation. For the field exponentiation ($x \mapsto x^{254}$ over $\text{GF}(256)$), we use the circuit representation of the processing proposed in [16] and presented in Fig. 7. It corresponds to the *addition chain* (1, 2, 4, 8, 9, 18, 19, 36, 55, 72, 127, 254) and it has been chosen due to its optimality regarding the number of multiplications (11 in total). Each time an intermediate result had to be reused, a copy gate (marked with $\parallel$) has been inserted.

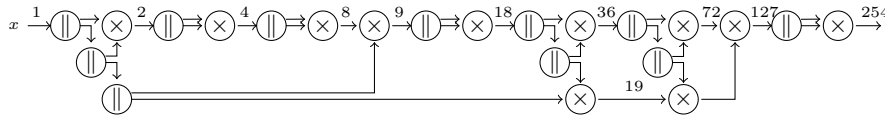


Fig. 7: Circuit for the exponentiation $x \mapsto x^{254}$.

For the second part of the sbox, the affine function is implemented according to the following equation:

$$\text{Affine}(x) = (((((((207x)^2 + 22x)^2 + 1x)^2 + 73x)^2 + 204x)^2 + 168x)^2 + 238x)^2 + 5x + 99$$

with the necessary copy gates. Similarly, the inverse of the affine function is implemented for the sbox inversion as follows:

$$\text{Affine}^{-1}(x) = (((((((147x)^2 + 146x)^2 + 190x)^2 + 41x)^2 + 73x)^2 + 139x)^2 + 79x)^2 + 5x + 5$$

The rest of the operations (MixColumns, ShiftRows, AddRoundKey) are considered as in a standard AES, while adding the necessary copy gates.

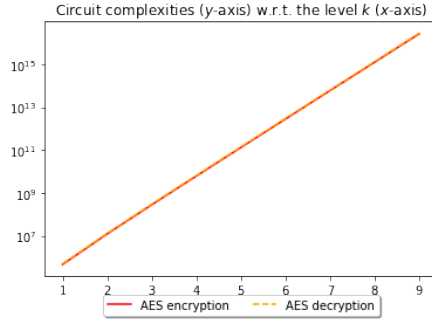
Gate count: Table 3 displays the gate count vectors for AES-128 encryption/decryption procedures as well as for their building blocks. The sbox (resp. sbox inversion) gate count vector was computed as the sum of the gate count vectors of both the exponentiation and affine (resp. affine inversion) functions. We recall that N_a , N_c , N_m , N_r stand for the number of addition gates, copy gates, multiplication gates, and random gates, respectively.

Using the gadgets G_{add}^2 , G_{mult}^1 and G_{copy}^1 proposed in Sec. 6 for the compilation of the AES algorithm, we obtain the instantiation given in Equation (17) of the matrix $\mathbf{M}$ introduced in Sec.

Table 3: AES operations complexity.

AES Operation	Complexity (N_a, N_c, N_m, N_r)
AddRoundKey (for 1 byte)	(1, 0, 0, 0)
SubBytes (for 1 byte)	(8, 25, 26, 0)
MixColumns (for all columns)	(60, 60, 16, 0)
ShiftRows (for all rows)	(0, 0, 0, 0)
<i>AES-128 encryption</i>	<i>(1996, 4540, 4304, 0)</i>
SubBytes Inversion (for 1 byte)	(8, 25, 26, 0)
MixColumns Inversion (for all columns)	(104, 104, 36, 0)
ShiftRows Inversion (for all rows)	(0, 0, 0, 0)
<i>AES-128 decryption</i>	<i>(2392, 4936, 4484, 0)</i>

5.2. Applying the same complexity analysis done previously on the gate count vectors, we display in Fig. 8 the total number of gates in the AES-128 encryption/decryption procedures as functions of the level k . For instance, for the same security level of 2^{-76} exhibited in Sec. 6.5 for the gadgets of Fig. 5, the AES-128 would have to be compiled at a level $k = 9$, and would count around 10^{16} gates.

Fig. 8: Number of gates after compilation of AES-128 encryption/decryption circuits with respect to the level k .

Implementation in C: An n -share AES-128 implementation was developed in C from the above description. Compiled gadgets from Section 8.1 were used for basic operations (addition, multiplication, copy), as generated using our circuit compiler described in Sec. 8.1. We chose the C 8-bit unsigned integer type, and considered operations in $\text{GF}(256)$. For the generation of random values, we assume the availability of an efficient (pseudo)random number generator, and so we simply considered the values of an incremented counter variable to simulate the cost.

Table 4 shows the AES-128 execution time on a 16-byte message with 10 pre-computed sub-keys, using compiled gadgets $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$, with respect to the expansion level k and sharing order $n = 3^k$. It can be seen that the execution time increases with the expansion level with a similar growth as in Table 2. This is because the complexity of the AES circuit strongly depends on the gadgets that are used to replace each gate in the original arithmetic circuit. For example,

with our 3-share gadgets that tolerate a leakage probability of $p \approx 2^{-8}$, a 27-share ($k = 3$) AES-128 takes almost 200 milliseconds to encrypt or decrypt a message.

Table 4: Standard and n -share AES-128 execution time (in ms, on an Intel i7-8550U CPU) using compiled gadgets $G_{\text{add}}^{2(k)}$, $G_{\text{copy}}^{1(k)}$, $G_{\text{mult}}^{1(k)}$.

AES Version	Execution Time (in ms)	
	Encryption	Decryption
Standard (no sharing)	0.06	0.05
3-share ($k = 1$)	1.08	1.07
9-share ($k = 2$)	11.71	10.26
27-share ($k = 3$)	200.29	197.70

Acknowledgments. This work is partly supported by the French FUI-AAP25 VeriSiCC project.

References

1. Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 715–724, San Jose, CA, USA, June 6–8, 2011. ACM Press.
2. Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.
3. Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.
4. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 457–485, Sofia, Bulgaria, April 26–30, 2015. Springer, Heidelberg, Germany.
5. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 116–129, Vienna, Austria, October 24–28, 2016. ACM Press.
6. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.
7. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39, Santa Barbara, CA, USA, August 17–19, 2016. Springer, Heidelberg, Germany.
8. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.

9. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In Jonathan Katz and Hovav Shacham, editors, *Advances in Cryptology – CRYPTO 2017, Part III*, volume 10403 of *Lecture Notes in Computer Science*, pages 397–426, Santa Barbara, CA, USA, August 20–24, 2017. Springer, Heidelberg, Germany.
10. Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.
11. Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Heidelberg, Germany.
12. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO’99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.
13. Gil Cohen, Ivan Bjerre Damgård, Yuval Ishai, Jonas Kölker, Peter Bro Miltersen, Ran Raz, and Ron D. Rothblum. Efficient multiparty protocols via log-depth threshold formulae - (extended abstract). In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013, Part II*, volume 8043 of *Lecture Notes in Computer Science*, pages 185–202, Santa Barbara, CA, USA, August 18–22, 2013. Springer, Heidelberg, Germany.
14. Jean-Sébastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18: 16th International Conference on Applied Cryptography and Network Security*, volume 10892 of *Lecture Notes in Computer Science*, pages 65–82, Leuven, Belgium, July 2–4, 2018. Springer, Heidelberg, Germany.
15. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shihō Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany.
16. Ivan Damgård and Marcel Keller. Secure multiparty AES. In Radu Sion, editor, *FC 2010: 14th International Conference on Financial Cryptography and Data Security*, volume 6052 of *Lecture Notes in Computer Science*, pages 367–374, Tenerife, Canary Islands, Spain, January 25–28, 2010. Springer, Heidelberg, Germany.
17. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.
18. Louis Goubin and Jacques Patarin. DES and differential power analysis (the “duplication” method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES’99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer, Heidelberg, Germany.
19. Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *Journal of Cryptology*, 13(1):31–60, January 2000.
20. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.
21. Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO’99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.
22. Ueli M. Maurer. Secure multi-party computation made simple (invited talk). In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, *SCN 02: 3rd International Conference on Security in Communication Networks*, volume 2576 of *Lecture Notes in Computer Science*, pages 14–28, Amalfi, Italy, September 12–13, 2003. Springer, Heidelberg, Germany.
23. Silvio Micali and Leonid Reyzin. Physically observable cryptography (extended abstract). In Moni Naor, editor, *TCC 2004: 1st Theory of Cryptography Conference*, volume 2951 of *Lecture Notes in Computer Science*, pages 278–296, Cambridge, MA, USA, February 19–21, 2004. Springer, Heidelberg, Germany.
24. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.

25. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427, Santa Barbara, CA, USA, August 17–20, 2010. Springer, Heidelberg, Germany.

A Proof of Proposition 1

Proof. Since G is t -SNI there exist two simulators Sim_1^G and Sim_2^G satisfying (8) and (9) for any $J \subseteq [n]$ and any $\mathcal{W}$ satisfying $|\mathcal{W}| + |J| \leq t$. This is in particular true for every $\mathcal{W}$ and every J both of cardinality lower than or equal to $\frac{t}{2}$. For every set $\mathcal{W}$ such that $|\mathcal{W}| > \frac{t}{2}$ and every set J such that $|J| \leq \frac{t}{2}$, we modify simulator Sim_1^G to return the full set of input indices (*i.e.* $\mathbf{I} = [n]^\ell$). Then, the second simulator Sim_2^G is simply augmented to perfectly simulate $(\text{AssignWires}(G, \mathcal{W}, \hat{x}), \hat{y}|_J)$ from the full knowledge of the gadget inputs (which is trivially possible). By construction, for any J with $|J| \leq \frac{t}{2}$, the output $\mathbf{I} = (I_1, \dots, I_\ell)$ of $\text{Sim}_1^G(\mathcal{W}, J)$ contains at least one I_j with cardinality greater than $\frac{t}{2}$ only when $\mathcal{W}$ has cardinality strictly greater than $\frac{t}{2}$ (and in this case all the I_j 's have full cardinality $[n]$). Hence, the probability $\Pr((|I_1| > \frac{t}{2}) \vee \dots \vee (|I_\ell| > \frac{t}{2}))$ when J is a given set with $|J| \leq \frac{t}{2}$ and $\mathcal{W}$ is the output of $\text{LeakingWires}(G, p)$ satisfies (10), which concludes the proof. $\square$

B Random Probing Expandability for 1-to-2 Gadgets

We provide hereafter the formal definition of the RPE notion for gadgets with 1 input sharing and 2 output sharings.

Definition 11 (Random Probing Expandability). *Let $f = \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \rightarrow \mathbb{K}^n \times \mathbb{K}^n$ is (t, f) -random probing expandable (RPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $\hat{x} \in \mathbb{K}^n$, for every pair of sets $J_1 \subseteq [n]$ and $J_2 \subseteq [n]$, and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I, J'_1, J'_2) &\leftarrow \text{Sim}_1^G(\mathcal{W}, J_1, J_2) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J'_1, J'_2, \hat{x}|_I) \end{aligned}$$

ensures that

1. *the failure event probability satisfies $\Pr(|I| > t) \leq \varepsilon$ with $\varepsilon = f(p)$,*
2. *the set J'_1 is such that $J'_1 = J_1$ if $|J_1| \leq t$ and $J'_1 \subseteq [n]$ with $|J'_1| = n - 1$ otherwise,*
3. *the set J'_2 is such that $J'_2 = J_2$ if $|J_2| \leq t$ and $J'_2 \subseteq [n]$ with $|J'_2| = n - 1$ otherwise,*
4. *the output distribution satisfies*

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, \hat{x}), \hat{y}|_{J'_1}, \hat{z}|_{J'_2}) \tag{21}$$

where $(\hat{y}, \hat{z}) = G(\hat{x})$.

C Expandability

C.1 Illustration of the subsets S_k

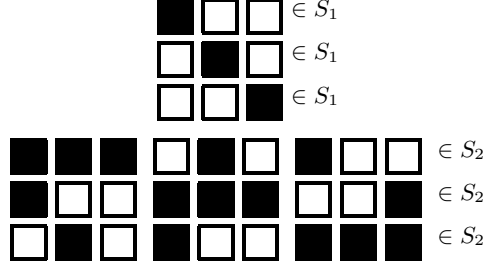


Fig. 9: Illustration of all elements of S_1 and some elements of S_2 , for $n = 3$ and $t = 1$.

C.2 Proof of Theorem 2

The proof of the theorem relies on what we shall call the *assignment expansion* property. Throughout the proof we shall denote $\varepsilon_k = f^{(k)}(p)$. We call *level- k gadget* a gadget that has been expanded $k - 1$ times $G^{(k)} = \text{CC}^{(k-1)}(G)$ where G is a base gadget (or a level-1 gadget) among G_{add} , G_{mult} , G_{copy} .

We proceed by induction to show that the level- k gadgets are $(S_k, f^{(k)})$ -RPE. The base case is one of the theorem hypotheses, namely the base gadgets G_{add} , G_{mult} and G_{copy} (*i.e.* the level-1 gadgets) are (t, f) -RPE, which is equivalent to (S_1, f) -RPE. We must then show the induction step: assuming that the level- k gadgets are $(S_k, f^{(k)})$ -RPE, show that the level- $(k + 1)$ gadgets are $(S_{k+1}, f^{(k+1)})$ -RPE. For the sake of simplicity, we depict our proof by assuming that all the gadgets are 2-to-1 gadget (which is actually not the case for copy gadgets). The proof mechanism for the general case (with 2-to-1 and 1-to-2 gadgets) is strictly similar but heavier on the form.

In order to show that $G^{(k+1)}$ is $(S_{k+1}, f^{(k+1)})$ -RPE we must construct two simulators $\text{Sim}_1^{G^{(k+1)}}$ and $\text{Sim}_2^{G^{(k+1)}}$ that satisfy the conditions of Definition 10 for the set of subsets S_{k+1} . More precisely, we must construct two simulators $\text{Sim}_1^{G^{(k+1)}}$ and $\text{Sim}_2^{G^{(k+1)}}$ such that for every $(\hat{x}^*, \hat{y}^*) \in \mathbb{K}^{n^{k+1}} \times \mathbb{K}^{n^{k+1}}$, and for every set $J^* \in S_{k+1} \cup [n^{k+1}]$, the random experiment

$$\begin{aligned} \mathcal{W}^* &\leftarrow \text{LeakingWires}(G^{(k+1)}, p) \\ (I_1^*, I_2^*, J^{*'}) &\leftarrow \text{Sim}_1^G(\mathcal{W}^*, J^*) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}^*, J^*, \hat{x}^*|_{I_1^*}, \hat{y}^*|_{I_2^*}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1^* \equiv (I_1^* \notin S_{k+1})$ and $\mathcal{F}_2^* \equiv (I_2^* \notin S_{k+1})$ verify

$$\Pr(\mathcal{F}_1^*) = \Pr(\mathcal{F}_2^*) = \varepsilon_{k+1} \quad \text{and} \quad \Pr(\mathcal{F}_1^* \wedge \mathcal{F}_2^*) = \varepsilon_{k+1}^2 \quad (22)$$

2. the set $J^{*'}$ is such that $J^{*'} = J^*$ if $J^* \in S_{k+1}$ and $J^{*'} = [n^{k+1}] \setminus \{j^*\}$ otherwise,

3. the output distribution satisfies

$$out \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, (\hat{x}, \hat{y})), \hat{z}|_{J^*}) \quad (23)$$

where $\hat{z} = G^{(k+1)}(\hat{x}, \hat{y})$.

We distinguish two cases: either $J^* \in S_{k+1}$ (normal case), or $J^* = [n^{k+1}]$ (saturated case).

Normal case: $J^* \in S_{k+1}$. By definition of the expanding compiler, we have that a level- $(k+1)$ gadget $G^{(k+1)}$ is obtained by replacing each gate of the base gadget by the corresponding level- k gadget and by replacing each wire of the base gadget by n^k wires carrying a (n^k) -linear sharing of the original wire. In particular $G^{(k+1)}$ has n^{k+1} output wires which can be split in n groups of n^k wires, each group being the output of a different $G^{(k)}$ gadget. We split the set J^* accordingly so that $J^* = J_1^* \cup \dots \cup J_n^*$, where each set J_i^* pertains to the i th group of output wires. By definition of S_k , since $J^* \in S_{k+1}$, we must have $J_i^* \in S_k$ for all $1 \leq i \leq n$, except at most t of them for which $J_i^* = [n^k]$. We define J_{base} as the set of indexes i such that $J_i^* \notin S_k$. Therefore we must have $|J_{\text{base}}| \leq t$.

We first describe the simulator $\text{Sim}_1^{G^{(k+1)}}$ that takes the leaking wires $\mathcal{W}^*$ and the output wires $J^* \in S_{k+1}$ to be simulated and produce the sets $I_1^* \subseteq [n^{k+1}]$ and $I_2^* \subseteq [n^{k+1}]$ of required inputs. The simulator $\text{Sim}_1^{G^{(k+1)}}$ starts by defining a set $\mathcal{W}_{\text{base}}$ which is initialized to $\emptyset$; this will correspond to the set of leaking wires for the base gadget. Then the simulation goes through all the level- k gadgets composing $G^{(k+1)}$ from bottom to top *i.e.* starting with the level- k gadgets producing the output sharing up to the level- k gadgets processing the input sharings. Let us denote by $\{G_j^{(k)}\}_j$ these level- k gadgets. For each $G_j^{(k)}$, one runs the simulator Sim_1 from the $(S_k, f^{(k)})$ -RPE property on input $\mathcal{W}_j$ and J_j defined as follows. The set of leaking wires $\mathcal{W}_j$ is defined as the subset of $\mathcal{W}^*$ corresponding to the wires of $G_j^{(k)}$. For the gadgets $G_j^{(k)}$ on the bottom layer, the set J_j is set to one of the J_i^* (with indices scaled to range in $[n^k]$). For all the other gadgets $G_j^{(k)}$ (which are not on the bottom layer), the set J is defined as the set I_1 or I_2 output from Sim_1 for the child gadget $G^{(k)}_{j'}$ (for which Sim_1 has already been run).

Whenever a failure event occurs for a $G_j^{(k)}$ gadget, namely when the set I (either I_1 or I_2) output from Sim_1 is such that $I \notin S_k$, we add the index of the wire corresponding to this input in the base gadget G to the set $\mathcal{W}_{\text{base}}$. Once the Sim_1 simulations have been run for all the $G_j^{(k)}$ gadgets, ending with the top layers, we get the final sets I corresponding to the input shares. Each of these sets corresponds to an n^k -sharing as input of a $G_j^{(k)}$ gadget, which corresponds to a wire as input of the base gadget among the $2 \cdot n$ wires carrying the two input n -sharings of the base gadget. We denote by $I_{1,1}^*, \dots, I_{1,n}^*$ and $I_{2,1}^*, \dots, I_{2,n}^*$ the corresponding sets so that defining

$$I_1^* = I_{1,1}^* \cup \dots \cup I_{1,n}^* \quad \text{and} \quad I_2^* = I_{2,1}^* \cup \dots \cup I_{2,n}^* , \quad (24)$$

the tuple $\hat{x}^*|_{I_1^*}$ and $\hat{y}^*|_{I_2^*}$ contains the shares designated by the final I sets.

At the end of the $\text{Sim}_1^{G^{(k+1)}}$ simulation, the set $\mathcal{W}_{\text{base}}$ contains all the labels of wires in the base gadget G for which a failure event has occurred in the simulation of the corresponding $G_j^{(k)}$ gadget. Thanks to the $(S_k, f^{(k)})$ -RPE property of these gadgets, the failure events happen (mutually independently) with probability ε_k which implies

$$\mathcal{W}_{\text{base}} \stackrel{\text{id}}{=} \text{LeakingWires}(G, \varepsilon_k) \quad (25)$$

Recall that $|J_{\text{base}}| \leq t$. We can then run Sim_1^G to obtain:

$$(I_{1,\text{base}}, I_{2,\text{base}}) = \text{Sim}_1^G(\mathcal{W}_{\text{base}}, J_{\text{base}}) . \quad (26)$$

For all $1 \leq i \leq n$, if $i \in I_{1,\text{base}}$, we force $I_{1,i}^* \leftarrow [n^k]$, so that the corresponding i -th input wire of the base gadget can be computed from the corresponding input wires in $I_{1,i}^*$. The simulator $\text{Sim}_1^{G^{(k+1)}}$ then returns (I_1^*, I_2^*) as output.

The (t, f) -RPE property of the base gadget G implies that the *base failure events* $|I_{1,\text{base}}| = n$ and $|I_{2,\text{base}}| = n$ are ε_{k+1} -mutually unlikely, where $\varepsilon_{k+1} = f(\varepsilon_k)$. We argue that for all $1 \leq i \leq n$, $I_{1,i}^* \notin S_k \iff i \in I_{1,\text{base}}$. Namely if a failure event has occurred for a set $I_{1,i}^*$ (i.e. $I_{1,i}^* \notin S_k$) then we must have $i \in I_{1,\text{base}}$. Indeed, if a failure event has occurred for a set $I_{1,i}^*$ then the label of the i th input wire (for the first sharing) of the base gadget G has been added to $\mathcal{W}_{\text{base}}$ and Sim_1^G has no choice but to include this index to the set $I_{1,\text{base}}$ so that Sim_2^G can achieve a perfect simulation of the wire assignment (as required by the RPE property of G). Moreover if $i \in I_{1,\text{base}}$ then by construction we have set $I_{1,i}^* = [n^k]$ and therefore $I_{1,i}^* \notin S_k$. This implies that if $|I_{1,\text{base}}| \leq t$ then $I_1^* \in S_{k+1}$ (and the same happens for I_2^* w.r.t. $I_{2,\text{base}}$). We deduce that the failure events $\mathcal{F}_1^*$ and $\mathcal{F}_2^*$ are also ε_{k+1} -mutually unlikely, as required by the $(S_{k+1}, f^{(k+1)})$ -RPE property of $G^{(k+1)}$.

We now describe the simulator $\text{Sim}_2^{G^{(k+1)}}$ that takes as input $\hat{x}^*|_{I_1^*}$ and $\hat{y}^*|_{I_2^*}$ and produces a perfect simulation of $(\text{AssignWires}(G^{(k+1)}, \mathcal{W}^*, (\hat{x}^*, \hat{y}^*)), \hat{z}|_{J^*})$ where $\hat{z} = G^{(k+1)}(\hat{x}, \hat{y})$. Let $\hat{x}^b$ and $\hat{y}^b$ denote the n -linear sharings obtained by applying the linear decoding to each group of n^k shares in $\hat{x}^*$ and $\hat{y}^*$, so that the elements of $\hat{x}^b$ and $\hat{y}^b$ correspond to the input wires in the base gadget G . The assignment expansion property implies that a perfect assignment of the wires of $G^{(k+1)}$ on input $\hat{x}^*$ and $\hat{y}^*$ can be derived from an assignment of the wires of the base gadget G on input $\hat{x}^b$ and $\hat{y}^b$. The simulator makes use of this property by first running

$$\text{out}_{\text{base}} \leftarrow \text{Sim}_2^G(\mathcal{W}_{\text{base}}, J_{\text{base}}, \hat{x}^b|_{I_{1,\text{base}}}, \hat{y}^b|_{I_{2,\text{base}}}) , \quad (27)$$

Note that the input values $\hat{x}^b|_{I_{1,\text{base}}}$ and $\hat{y}^b|_{I_{2,\text{base}}}$ can be obtained from the corresponding shares in I_1^* and I_2^* . Thanks to the (t, f) -RPE property of G and by construction of $I_{1,\text{base}}$ and $I_{2,\text{base}}$, this outputs a distribution satisfying

$$\text{out}_{\text{base}} \stackrel{\text{id}}{=} \left(\text{AssignWires}(G, \mathcal{W}_{\text{base}}, (\hat{x}^b, \hat{y}^b)), \hat{z}^b|_{J_{\text{base}}} \right) \quad (28)$$

The simulator then goes through all the $G_j^{(k)}$ gadgets from input to output and for each of them runs the simulator Sim_2 of the RPE property on inputs $\mathcal{W}_j$, J_j , $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$ where $\mathcal{W}_j$ and J_j are the sets from the first phase of the simulation for the gadget $G_j^{(k)}$, I_1 and I_2 are the corresponding sets produced by the Sim_1 simulator for $G_j^{(k)}$, and $\hat{x}$ and $\hat{y}$ are the inputs of $G_j^{(k)}$ in the evaluation of $G^{(k+1)}(\hat{x}^*, \hat{y}^*)$. Provided that the partial inputs $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$ are perfectly simulated, this call to Sim_2 produces a perfect simulation of $(\text{AssignWires}(G_j^{(k)}, \mathcal{W}_j, (\hat{x}, \hat{y}), \hat{z}|_{J_j}))$ where $\hat{z} = G_j^{(k)}(\hat{x}, \hat{y})$. In order to get perfect simulations of the partial inputs $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$, the simulator proceeds as follows. For the top layer of $G^{(k)}$ gadgets (the ones processing the input shares) the shares $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$ can directly be taken from the inputs $\hat{x}^*|_{I_1^*}$ and $\hat{y}^*|_{I_2^*}$. For the next gadgets the shares $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$ match the shares $\hat{z}|_J$ output from the call to Sim_2 for a parent gadget. The only exception occurs in case of a failure event.

In that case the simulation needs the full input $\hat{x} = (x_1, \dots, x_{n^k})$ (and/or $\hat{y} = (y_1, \dots, y_{n^k})$), while we have set $|I_1| = n^k - 1$ (and/or $|I_2| = n^k - 1$) to satisfy the RPE requirements of the parent gadget in the first simulation phase. Nevertheless, for such cases a perfect simulation of the plain value $x = \text{LinDec}(\hat{x})$ (and/or $y = \text{LinDec}(\hat{y})$) is included to out_{base} by construction of $\mathcal{W}_{\text{base}}$. We can therefore perfectly simulate the missing share from the $n^k - 1$ other shares and the plain value x (or y). We thus get a perfect simulation of $(\text{AssignWires}(G_j^{(k)}, \mathcal{W}_j, (\hat{x}, \hat{y}), \hat{z}|_{J_j}))$ for all the level- k gadgets $G_j^{(k)}$ which gives us a perfect simulation of $(\text{AssignWires}(G^{(k+1)}, \mathcal{W}^*, (\hat{x}^*, \hat{y}^*)), \hat{z}|_{J^*})$.

Saturated case: $J^* = [n^{k+1}]$. The saturated case proceeds similarly. The difference is that we must simulate all n^{k+1} output shares of the level- $(k+1)$ gadget, except for one share index j^* that can be chosen by the simulator.

The simulator $\text{Sim}_1^{G^{(k+1)}}$ is defined as previously. Since $J^* = [n^{k+1}]$, we must define $J_{\text{base}} = [1, n]$. Moreover we have $J_i^* = [n^k]$ for all $1 \leq i \leq n$. This implies that for the gadgets $G_j^{(k)}$ on the output layer, the sets J_j are all equal to $[n^k]$ as well. The set $\mathcal{W}_{\text{base}}$ is defined as previously, and the simulator $\text{Sim}_1^{G^{(k+1)}}$ returns (I_1^*, I_2^*) as previously. The failure events $\mathcal{F}_1^*$ and $\mathcal{F}_2^*$ are still ε_{k+1} -mutually unlikely, as required by the $(S_{k+1}, f^{(k+1)})$ -RPE property of $G^{(k+1)}$.

The simulator $\text{Sim}_2^{G^{(k+1)}}$ is defined as previously. In particular, from the running of the base gadget simulator Sim_2^G , we obtain a perfect simulation of the output wires $\hat{z}^b|_{J'_{\text{base}}}$ for some J'_{base} with $|J'_{\text{base}}| = n - 1$. Combined with the perfect simulation of the output wires corresponding to the output sets J'_j from the gadgets $G_j^{(k)}$ on the output layer, with $|J'_j| = n^k - 1$, we obtain a subset J' of output wires for our level- $(k+1)$ gadget with $|J'| = n^{k+1} - 1$ as required. Eventually this gives us a perfect simulation of $(\text{AssignWires}(G^{(k+1)}, \mathcal{W}^*, (\hat{x}^*, \hat{y}^*)), \hat{z}|_{J'})$. This terminates the proof of Theorem 2.

C.3 Proposition 3

We give here the proof of Proposition 3.

Proof. Let Sim_1^G be the simulator from the (t, f) -wRPE property. This simulator outputs I_1 and I_2 such that

$$\Pr(\mathcal{F}_1) = \varepsilon_1 \leq \varepsilon, \quad \Pr(\mathcal{F}_2) = \varepsilon_2 \leq \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon_{12} \leq \varepsilon^2, \quad (29)$$

where $\mathcal{F}_1 \equiv (|I_1| > t)$ and $\mathcal{F}_2 \equiv (|I_2| > t)$. We show how to construct $\text{Sim}_1^{G'}$ which outputs I'_1 and I'_2 such that

$$\Pr(\mathcal{F}'_1) = \Pr(\mathcal{F}'_2) = \varepsilon' \quad \text{and} \quad \Pr(\mathcal{F}'_1 \wedge \mathcal{F}'_2) = (\varepsilon')^2 \quad \text{with} \quad \varepsilon' = \varepsilon + \frac{3}{2}\varepsilon^2 \quad (30)$$

where $\mathcal{F}'_1 \equiv (|I'_1| > t)$ and $\mathcal{F}'_2 \equiv (|I'_2| > t)$ and such that $I_1 \subseteq I'_1$ and $I_2 \subseteq I'_2$. In particular, the latter implies that we can keep the same Sim_2^G simulator since it is always given the same input shares plus additional input shares to achieve the same simulation as before.

The simulator $\text{Sim}_1^{G'}$ first calls the simulator Sim_1^G to get I_1 and I_2 . Whenever $|I_1|$ and $|I_2|$ are both lower than t , *i.e.* no failure event occurs, which happens with probability $p_{\text{succ}} = 1 - (\varepsilon_1 +$

$\varepsilon_2 - \varepsilon_{12}$), $\text{Sim}_1^{G'}$ outputs

$$(I'_1, I'_2) = \begin{cases} ([n], I_2) & \text{with probability } p_1 = \delta_1/p_{\text{succ}} \\ (I_1, [n]) & \text{with probability } p_2 = \delta_2/p_{\text{succ}} \\ ([n], [n]) & \text{with probability } p_{12} = \delta_{12}/p_{\text{succ}} \\ (I_1, I_2) & \text{with probability } 1 - (p_1 + p_2 + p_{12}) \end{cases}$$

for some $\delta_1, \delta_2, \delta_{12} \geq 0$ such that $\delta_1 + \delta_2 + \delta_{12} \leq p_{\text{succ}}$. We hence get

$$\begin{aligned} \Pr(\mathcal{F}'_1) &= \varepsilon_1 + \delta_1 + \delta_{12} \\ \Pr(\mathcal{F}'_2) &= \varepsilon_2 + \delta_2 + \delta_{12} \\ \Pr(\mathcal{F}'_1 \wedge \mathcal{F}'_2) &= \varepsilon_{12} + \delta_{12} \end{aligned}$$

We must now fix $\delta_1, \delta_2, \delta_{12} \geq 0$ to satisfy (30), with $\varepsilon' := \varepsilon + 3\varepsilon^2/2$ and $\delta_1 + \delta_2 + \delta_{12} \leq p_{\text{succ}} = 1 - (\varepsilon_1 + \varepsilon_2 - \varepsilon_{12})$. We fix $\delta_{12} = (\varepsilon')^2 - \varepsilon_{12}$; this gives $\Pr(\mathcal{F}'_1 \wedge \mathcal{F}'_2) = (\varepsilon')^2$, and from (29) we obtain $\delta_{12} \geq 0$ as required. We let:

$$\delta_1 := \varepsilon' - \varepsilon_1 - \delta_{12}$$

which gives $\Pr(\mathcal{F}'_1) = \varepsilon'$ as required. Moreover we obtain using (29):

$$\begin{aligned} \delta_1 &= \varepsilon + \frac{3}{2}\varepsilon^2 - \varepsilon_1 - \left((\varepsilon + \frac{3}{2}\varepsilon^2)^2 - \varepsilon_{12} \right) \geq \frac{3}{2}\varepsilon^2 - \left(\varepsilon^2 + 3\varepsilon^3 + \frac{9}{4}\varepsilon^4 \right) \\ &\geq \varepsilon^2 \cdot \left(\frac{1}{2} - 3\varepsilon - \frac{9}{4}\varepsilon^2 \right) \geq 0 \quad \text{for } \varepsilon \leq 0.14. \end{aligned}$$

We obtain similar conditions for $\delta_2 := \varepsilon' - \varepsilon_2 - \delta_{12}$. Finally, we have

$$\begin{aligned} \delta_1 + \delta_2 + \delta_{12} &= \varepsilon' - \varepsilon_1 - \delta_{12} + \varepsilon' - \varepsilon_2 - \delta_{12} + \delta_{12} \\ &= 2\varepsilon' - \varepsilon_1 - \varepsilon_2 - (\varepsilon')^2 + \varepsilon_{12} = p_{\text{succ}} + 2\varepsilon' - (\varepsilon')^2 - 1 \\ &\leq p_{\text{succ}} + 2\varepsilon' - 1 \leq p_{\text{succ}} \quad \text{for } \varepsilon < 0.14. \end{aligned}$$

as required. □

D Instantiation

D.1 Verification functions

In this section, we give the whole set of coefficients obtained for gadgets in Section 6. When sets of coefficients are completed with $\dots$, then a bound of the subsequent function can be obtained from the binomial coefficients as explained in Section 3. In such cases, the number of coefficients (as the number of wires in the circuit) is given in a last column.

Verification timings are also given in the tables by running the tool a laptop computer (Intel(R) Core(TM) i7-8550U CPU, 1.80GHz with 4 cores) using Ubuntu operating system.

D.2 Addition Gadgets

Hereafter are the coefficients c_i as defined in Section 3 for addition gadgets G_{add}^1 and G_{add}^2 defined in Section 6.

gadget	function	coefficient computed by our automatic tool ($\beta = 5$)	# wires	Verification Time (in s)
G_{add}^1	$f_1^{(1)}$	$\{0, 3, 150, 3649, 53830, \dots\}$	36	148
	$f_2^{(1)}$	$\{0, 3, 116, 2429, 34469, \dots\}$		
	$f_3^{(1)}$	$\{0, 0, 10, 495, 10959, \dots\}$		
	$f_1^{(2)}$	$\{0, 3, 144, 3342, 45611, \dots\}$		
	$f_2^{(2)}$	$\{0, 3, 110, 2208, 27580, \dots\}$		
	$f_3^{(2)}$	$\{0, 0, 4, 228, 4933, \dots\}$		
G_{add}^2	$f_1^{(1)}$	$\{0, 3, 118, 2457, 34998, \dots\}$	36	176
	$f_2^{(1)}$	$\{0, 3, 106, 2035, 27812, \dots\}$		
	$f_3^{(1)}$	$\{0, 0, 0, 69, 3034, \dots\}$		
	$f_1^{(2)}$	$\{0, 3, 118, 2403, 29859, \dots\}$		
	$f_2^{(2)}$	$\{0, 3, 106, 2007, 22079, \dots\}$		
	$f_3^{(2)}$	$\{0, 0, 0, 9, 600, \dots\}$		

D.3 Multiplication Gadgets

Hereafter are the coefficients c_i as defined in Section 3 for multiplication gadget G_{mult} defined in Section 6.

gadget	function	coefficient computed by our automatic tool ($\beta = 5$)	# wires	verification time (in s)
G_{mult}	$f_1^{(1)}$	$\{0, 3, 1232, 60940, 1653719, \dots\}$	97	5228
	$f_2^{(1)}$	$\{0, 7, 1688, 74662, 2152987, \dots\}$		
	$f_3^{(1)}$	$\{0, 0, 62, 5300, 291603, \dots\}$		
	$f_1^{(2)}$	$\{0, 3, 1254, 42135, 1428624, \dots\}$		
	$f_2^{(2)}$	$\{0, 11, 2135, 47322, 1437774, \dots\}$		
	$f_3^{(2)}$	$\{0, 0, 83, 4248, 255461, \dots\}$		

D.4 Copy Gadgets

Hereafter are the coefficients c_i as defined in Section 3 for the copy gadget G_{copy} defined in Section 6.

gadget	function	coefficient computed by our automatic tool ($\beta = s = 33$)	# wires	verification time (in s)
G_{copy}	$f_{1,1}$	$\{0, 33, 1137, 16812, 145288, 852472, 3750849, 13073855, 37574146, 91573962, 192726070, 354263297, 572852089, 818662608, 1037103082, 1166786707, 1166799413, 1037157725, 818809139, 573166437, 354817320, 193536720, 92561040, 38567100, 13884156, 4272048, 1107568, 237336, 40920, 5456, 528, 33, 1\}$	33	49
	$f_{1,2}$	$\{0, 30, 1285, 19887, 166695, 951201, 4021599, 13567630, 38231896, 92255103, 193295461, 354654683, 573074084, 818765733, 1037141693, 1166798076, 1166801950, 1037158129, 818809180, 573166439, 354817320, 193536720, 92561040, 38567100, 13884156, 4272048, 1107568, 237336, 40920, 5456, 528, 33, 1\}$		
	$f_{2,1}$	same coefficients than $f_{1,2}$		
	$f_{2,2}$	$\{0, 27, 1433, 23538, 188460, 1016149, 4150387, 13760724, 38465921, 92491608, 193496624, 354798258, 573159259, 818807160, 1037157912, 1166803059, 1166803107, 1037158320, 818809200, 573166440, 354817320, 193536720, 92561040, 38567100, 13884156, 4272048, 1107568, 237336, 40920, 5456, 528, 33, 1\}$		

E Complexity of the MPC Protocol in the AIS Compiler

In the following we compute the complexity and the value of N_g in the instantiation of the AIS compiler [2]. First, using this compiler, given a circuit C to compile, each gate G is implemented using a functionality F associated to the MPC protocol. Such a functionality F receives m shares of each input and then reconstructs them to obtain original values. This reconstruction can be done with $2(m - 1)$ addition gates. Then after computing the gate G , m additive shares of the output are computed twice. This step consists of one gate for G , and $2(m - 1)$ gates for the additive sharing along with $2(m - 1)$ random gates.⁸ So each gate G to compile is replaced by $6m - 5$ gates, each computed jointly by the m parties in the MPC protocol. Next, we state the complexity of the protocol from [22]. Each gate in a functionality F is jointly computed by all m parties. In the beginning, each party holds one share of each input.

The first step consists in a k -secret sharing of each input share where $k = \binom{m}{c}$. For an input of m shares, each party will hold a total of $m \binom{m-1}{c}$ shares. For two inputs, this step has a complexity of $m(2k - 2)$.

The second step is either performing an addition or a multiplication, depending on the gate G associated to the functionality. An addition simply means each party locally adding all its shares, holding a complexity of $m \binom{m-1}{c}$. In case of a multiplication gate, each party will locally compute the sum of the product of the shares of both inputs, and then share its local result using a secret sharing scheme as in the first step. This procedure holds a complexity of $\binom{m-1}{c}^2$ for computing the result, $m(2k - 2)$ for the secret sharing, and $2k^2$ copy gates. Clearly, the cost of the second step is more important for the multiplication and can be upper bounded by⁹

$$\binom{m-1}{c}^2 + m \cdot (2k - 2) + 2k^2.$$

In the final step, every party broadcasts its shares to all other parties, and then adds all the shares it receives. The complexity of this step is $\binom{m}{c}$.

Considering the cost of replacing each gate G in the circuit to compile by $6m - 5$ gates, and the cost to compute each of these gates using the protocol Π , the total number of gates N_g is upper bounded by

$$(6m - 5) \cdot \left(\binom{m-1}{c}^2 + m(2k - 2) + 2k^2 \right).$$

⁸ In [2], the authors only consider $2(m - 1)$ for the cost of this step, not counting the number of random gates necessary to compute the additive sharing of the output.

⁹ The authors claim in their paper a complexity of $\binom{m-1}{c}^2 + 2mk$, since they do not take into account the copy gates needed to compute the product of input shares.

Appendix B

On the Power of Expansion: More Efficient Constructions in the Random Probing Model

Hereafter is appended the full version of our paper [\[BRT21\]](#), joint work with Matthieu Rivain and Abdul Rahman Taleb, published at **EUROCRYPT 2021**.

On the Power of Expansion: More Efficient Constructions in the Random Probing Model

Sonia Belaïd¹, Matthieu Rivain¹, and Abdul Rahman Taleb^{1,2}

¹ CryptoExperts, France

² Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

{sonia.belaid,matthieu.rivain,abdul.taleb}@cryptoexperts.com

Abstract. The random probing model is a leakage model in which each wire of a circuit leaks with a given probability p . This model enjoys practical relevance thanks to a reduction to the noisy leakage model, which is admitted as the right formalization for power and electromagnetic side-channel attacks. In addition, the random probing model is much more convenient than the noisy leakage model to prove the security of masking schemes. In a recent work, Ananth, Ishai, and Sahai (CRYPTO 2018) introduce a nice expansion strategy to construct random probing secure circuits. Their construction tolerates a leakage probability of 2^{-26} , which is the first quantified achievable leakage probability in the random probing model. In a follow-up work, Belaïd, Coron, Prouff, Rivain, and Taleb (CRYPTO 2020) generalize their idea and put forward a complete and practical framework to generate random probing secure circuits. The so-called expanding compiler can bootstrap simple base gadgets as long as they satisfy a new security notion called *random probing expandability* (RPE). They further provide an instantiation of the framework which tolerates a 2^{-8} leakage probability in complexity $\mathcal{O}(\kappa^{7.5})$ where κ denotes the security parameter.

In this paper, we provide an in-depth analysis of the RPE security notion. We exhibit the first upper bounds for the main parameter of a RPE gadget, which is known as the *amplification order*. We further show that the RPE notion can be made tighter and we exhibit strong connections between RPE and the *strong non-interference* (SNI) composition notion. We then introduce the first generic constructions of gadgets achieving RPE for any number of shares and with nearly optimal amplification orders and provide an asymptotic analysis of such constructions. Last but not least, we introduce new concrete constructions of small gadgets achieving maximal amplification orders. This allows us to obtain much more efficient instantiations of the expanding compiler: we obtain a complexity of $\mathcal{O}(\kappa^{3.9})$ for a slightly better leakage probability, as well as $\mathcal{O}(\kappa^{3.2})$ for a slightly lower leakage probability.

Keywords: Random probing model, masking, side-channel security

1 Introduction

Most commonly used cryptographic algorithms are assumed to be secure against *black-box* attacks, when the adversary is limited to the knowledge of some inputs and outputs. However, as revealed in the late nineties [18], their implementation on physical devices can be vulnerable to the more powerful *side-channel attacks*. The latter additionally exploit the physical emanations of the underlying device such as the execution time or the device temperature, power consumption, or electromagnetic radiations during the algorithm execution.

To counteract side-channel attacks which often only require cheap equipment and can be easily mounted in a short time interval, the cryptographic community has searched for efficient countermeasures. Among the different approaches, one of the most widely used is known as *masking*. Simultaneously introduced by Chari, Jutla, Rao and Rohatgi [10], and by Goubin and Patarin [16] in 1999, it happens to be strongly related to techniques usually applied in secure multi-party computation. In a nutshell, the idea is to split each sensitive variable of the implementation into n

shares such that $n - 1$ of them are generated uniformly at random whereas the last one is computed as a combination of the original value and the random shares. Doing so, one aims to ensure that an adversary cannot recover the secret without knowledge of all the shares. When the shares are combined by bitwise addition, the masking is said to be *Boolean*, and it enjoys simple implementation for linear operations which can be simply applied on each share separately. However, things are trickier for non-linear operations for which it is impossible to compute the result without combining shares.

In order to reason about the security of these countermeasures, the community has introduced a variety of models. Among them, the *probing model* introduced by Ishai, Sahai, and Wagner in 2003 [17] is well suited to analyze the security of masked implementations. Basically, it assumes that an adversary is able to get the exact values of a certain number t of intermediate variables in an implementation. This way, it captures the increasing difficulty of combining noisy leakage to recover secrets. Despite its wide use by the community [20, 13, 11, 8, 12], the probing model raised a number of concerns regarding its relevance in practice. Therefore, in 2013, Prouff and Rivain introduced a general and practical model, known as the *noisy leakage model* [19]. This model well captures the reality of embedded devices by assuming that all the manipulated data leak together with some noise. Unfortunately, proving the security of a masking scheme in this model is rather tedious, which is why Duc, Dziembowski, and Faust provided in 2014 a reduction showing that a scheme secure in the probing model is also secure in the noisy leakage model [14].

This reduction is based on an intermediate leakage model, known as *random probing model*, to which the security in the noisy leakage model tightly reduces. In this model, every wire of a circuit is assumed to leak with some constant leakage probability. Then, a circuit is secure if there is a negligible probability that these leaking wires actually reveal information on the secrets. Compared to the probing model, the random probing model is closer to the noisy leakage model and, in particular, captures *horizontal attacks* which exploit the repeated manipulations of variables throughout the implementation. Classical probing secure schemes are also secure in the random probing model but the tolerated leakage probability (a.k.a. leakage rate) might not be constant which is not satisfactory from a practical viewpoint. Indeed, in practice, the leakage probability translates to some side-channel noise amount which might not be customizable by the implementer.

So far, only a few constructions [1, 3, 2, 9] tolerate a constant leakage probability. The two former ones [1, 3] are based on expander graphs and the tolerated probability is not made explicit. The third construction [2] is based on multi-party computation protocols and an expansion strategy. It reaches a tolerated leakage probability of around 2^{-26} for a complexity of $\mathcal{O}(\kappa^{8.2})$ for some security parameter κ , as computed by the authors of [9]. Finally, the more recent construction [9] relies on masking gadgets and a similar expansion strategy and reaches a tolerated leakage probability of 2^{-8} for a complexity of $\mathcal{O}(\kappa^{7.5})$. While obtaining such quantified tolerated leakage probability is of great practical interest, the obtained complexity is high which makes this construction hardly practical.

Besides their explicit construction, the authors of [9] provide a complete and practical framework to generate random probing secure implementations. Namely, they formalize the *expanding compiler* which produces a random probing secure version of any circuit from three base gadgets (for addition, copy, and multiplication) achieving a *random probing expandability* (RPE) property. The advantage of this approach is that it enables to bootstrap small gadgets (defined for a small number of shares) into a circuit achieving arbitrary security in the random probing model while tolerating a constant and quantified leakage probability. Although the concrete results of [9] in terms of complexity and

tolerated leakage probability are promising, the authors left open the analysis of this RPE property and the design of better gadgets in this paradigm.

Our contributions. In this paper, we provide an in-depth analysis of the random probing expandability security notion. We first provide some upper bounds for the *amplification order* of an RPE gadget, which is the crucial parameter in view of a low-complexity instantiation of the expanding compiler. We further show that the RPE notion can be made tighter and we exhibit strong relations between RPE and the *strong non-interference* (SNI) composition notion for probing-secure gadgets.

From these results, we introduce the first generic constructions of gadgets achieving RPE for any number of shares and with nearly optimal amplification orders. These generic gadgets are derived from the widely known Ishai-Sahai-Wagner (ISW) construction. We show that the obtained expanding compiler can approach a quadratic complexity depending on the leakage probability that must be tolerated: the smaller the leakage probability, the closer the complexity to $\mathcal{O}(\kappa^2)$. We further introduce a new multiplication gadget achieving the optimal amplification order, which allows us to improve the convergence to a quadratic complexity.

Finally, we provide new concrete constructions of copy, addition, and multiplication gadgets achieving maximal amplification orders for small numbers of shares. These gadgets yield much more efficient instantiations than all the previous schemes (including the analysed ISW-based constructions). While slightly improving the tolerated leakage probability to $p = 2^{-7.5}$, our 3-share instantiation achieves a complexity of $\mathcal{O}(\kappa^{3.9})$. For a slightly lower leakage probability, our 5-share instantiation drops the complexity to $\mathcal{O}(\kappa^{3.2})$.

We thus achieve a significant step forward in the quest for efficient random probing secure schemes that tolerate a quantified leakage probability. Besides our concrete instantiations, our work introduces several tools (new bounds, relations, and generic gadgets) that shall be instrumental for future constructions.

2 Preliminaries

Along the paper, we shall use similar notations and formalism as [9]. In particular, $\mathbb{K}$ shall denote a finite field. For any $n \in \mathbb{N}$, we shall denote $[n]$ the integer set $[n] = [1, n] \cap \mathbb{Z}$. For any tuple $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{K}^n$ and any set $I \subseteq [n]$, we shall denote $\mathbf{x}|_I = (x_i)_{i \in I}$.

2.1 Linear Sharing, Circuits, and Gadgets

In the following, the *n-linear decoding* mapping, denoted LinDec , refers to the function $\mathbb{K}^n \rightarrow \mathbb{K}$ defined as

$$\text{LinDec} : (x_1, \dots, x_n) \mapsto x_1 + \dots + x_n ,$$

for every $n \in \mathbb{N}$ and $(x_1, \dots, x_n) \in \mathbb{K}^n$. We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $(\hat{x}_1, \dots, \hat{x}_\ell) \in (\mathbb{K}^n)^\ell$ the *n-linear decoding* mapping acts as

$$\text{LinDec} : (\hat{x}_1, \dots, \hat{x}_\ell) \mapsto (\text{LinDec}(\hat{x}_1), \dots, \text{LinDec}(\hat{x}_\ell)) .$$

Definition 1 (Linear Sharing). Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an *n-linear sharing* of x is a random vector $\hat{x} \in \mathbb{K}^n$ such that $\text{LinDec}(\hat{x}) = x$. It is said to be *uniform* if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $\hat{x}|_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. A *n-linear encoding* is a probabilistic algorithm LinEnc which on input a tuple $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{K}^\ell$ outputs a tuple $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_\ell) \in (\mathbb{K}^n)^\ell$ such that $\hat{x}_i$ is a uniform *n-sharing* of x_i for every $i \in [\ell]$.

An *arithmetic circuit* on a field $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations on $\mathbb{K}$. We consider circuits composed of addition gates, $(x_1, x_2) \mapsto x_1 + x_2$, multiplication gates, $(x_1, x_2) \mapsto x_1 \cdot x_2$, and copy gates, $x \mapsto (x, x)$. A *randomized arithmetic circuit* is equipped with an additional random gate which outputs a fresh uniform random value of $\mathbb{K}$.

In the following, we shall call an $(n\text{-share}, \ell\text{-to-}m)$ *gadget*, a randomized arithmetic circuit that maps an input $\hat{\mathbf{x}} \in (\mathbb{K}^n)^\ell$ to an output $\hat{\mathbf{y}} \in (\mathbb{K}^n)^m$ such that $\mathbf{x} = \text{LinDec}(\hat{\mathbf{x}}) \in \mathbb{K}^\ell$ and $\mathbf{y} = \text{LinDec}(\hat{\mathbf{y}}) \in \mathbb{K}^m$ satisfy $\mathbf{y} = g(\mathbf{x})$ for some function g . In this paper, we shall consider gadgets for three types of functions (corresponding to the three types of gates): the addition $g : (x_1, x_2) \mapsto x_1 + x_2$, the multiplication $g : (x_1, x_2) \mapsto x_1 \cdot x_2$ and the copy $g : x \mapsto (x, x)$. We shall generally denote such gadgets G_{add} , G_{mult} and G_{copy} respectively.

2.2 Random Probing Security

Let $p \in [0, 1]$ be some constant leakage probability parameter, a.k.a. the *leakage rate*. In the p -random probing model, an evaluation of a circuit C leaks the value carried by each wire with a probability p (and leaks nothing otherwise), all the wire leakage events being mutually independent.

As in [9], we formally define the random-probing leakage of a circuit from the two following probabilistic algorithms:

- The *leaking-wires sampler* takes as input a randomized arithmetic circuit C and a probability $p \in [0, 1]$, and outputs a set $\mathcal{W}$, denoted as

$$\mathcal{W} \leftarrow \text{LeakingWires}(C, p) ,$$

where $\mathcal{W}$ is constructed by including each wire label from the circuit C with probability p to $\mathcal{W}$ (where all the probabilities are mutually independent).

- The *assign-wires sampler* takes as input a randomized arithmetic circuit C , a set of wire labels $\mathcal{W}$ (subset of the wire labels of C), and an input $\mathbf{x}$, and it outputs a $|\mathcal{W}|$ -tuple $\mathbf{w} \in (\mathbb{K} \cup \{\perp\})^{|\mathcal{W}|}$, denoted as

$$\mathbf{w} \leftarrow \text{AssignWires}(C, \mathcal{W}, \mathbf{x}) ,$$

where $\mathbf{w}$ corresponds to the assignments of the wires of C with label in $\mathcal{W}$ for an evaluation on input $\mathbf{x}$.

Definition 2 (Random Probing Leakage). *The p -random probing leakage of a randomized arithmetic circuit C on input $\mathbf{x}$ is the distribution $\mathcal{L}_p(C, \mathbf{x})$ obtained by composing the leaking-wires and assign-wires samplers as*

$$\mathcal{L}_p(C, \mathbf{x}) \stackrel{id}{=} \text{AssignWires}(C, \text{LeakingWires}(C, p), \mathbf{x}) .$$

Definition 3 (Random Probing Security). *A randomized arithmetic circuit C with $\ell \cdot n \in \mathbb{N}$ input gates is (p, ε) -random probing secure with respect to encoding Enc if there exists a simulator Sim such that for every $\mathbf{x} \in \mathbb{K}^\ell$:*

$$\text{Sim}(C) \approx_\varepsilon \mathcal{L}_p(C, \text{Enc}(\mathbf{x})) . \tag{1}$$

2.3 Expanding Compiler

In [2], Ananth, Ishai and Sahai propose an *expansion* approach to build a random-probing-secure circuit compiler from a secure multiparty protocol. This approach was later revisited by Belaïd, Coron, Prouff, Rivain, and Taleb who formalize the notion of *expanding compiler* [9].

The principle of the expanding compiler is to recursively apply a base compiler, denoted CC , and which simply consists in replacing each gate in the input circuit by the corresponding gadget. More specifically, assume we have three n -share gadgets G_{add} , G_{mult} , G_{copy} , for the addition, the multiplication, and the copy on $\mathbb{K}$. The base compiler CC simply consists in replacing each addition gate in the original gadget by G_{add} , each multiplication gate by G_{mult} , and each copy gate by G_{copy} , and by replacing each wire by n wires carrying a sharing of the original wire. One can derive three new n^2 -share gadgets by simply applying CC to each gadget: $G_{\text{add}}^{(2)} = \text{CC}(G_{\text{add}})$, $G_{\text{mult}}^{(2)} = \text{CC}(G_{\text{mult}})$, and $G_{\text{copy}}^{(2)} = \text{CC}(G_{\text{copy}})$. Doing so, we obtain n^2 -share gadgets for the addition, multiplication, and copy on $\mathbb{K}$. This process can be iterated an arbitrary number of times, say k , to an input circuit C :

$$C \xrightarrow{\text{CC}} \hat{C}_1 \xrightarrow{\text{CC}} \dots \xrightarrow{\text{CC}} \hat{C}_k.$$

The first output circuit $\hat{C}_1$ is the original circuit in which each gate is replaced by a base gadget G_{add} , G_{mult} , or G_{copy} . The second output circuit $\hat{C}_2$ is the original circuit C in which each gate is replaced by an n^2 -share gadget $G_{\text{add}}^{(2)}$, $G_{\text{mult}}^{(2)}$, or $G_{\text{copy}}^{(2)}$ as defined above. Equivalently, $\hat{C}_2$ is the circuit $\hat{C}_1$ in which each gate is replaced by a base gadget. In the end, the output circuit $\hat{C}_k$ is hence the original circuit C in which each gate has been replaced by a k -expanded gadget and each wire has been replaced by n^k wires carrying an (n^k) -linear sharing of the original wire.

This expanding compiler achieves random probing security if the base gadgets verify a property called *random probing expandability* [9].

2.4 Random Probing Expandability

We recall hereafter the original definition of the random probing expandability (RPE) property for 2-input 1-output gadgets.

Definition 4 (Random Probing Expandability [9]). *Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -random probing expandable (RPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2, J') &\leftarrow \text{Sim}_1^G(\mathcal{W}, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J', \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. *the failure events $\mathcal{F}_1 \equiv (|I_1| > t)$ and $\mathcal{F}_2 \equiv (|I_2| > t)$ verify*

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \tag{2}$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. J' is such that $J' = J$ if $|J| \leq t$ and $J' \subseteq [n]$ with $|J'| = n - 1$ otherwise,
3. the output distribution satisfies

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, (\hat{x}, \hat{y})), \hat{z}|_{J'}) \quad (3)$$

where $\hat{z} = G(\hat{x}, \hat{y})$.

The RPE notion can be simply extended to gadgets with 2 outputs: the Sim_1^G simulator takes two sets $J_1 \subseteq [n]$ and $J_2 \subseteq [n]$ as input and produces two sets J'_1 and J'_2 satisfying the same property as J' in the above definition (w.r.t. J_1 and J_2). The Sim_2^G simulator must then produce an output including $\hat{z}_1|_{J'_1}$ and $\hat{z}_2|_{J'_2}$ where $\hat{z}_1$ and $\hat{z}_2$ are the output sharings. The RPE notion can also be simply extended to gadgets with a single input: the Sim_1^G simulator produces a single set I so that the failure event ($|I| > t$) occurs with probability ε (and the Sim_2^G simulator is then simply given $\hat{x}|_I$ where $\hat{x}$ is the single input sharing). We refer the reader to [9] for the formal definitions of these variants. Eventually, the RPE notion can also be extended to gadgets with an arbitrary number ℓ of inputs. The Sim_1^G simulator then produces ℓ sets $I_1, \dots, I_\ell$ so that the corresponding failures ($|I_1| > t$), $\dots$, ($|I_\ell| > t$) occur with probability ε and are additionally mutually independent. The Sim_2^G simulator then simply gets use of the shares of each input as designated respectively by the corresponding sets $I_1, \dots, I_\ell$.

Note that as explained in [9], the requirement of the RPE notion on the mutual independence of the failure events might seem too strong. We can actually use the proposed relaxation referred to as *weak random probing expandability*. Namely, the equalities (Equation (2)) are replaced by inequalities as upper bounds are sufficient in our context. We refer the reader to [9] for the concrete reduction, which does not impact the amplification orders.

2.5 Complexity of the Expanding Compiler

We start by recalling the definition of the *amplification order* of a function and of a gadget.

Definition 5 (Amplification Order).

- Let $f : \mathbb{R} \rightarrow \mathbb{R}$ which satisfies

$$f(p) = c_d p^d + \mathcal{O}(p^{d+\varepsilon})$$

as p tends to 0, for some $c_d > 0$ and $\varepsilon > 0$. Then d is called the *amplification order* of f .

- Let $t > 0$ and G a gadget. Let d be the maximal integer such that G achieves (t, f) -RPE for $f : \mathbb{R} \rightarrow \mathbb{R}$ of amplification order d . Then d is called the *amplification order* of G (with respect to t).

We stress that the amplification order of a gadget G is defined with respect to the RPE threshold t . Namely, different RPE thresholds t are likely to yield different amplification orders d for G (or equivalently d can be thought of as a function of t).

As shown in [9], the complexity of the expanding compiler relates to the (minimum) amplification order of the three gadgets used in the base compiler CC. If the latter achieves (t, f) -RPE with an amplification order d , the expanding compiler achieves $(p, 2^{-\kappa})$ -random probing security with a complexity blowup of $\mathcal{O}(\kappa^e)$ for an exponent e satisfying

$$e = \frac{\log N_{\max}}{\log d} \quad (4)$$

with

$$N_{\max} = \max \left(N_{m,m}, \text{eigenvalues} \left(\begin{pmatrix} N_{a,a} & N_{c,a} \\ N_{a,c} & N_{c,c} \end{pmatrix} \right) \right) \quad (5)$$

where $N_{x,y}$ denotes the number of gates “x” in a gadget “y”, with “m” meaning multiplication, “a” meaning addition, and “c” meaning copy. As an illustration, the instantiation proposed in [9] satisfies $N_{\max} = 21$ and $d = \frac{3}{2}$ which yields an asymptotic complexity of $\mathcal{O}(\kappa^{7.5})$.

Finally, we recall the notion of maximum *tolerated leakage probability* which corresponds to the maximum value p for which we have $f(p) < p$. This happens to be a necessary and sufficient condition for the expansion strategy to apply with (t, f) -RPE gadgets. The instantiation proposed in [9] tolerates a leakage probability up to $2^{-7.80}$.

3 Bounding the Amplification Order

As recalled above, the amplification order of a gadget is a crucial parameter of its random probing expandability. The higher the amplification order, the lower the asymptotic complexity of the expanding compiler, *ceteris paribus*. A natural question which was left open in [9] is to determine the best amplification order that can be hoped for given the different parameters of a gadget. In this section, we exhibit concrete upper bounds on the amplification order that can be achieved by a gadget depending on its input-output dimensions (ℓ, m) , its number of shares n , and its RPE threshold t .

Before giving the bounds let us make a key observation on the amplification order of a gadget. Let G be a 2-to-1 n -share gadget achieving (t, f) -RPE. A subset $\mathcal{W}$ of the wires of G is said to be a *failure set* with respect to the first input (resp. the second input) if there exists a set $J \subseteq [n]$ such that $(I_1, I_2, J') \leftarrow \text{Sim}_1^G(\mathcal{W}, J)$ implies $|I_1| > t$ (resp. $|I_2| > t$), namely if a leaking set $\mathcal{W}$ implies the failure event $\mathcal{F}_1$ (resp. $\mathcal{F}_2$) in the definition of RPE. One can check that G has amplification order $d \leq d_{up}$ if one of the two following events occurs:

1. there exists a failure set $\mathcal{W}$ w.r.t. the first input *or* the second input such that $|\mathcal{W}| = d_{up}$,
2. there exists a failure set $\mathcal{W}$ w.r.t. the first input *and* the second input such that $|\mathcal{W}| = 2d_{up}$.

In the former case, the existence of the failure set implies that the function $f(p)$ has a non-zero coefficient in $p^{d_{up}}$ and hence $d \leq d_{up}$. In the latter case, the existence of the double failure set implies that the function $f^2(p)$ has a non-zero coefficient in $p^{2d_{up}}$ and hence $d \leq d_{up}$. The case of a single-input gadget is simpler: it has amplification order $d \leq d_{up}$ if there exists a failure set $\mathcal{W}$ (w.r.t. its single input) such that $|\mathcal{W}| = d_{up}$.

We start by exhibiting a generic upper bound for the amplification order and then look at the particular case of what we shall call a *standard* multiplication gadget.

3.1 Generic Upper Bound

In the following we will say that a function $g : \mathbb{K}^\ell \rightarrow \mathbb{K}^m$ is *complete* if at least one of its m outputs is functionally dependent on the ℓ inputs. Similarly, we say that a gadget G is complete if its underlying function g is complete.

The following lemma gives our generic upper bound on the amplification order.

Lemma 1. *Let $f : \mathbb{R} \rightarrow \mathbb{R}$, $n \in \mathbb{N}$ and $\ell, m \in \{1, 2\}$. Let $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ be an ℓ -to- m n -share complete gadget achieving (t, f) -RPE. Then its amplification order d is upper bounded by*

$$\min((t+1), (3-\ell) \cdot (n-t)).$$

Proof. The first part of the bound on the amplification order $d \leq (t + 1)$ is immediate since by probing $t + 1$ shares of any input, the considered set will be a failure set of cardinality $t + 1$. We then consider two cases depending on the number of inputs:

1. *1-input gadgets* ($\ell = 1$): We show that we can exhibit a failure set of size $2(n - t)$. Let us denote the output shares $z_1, \dots, z_n$ (for two-output gadgets, *i.e.* $m = 2$, $z_1, \dots, z_n$ can be any of the output sharings). In the evaluation of the (t, f) -RPE property, t shares among the z_i 's (corresponding to the set J) must be simulated. Without loss of generality, let $z_1, \dots, z_t$ be those shares (*i.e.* $J = [t]$). By including both input gates of each of the remaining output shares $z_{t+1}, \dots, z_n$ in the set $\mathcal{W}$, the distribution to be simulated requires the knowledge of the full input (by completeness of the gadget). The set $\mathcal{W}$ is thus a failure set with $2(n - t)$ elements.
2. *2-input gadgets* ($\ell = 2$): Considering the same failure set as in the above case, the simulation of *out* requires the full two input sharings. Hence $\mathcal{W}$ is a failure set of size $2(n - t)$ with respect to the two inputs, and so the amplification order satisfies $d \leq (n - t)$.

We hence conclude that $d \leq \min((t + 1), 2(n - t))$ for one-input gadgets, and $d \leq \min((t + 1), (n - t))$ for two-input gadgets. $\square$

Corollary 1 (One-input gadget). *The amplification order d of a one-input gadget achieving (t, f) -RPE is upper bounded by*

$$d \leq \frac{2(n + 1)}{3}.$$

The above corollary directly holds from Lemma 1 for a RPE threshold $t = \frac{2n-1}{3}$ (which balances the two sides of the min).

Corollary 2 (Two-input gadget). *The amplification order d of a two-input gadget achieving (t, f) -RPE is upper bounded by*

$$d \leq \frac{n + 1}{2}.$$

The above corollary directly holds from Lemma 1 for a RPE threshold $t = \frac{n-1}{2}$ (which balances the two sides of the min).

We deduce from the two above corollaries that for a circuit composed of addition, multiplication and copy gadgets, the amplification order is upper bounded

$$d \leq \min\left(\frac{2(n + 1)}{3}, \frac{n + 1}{2}\right) = \frac{n + 1}{2},$$

which can only be achieved for an odd number of shares by taking $t = \frac{n-1}{2}$ as RPE threshold.

3.2 Upper Bound for Standard Multiplication Gadgets

The generic bound exhibited above is not tight in the special case of a standard multiplication gadget which computes cross products between the input shares, such as the ISW multiplication gadget [17]. We exhibit hereafter a tighter bound for such gadgets.

Formally, a n -share multiplication gadget G is a *standard multiplication gadget*, if on input $(\mathbf{x}, \mathbf{y}) \in (\mathbb{K}^n)^2$, G computes the cross products $x_i \cdot y_j$ for $1 \leq i, j \leq n$. Our upper bound on the amplification order for such gadgets is given in the following lemma.

Lemma 2. *Let $f : \mathbb{R} \rightarrow \mathbb{R}$ and $n \in \mathbb{N}$. Let G be an n -share standard multiplication gadget achieving (t, f) -RPE. Then its amplification order d is upper bounded by*

$$d \leq \min \left(\frac{t+1}{2}, (n-t) \right).$$

Proof. The second part of the bound $(n-t)$ holds directly from Lemma 1. We now prove the bound $(t+1)/2$ by exhibiting a failure set of size $t+1$ with t output shares, which will be a failure on both inputs. Let $\{m_{ij}\}_{0 \leq i, j \leq n}$ denote the cross products such that $m_{ij} = x_i \cdot y_j$. Consider a set $\mathcal{W}$ made of $t+1$ such variables $\{m_{ij}\}$ for which the indexes i and j are all distinct. Specifically, $\mathcal{W} = \{x_{i_1} \cdot y_{j_1}, \dots, x_{i_{t+1}} \cdot y_{j_{t+1}}\}$ such that $\{i_\ell\}_{1 \leq \ell \leq t+1}$ and $\{j_\ell\}_{1 \leq \ell \leq t+1}$ are both sets of $(t+1)$ distinct indexes. Clearly, such a set is a failure set for both inputs $\mathbf{x}$ and $\mathbf{y}$ since it requires $t+1$ shares of each of them to be perfectly simulated (even without considering the output shares to be also simulated). We hence have a double failure set of cardinality $t+1$ which implies the $(t+1)/2$ upper bound on the amplification order. $\square$

The above lemma implies that the highest amplification order for standard multiplication gadgets might be achieved for a RPE threshold $t = \frac{2n-1}{3}$ which yields the following maximal upper bound:

$$d \leq \frac{n+1}{3},$$

which is lower than the generic upper bound for 2-to-1 gadgets exhibited in Corollary 2. This loss suggests that better amplification orders could be achieved for multiplication gadgets that do not compute direct cross products of the input shares. We actually provide new constructions of multiplication gadgets avoiding this loss in Section 5.

4 A Closer Look at Random Probing Expandability

In this section, we give a closer look at the RPE notion. We first show that it naturally splits into two different notions, that we shall call RPE1 and RPE2, and further introduce a tighter variant which will be useful for our purpose. We then study the relations between (tight) RPE and the *Strong Non-Interference* (SNI) notion used for probing security. We exhibit strong connections between (tight) RPE1 and SNI, which will be very useful for our constructive results depicted in Section 5.

4.1 Splitting RPE

From Definition 4, we can define two sub-properties which are jointly equivalent to RPE. In the first one, designated by RPE1, the set J is constrained to satisfy $|J| \leq t$ and $J' = J$ (the simulator does not choose J'). In the second one, designated by RPE2, J' is chosen by the simulator such that $J' \subseteq [n]$ with $|J'| = n-1$ (and J does not matter anymore). For the sake of completeness, these two notions are formally defined in Appendix A.

This split is somehow a partition of the RPE notion since we have:

$$G \text{ is } (t, f)\text{-RPE} \iff G \text{ is } (t, f)\text{-RPE1 and } G \text{ is } (t, f)\text{-RPE2}$$

for any gadget G . As a result of the above equivalence, we can show that a gadget achieves RPE1 and RPE2 independently in order to obtain RPE for this gadget. Formally, we use the following lemma.

Lemma 3. *An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ which is (t, f_1) -RPE1 and (t, f_2) -RPE2 is also (t, f) -RPE with $f(p) \geq \max(f_1(p), f_2(p))$ for every $p \in [0, 1]$.*

We can refine the upper bounds introduced in Section 3 with respect to this split. In Lemma 1, the bound $d \leq t + 1$ applies to both RPE1 and RPE2, while the bound $d \leq (3 - \ell) \cdot (n - t)$ only applies to RPE1. Similarly, in Lemma 2, the bound $d \leq (t + 1)/2$ applies to both RPE1 and RPE2, while the bound $d \leq (n - t)$ only applies to RPE1.

4.2 Tightening RPE

We introduce a tighter version of the RPE security property. The so-called *tight random probing expandability* (TRPE) is such that a failure occurs when the simulation requires more than t input shares (as in the original RPE notion) but also whenever this number of shares is greater than the size of the leaking set $\mathcal{W}$. Formally, the failure event $\mathcal{F}_j$ is defined as

$$\mathcal{F}_j \equiv (|I_j| > \min(t, |\mathcal{W}|))$$

for every $j \in [\ell]$.

This tighter security property will be instrumental in the following to obtain generic RPE constructions. Similarly to the original RPE property, the TRPE property can be split into two intermediate properties, namely TRPE1 and TRPE2 and Lemma 3 also applies to the case of TRPE. Moreover the upper bounds on the amplification order for RPE in Lemmas 1 and 2 further apply to the amplification order for TRPE (which holds by definition). The formal TRPE, TRPE1, and TRPE2 definitions are given in Appendix B for the sake of completeness.

We show hereafter that the TRPE notion is actually equivalent to the RPE notion if and only if the function f is of maximal amplification order $t + 1$.

Lemma 4. *Let $t \in \mathbb{N}$, let $f : \mathbb{R} \rightarrow \mathbb{R}$ of amplification order d . Let G be a gadget.*

1. *If G achieves (t, f) -TRPE, then it achieves (t, f') -RPE for some $f' : \mathbb{R} \rightarrow \mathbb{R}$ of amplification order $d' \geq d$.*
2. *If G is of amplification order d with respect to t (i.e. d is the max amplification order of a function f for which G is (t, f) -RPE), then for all $f' : \mathbb{R} \rightarrow \mathbb{R}$ for which G achieves (t, f') -TRPE, f' is of amplification order $d' \leq d$.*
3. *If $d = t + 1$, then G achieves (t, f) -TRPE if and only if G achieves (t, f) -RPE.*

Proof. The proof for the first two points is easy. In particular, for the first point, if G achieves TRPE with an amplification order of d , then G achieves RPE with amplification order at least d , since a failure in the TRPE setting i.e. $|I_j| > \min(t, |\mathcal{W}|)$ does not necessarily imply a failure in the RPE setting i.e. $|I_j| > t$, meanwhile if there is no failure for TRPE for a leaking set of wires $\mathcal{W}$, then this implies that $|I_j| \leq \min(t, |\mathcal{W}|) \leq t$ so there is no failure in the RPE setting either.

As for the second point, the proof is similar: if G achieves an amplification of d in the RPE setting, then it achieves an amplification order of at most d in the TRPE setting, since a failure in the RPE setting i.e. $|I_j| > t$ immediately implies a failure in the TRPE setting $|I_j| > \min(t, |\mathcal{W}|)$. But also, even if there is no failure for a leaking set of wires $\mathcal{W}$ in the RPE setting we might still have a failure in the TRPE setting for the same set $\mathcal{W}$. This is mainly the case where $\mathcal{W}$ can be simulated with sets of input shares I_j such that $|\mathcal{W}| < |I_j| \leq t$, so we have $|I_j| \leq t$ (i.e. no failure

for RPE) and $|I_j| > \min(t, |\mathcal{W}|) = |\mathcal{W}|$ (*i.e.* failure on TRPE). This concludes the proof for the second point.

We will now prove the third point. Let $d = t + 1$. We will show that for every set $J' \subseteq [n]$ of output shares and every leaking set of wires $\mathcal{W}$, a failure occurs in the TRPE setting if and only if a failure also occurs in the RPE setting. If $|\mathcal{W}| \geq t$, then the two settings are equivalent since $\min(t, |\mathcal{W}|) = t$. We will thus only focus on the case $|\mathcal{W}| < t$. Clearly, a failure in the RPE setting, *i.e.* $|I_j| > t$, implies a failure in the TRPE setting, *i.e.* $|I_j| > \min(t, |\mathcal{W}|)$. Let us now show that the converse is also true.

We assume by contradiction that there exists J' and $\mathcal{W}$ implying a TRPE failure which is not an RPE failure, that is a set I_j satisfying $|\mathcal{W}| < |I_j| \leq t$. We then show that there exists a leaking set $\mathcal{W}'$ of size $|\mathcal{W}'| < t + 1$ for which an RPE failure always occurs, which implies an amplification order strictly lower than $t + 1$ and hence contradicts the lemma hypothesis. This set $\mathcal{W}'$ is constructed as $\mathcal{W}' = \mathcal{W} \cup I'_j$ for some set $I'_j \subset [n] \setminus I_j$ such that $|I'_j| = t + 1 - |I_j|$. The simulation of $\mathcal{W}'$ and J' then requires the input shares from $I_j \cup I'_j$. However, we have

$$|I_j \cup I'_j| = |I_j| + |I'_j| = t + 1$$

implying an RPE failure, and

$$|\mathcal{W}'| = |\mathcal{W} \cup I'_j| \leq |\mathcal{W}| + |I'_j| = |\mathcal{W}| + t + 1 - |I_j| < |\mathcal{W}| + t + 1 - |\mathcal{W}| = t + 1.$$

Thus, we have built a failure set $\mathcal{W}'$ of size strictly less than the amplification order $t + 1$, which contradicts the hypothesis and hence concludes the proof. $\square$

The above proof also applies to the case of the split notions, specifically for $((t, f)$ -RPE1, (t, f) -TRPE1) and for $((t, f)$ -RPE2, (t, f) -TRPE2).

4.3 Unifying (Tight) RPE and SNI

Strong non-interference (SNI) is a widely used notion to compose probing-secure gadgets [5]. In [9], the authors exhibit a relation between the SNI and the *random probing composability* (RPC) property in their Proposition 1. We go one step further and study the relation between SNI and (T)RPE.

We state hereafter some equivalence results between the (T)RPE1 and SNI notions, up to some constraints on the parameters. Let us first recall the definition of the SNI notion.

Definition 6 (Strong Non-Interference (SNI)). *Let n, ℓ and τ be positive integers. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow \mathbb{K}^n$ is τ -SNI if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every set $J \subseteq [n]$ and subset $\mathcal{W}$ of wire labels from G satisfying $|\mathcal{W}| + |J| \leq \tau$, the following random experiment with any $\hat{\mathbf{x}} \in (\mathbb{K}^n)^\ell$*

$$\begin{aligned} \mathbf{I} &\leftarrow \text{Sim}_1^G(\mathcal{W}, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(\hat{\mathbf{x}}|_{\mathbf{I}}) \end{aligned}$$

yields

$$|I_1| \leq |\mathcal{W}|, \dots, |I_\ell| \leq |\mathcal{W}| \tag{6}$$

and

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, \hat{\mathbf{x}}), \hat{\mathbf{y}}|_J) \tag{7}$$

where $\mathbf{I} = (I_1, \dots, I_\ell)$ and $\hat{\mathbf{y}} = G(\hat{\mathbf{x}})$.

We first formally show that (T)RPE1 implies SNI.

Lemma 5. *Let $t \in \mathbb{N}$ and $f : \mathbb{R} \rightarrow \mathbb{R}$ of amplification order $t+1$. Let G be a gadget which achieves (t, f) -TRPE1. Then G is also t -SNI.*

Proof. By definition of TRPE1 and by hypothesis on the amplification order, there exist input sets $I_1, \dots, I_\ell$ which can perfectly simulate any leaking wires set $\mathcal{W}$ such that $|\mathcal{W}| \leq t$ and any set of output shares J such that $|J| \leq t$, satisfying $|I_1|, \dots, |I_\ell| \leq |\mathcal{W}|$. Consequently, there exist input sets $I_1, \dots, I_\ell$ which can perfectly simulate any leaking wires set $\mathcal{W}$ such that $|\mathcal{W}| = t_i \leq t$ and any set of output shares J such that $|\mathcal{W}| + |J| \leq t$ with $|I_1|, \dots, |I_\ell| \leq t_i$. G is thus t -SNI. $\square$

We now show that SNI implies TRPE1 up to some constraints on the parameters t and τ .

Lemma 6. *Let $\tau, \ell \in \mathbb{N}$. Let G be an ℓ -to-1 gadget which achieves τ -SNI. Then G satisfies (t, f) -TRPE1 for some $f : \mathbb{R} \rightarrow \mathbb{R}$ with an amplification order of*

$$d \geq \frac{1}{\ell} \min(t+1, \tau-t+1) .$$

Proof. Since G is τ -SNI, then for any set of leaking wires $\mathcal{W}$ and output shares J such that $|\mathcal{W}| + |J| \leq \tau$, the wires indexed by $\mathcal{W}$ and the output shares indexed by J can be perfectly simulated from input shares indexed by $I_1, \dots, I_\ell$ such that $|I_j| \leq |\mathcal{W}|$ for every $1 \leq j \leq \ell$. In the TRPE1 property, the set J of output shares can be any set of size $|J| \leq t$ so we can assume $|J| = t$ without loss of generality.

For a leaking set $\mathcal{W}$ of size $|\mathcal{W}| < \min(t+1, \tau-t+1)$ no failure event occurs. Indeed τ -SNI and $|\mathcal{W}| < \tau - t + 1$ implies $|\mathcal{W}| + |J| \leq \tau$ and hence the existence of the sets $I_1, \dots, I_\ell$ allowing the simulation with $|I_j| \leq |\mathcal{W}|$. And $|\mathcal{W}| < t + 1$ implies $|I_j| \leq \min(t, |\mathcal{W}|)$ for every j which implies the absence of failure. Then for a leaking set $\mathcal{W}$ of size $|\mathcal{W}| \geq \min(t+1, \tau-t+1)$, no condition remains to rule out simulation failures and one could actually get a failure for every input. In the latter case, the amplification order would equal $\frac{1}{\ell} \min(t+1, n-t)$, but in all generality it could be higher (*i.e.* this value is a lower bound). $\square$

An illustrative summary of the relations between RPE1, TRPE1 and SNI is depicted in Figure 1 (d denotes the amplification order of the function f). We hence observe an equivalence between the three notions up to some constraints on the parameters t , d , τ and ℓ .

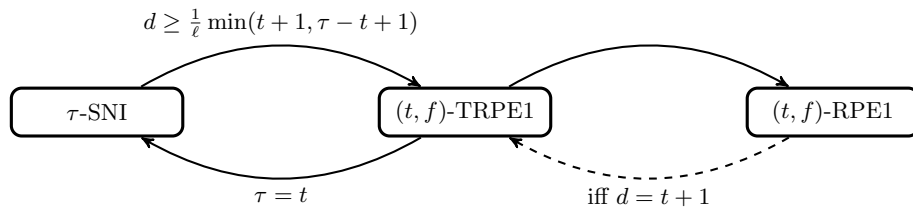


Fig. 1: Summary of relations between the different notions.

Relation and separation between (T)RPE2 and SNI. For a given n -share gadget G , the (T)RPE2 notion exclusively focuses on the simulation of a set of leaking intermediate variables together with a chosen set of $(n - 1)$ output shares. If G is τ -SNI for $\tau < n - 1$, then nothing can be claimed on the simulation of the latter sets. But if G is $(n - 1)$ -SNI, then any set of $(n - 1)$ output shares can be perfectly simulated without the knowledge of any input share. Concretely, it implies that G is (t, f) -(T)RPE2 of amplification order at least 1 as a chosen output set of $(n - 1)$ shares alone can be perfectly simulated without any additional knowledge on the input shares. Namely, we have

$$(n - 1)\text{-SNI} \Rightarrow (t, f)\text{-(T)RPE2 of amplification order at least 1.}$$

Nevertheless, there is no relation from τ -SNI to (t, f) -(T)RPE2 for amplification orders strictly greater than 1 as (T)RPE2 would then consider leaking sets of size larger than or equal to n (for n -share gadgets, $\tau < n$). On the other side, there is no direct implication either from (t, f) -(T)RPE2 to τ -SNI since the former property does not consider all possible output sets of size $(n - 1)$, but only a chosen one.

5 Generic Constructions

To the best of our knowledge, the only RPE gadgets in the literature are the ones designed in [9] which are restricted to a small number of shares, specifically $n \in \{2, 3\}$. A natural open question is the definition of RPE gadgets with good amplification orders, typically achieving or approaching the upper bounds exhibited in Section 3, for *any* number of shares n . In this section, we exhibit copy, addition, and multiplication gadgets derived from the widely known Ishai-Sahai-Wagner (ISW) construction [17]. Based on the results demonstrated in Section 4, we are able to show that these gadgets achieve RPE for any number of shares n with amplification orders close to the upper bounds (up to a small constant factor). We further provide an asymptotic analysis of the expanding compiler using these gadgets as well as a new multiplication gadget reaching the optimal amplification order hence improving the convergence to a better asymptotic complexity.

5.1 Generic Copy and Addition Gadgets

As intuitively proposed in [9] for small gadgets, copy and addition gadgets can be naturally derived from a refresh gadget. Such a gadget takes one sharing as input and outputs a new refreshed sharing of the same value. We formally introduce these natural constructions hereafter and show that their RPE security can be reduced to that of the underlying refresh gadget.

Generic Copy Gadget. Algorithm 1 displays the generic construction for the copy gadget from a refresh gadget. It simply consists in refreshing the input sharing twice to obtain two fresh copies.

Algorithm 1: Copy gadget G_{copy}

Input : $(a_1, \dots, a_n)$ input sharing
Output: $(e_1, \dots, e_n), (f_1, \dots, f_n)$ fresh copies of $(a_1, \dots, a_n)$
 $(e_1, \dots, e_n) \leftarrow G_{\text{refresh}}(a_1, \dots, a_n);$
 $(f_1, \dots, f_n) \leftarrow G_{\text{refresh}}(a_1, \dots, a_n);$

We have the following lemma (see the proof in Appendix C).

Lemma 7. *Let G_{refresh} be an n -share (t, f) -TRPE refresh gadget of amplification order d . Then, the copy gadget G_{copy} displayed in Algorithm 1 is (t, f') -TRPE also of amplification order d .*

As a consequence of this result, a TRPE refresh gadget directly yields a TRPE copy gadget achieving the same amplification order. Both gadgets can then reach the upper bound for 1-input gadgets whenever $t + 1 = 2(n - t)$ implying an amplification order $d = \frac{2(n+1)}{3}$.

Generic Addition Gadget. Algorithm 2 displays the generic construction for the addition gadget from a refresh gadget. It simply consists in refreshing both input sharings before adding them.

Algorithm 2: Addition Gadget G_{add}
Input : $(a_1, \dots, a_n), (b_1, \dots, b_n)$ input sharings
Output: $(c_1, \dots, c_n)$ sharing of $a + b$
$(e_1, \dots, e_n) \leftarrow G_{\text{refresh}}(a_1, \dots, a_n);$
$(f_1, \dots, f_n) \leftarrow G_{\text{refresh}}(b_1, \dots, b_n);$
$(c_1, \dots, c_n) \leftarrow (e_1 + f_1, \dots, e_n + f_n);$

We have the following lemma (see the proof in Appendix D).

Lemma 8. *Let G_{refresh} be an n -share refresh gadget and let G_{add} be the corresponding addition gadget displayed in Algorithm 2. Then if G_{refresh} is (t, f) -RPE (resp. (t, f) -TRPE) of amplification order d , then G_{add} is (t, f') -RPE (resp. (t, f') -TRPE) for some f' of amplification order $d' \geq \lfloor \frac{d}{2} \rfloor$.*

The above lemma shows that a (T)RPE refresh gadget of amplification order d directly yields a (T)RPE addition gadget of amplification order at least $\lfloor \frac{d}{2} \rfloor$. If the refresh gadget achieves the optimal $d = \frac{2(n+1)}{3}$, then the generic addition gadget has an amplification order at least $\lfloor \frac{n}{3} \rfloor$ which is not far from the upper bound for two-input gadgets of $\frac{n+1}{2}$.

We stress that the results of Lemma 7 and Lemma 8 are general and apply for any refresh gadget satisfying the (T)RPE property. In the rest of the section, we shall focus on a particular refresh gadget, namely the ISW-based refresh gadget. We show that this gadget achieves (T)RPE from which we obtain (T)RPE copy and addition gadgets for any number of shares n and with amplification orders close to the upper bound (up to a small constant factor).

5.2 ISW-based Copy and Addition Gadgets

As a basis of further constructions, we focus our analysis on the most deployed refresh gadget, which is based on the ISW construction [17].

ISW Refresh Gadget. This refresh can be seen as an ISW multiplication between the input sharing and the n -tuple $(1, 0, \dots, 0)$. This is formally depicted in Algorithm 3.

Algorithm 3: ISW Refresh

Input : $(a_1, \dots, a_n)$ input sharing, $\{r_{ij}\}_{1 \leq i < j \leq n}$ random values

Output: $(c_1, \dots, c_n)$ such that $c_1 + \dots + c_n = a_1 + \dots + a_n$

for $i \leftarrow 1$ **to** n **do**

$c_i \leftarrow a_i$;

end

for $i \leftarrow 1$ **to** n **do**

for $j \leftarrow 1$ **to** $i - 1$ **do**

$c_i \leftarrow c_i + r_{ji}$;

end

for $j \leftarrow i + 1$ **to** n **do**

$c_i \leftarrow c_i + r_{ij}$;

end

end

return $(c_1, \dots, c_n)$;

We demonstrate through Lemma 9 that the ISW refresh gadget satisfies TRPE with an amplification order close to the optimal one. The proof is given in Appendix E.

Lemma 9. *Let $n \in \mathbb{N}$. For every $t \leq n - 2$, the n -share ISW refresh gadget is (t, f_1) -TRPE1 and (t, f_2) -TRPE2 for some functions $f_1, f_2 : \mathbb{R} \rightarrow \mathbb{R}$ of amplification orders d_1, d_2 which satisfy:*

- $d_1 = \min(t + 1, n - t)$ for f_1 ,
- $d_2 = t + 1$ for f_2 .

Corollary 3 then directly follows from Lemma 3 applied to TRPE and Lemma 9.

Corollary 3. *Let $n \in \mathbb{N}$. For every $t \leq n - 2$, the n -share ISW refresh gadget is (t, f) -TRPE of amplification order*

$$d = \min(t + 1, n - t).$$

According to Lemma 1, the upper bound on the amplification order of 1-input gadgets is $d \leq \min(t + 1, 2(n - t))$ which gives $d \leq \frac{2n+2}{3}$ for $t = \frac{2n-1}{3}$. In contrast, the ISW refresh gadget reaches $d = \lfloor \frac{n+1}{2} \rfloor$ by taking $t = \lceil \frac{n-1}{2} \rceil$. While applying this result to the generic constructions of addition and copy gadgets introduced above, we obtain:

- a copy gadget of amplification order $d_c = \lfloor \frac{n+1}{2} \rfloor$ (Lemma 7),
- an addition gadget of amplification order at least $d_a = \lfloor \frac{n+1}{4} \rfloor$ (Lemma 8).

In the following, we demonstrate a tighter result than Lemma 8 for the ISW-based addition gadget (namely which does not imply the loss of a factor 2).

ISW-based Copy Gadget. The copy gadget G_{copy} that uses the n -share ISW refresh gadget as a building block in Algorithm 1 achieves the same amplification order as the ISW refresh for the TRPE setting, *i.e.* $d = \min(t + 1, n - t)$. This is a direct implication from Lemma 7. Then, from Lemma 4, we have that ISW-based G_{copy} also achieves (t, f') -RPE with amplification order $d' \geq d$. We can actually prove that ISW-based G_{copy} achieves (t, f') -RPE with amplification order d' exactly equal to the amplification order in the TRPE setting, *i.e.* $d' = d = \min(t + 1, n - t)$. This is stated in the following lemma which proof is given in Appendix F.

Lemma 10. *Let G_{copy} be the n -share copy gadget displayed in Algorithm 1 and instantiated with the ISW refresh gadget. Then for every $t \leq n - 2$, G_{copy} achieves (t, f) -RPE with amplification order $d = \min(t + 1, n - t)$.*

ISW-based Addition Gadget. The addition gadget G_{add} that uses the n -share ISW refresh gadget as a building block in Algorithm 2 achieves the same amplification order as the ISW refresh gadget, which is tighter than the bound from Lemma 8. This is stated in the following Lemma, which follows from Lemma 9, and from the fact that ISW refresh is $(n - 1)$ -SNI. The proof is given in Appendix G.

Lemma 11. *Let G_{add} be the n -share addition gadget displayed in Algorithm 2 and instantiated with the ISW refresh gadget. Then for every $t \leq n - 2$, G_{add} achieves (t, f_1) -TRPE1 and (t, f_2) -TRPE2 for some functions $f_1, f_2 : \mathbb{R} \rightarrow \mathbb{R}$ of amplification orders d_1, d_2 which satisfy:*

- $d_1 = \min(t + 1, n - t)$,
- $d_2 = t + 1$.

Corollary 4 then directly follows from Lemma 11 by applying Lemma 3 ($\text{TRPE1} \cap \text{TRPE2} \Rightarrow \text{TRPE}$) and Lemma 4 ($\text{TRPE} \Rightarrow \text{RPE}$).

Corollary 4. *Let $n \in \mathbb{N}$. For every $t \leq n - 2$, the n -share gadget G_{add} displayed in Algorithm 2 and instantiated with the ISW refresh gadget is (t, f) -RPE of amplification order $d = \min(t + 1, n - t)$.*

5.3 ISW Multiplication Gadget

In contrast to the copy and addition gadgets that are built from generic schemes with a refresh gadget as a building block, the multiplication gadget can be directly defined as the standard ISW multiplication, which is recalled in Algorithm 4.

Algorithm 4: ISW Multiplication

Input : $(a_1, \dots, a_n), (b_1, \dots, b_n)$ input sharings, $\{r_{ij}\}_{1 \leq i < j \leq n}$ random values

Output: $(c_1, \dots, c_n)$ sharing of $a \cdot b$

for $i \leftarrow 1$ **to** n **do**

$c_i \leftarrow a_i \cdot b_i$;

end

for $i \leftarrow 1$ **to** n **do**

for $j \leftarrow i + 1$ **to** n **do**

$c_i \leftarrow c_i + r_{ij}$;

$r_{ji} \leftarrow (a_i \cdot b_j + r_{ij}) + a_j \cdot b_i$;

$c_j \leftarrow c_j + r_{ji}$;

end

end

return $(c_1, \dots, c_n)$;

We have the following lemma (see the proof in Appendix H).

Lemma 12. *Let $n \in \mathbb{N}$. For every $t \leq n - 2$, the n -share ISW multiplication gadget displayed in Algorithm 4 is (t, f_1) -RPE1 and (t, f_2) -RPE2 for some functions $f_1, f_2 : \mathbb{R} \rightarrow \mathbb{R}$ of amplification orders d_1, d_2 which satisfy:*

$$\begin{aligned} - d_1 &= \frac{\min(t+1, n-t)}{2}, \\ - d_2 &= \frac{t+1}{2}. \end{aligned}$$

Corollary 5 then directly follows from Lemma 12 by applying Lemma 3 ($\text{RPE1} \cap \text{RPE2} \Rightarrow \text{RPE}$).

Corollary 5. *Let $n \in \mathbb{N}$. For every $t \leq n - 2$, the n -share ISW multiplication gadget displayed in Algorithm 4 is (t, f) -RPE of amplification order*

$$d = \frac{\min(t+1, n-t)}{2}.$$

According to Lemma 2, the upper bound on the amplification order of a standard multiplication gadget (*i.e.* which starts with the cross-products of the input shares) is $d \leq \min((t+1)/2, (n-t))$ which gives $d \leq (n+1)/3$ for $t = (2n-1)/3$. In contrast, the ISW multiplication gadget reaches $d = \lfloor \frac{n+1}{4} \rfloor$ by taking $t = \lceil \frac{n-1}{2} \rceil$.

5.4 Application to the Expanding Compiler

As recalled in Section 2.5, instantiating the expanding compiler with three RPE base gadgets gives a $(p, 2^{-\kappa})$ -random probing secure compiler (*i.e.* achieving κ bits of security against a leakage probability p) with a complexity blowup of $\mathcal{O}(\kappa^e)$ for an exponent e satisfying

$$e = \frac{\log N_{\max}}{\log d}$$

where $N_{\max}$ satisfies (5) and where d is the minimum amplification order of the three base gadgets.

We can instantiate the expanding compiler using the above ISW-based gadgets. Specifically, we use the ISW multiplication for the multiplication gadget G_{mult} , and the generic constructions of addition and copy gadgets based on the ISW refresh. From Lemmas 10, 11, and 12, the maximum amplification order achievable by the compiler is the minimum of the three gadgets, which is the order of the ISW multiplication gadget:

$$d = \frac{\min(t+1, n-t)}{2}.$$

Hence, for a given number of shares n , the maximum amplification order achievable is

$$d_{\max} = \left\lfloor \frac{n+1}{4} \right\rfloor$$

which is obtained for $t = \lceil \frac{n-1}{2} \rceil$. On the other hand, the value of $N_{\max}$ can be characterized in terms of the number of shares n from the ISW algorithm. Recall from Section 2.5 that

$$N_{\max} = \max \left(N_{\text{m,m}}, \text{eigenvalues} \left(\begin{pmatrix} N_{\text{a,a}} & N_{\text{c,a}} \\ N_{\text{a,c}} & N_{\text{c,c}} \end{pmatrix} \right) \right).$$

In the case of the ISW-based gadgets, we have $N_{m,m} = n^2$ and

$$\begin{pmatrix} N_{a,a} & N_{c,a} \\ N_{a,c} & N_{c,c} \end{pmatrix} = \begin{pmatrix} n(2n-1) & 2n(n-1) \\ n(n-1) & n^2 \end{pmatrix}.$$

The eigenvalues of the above matrix are $\lambda_1 = n$ and $\lambda_2 = 3n^2 - 2n$, implying $N_{\max} = 3n^2 - 2n$. Thus, the expanding compiler instantiated by our ISW-based gadgets has a complexity blowup $\mathcal{O}(\kappa^e)$ with exponent

$$e = \frac{\log(3n^2 - 2n)}{\log(\lfloor (n+1)/4 \rfloor)}.$$

Figure 2 (blue curve) shows the evolution of the value of this exponent with respect to the number of shares n (where we assume an odd n). The value of e clearly decreases as the number of shares grows, and this decrease is faster for a small number of shares ($5 \leq n \leq 10$). The exponent value reaches $e \approx 4$ for a number of shares around 25 and then slowly converges towards $e = 2$ as n grows. This is to be compared with the $\mathcal{O}(\kappa^{7.5})$ complexity achieved by the instantiation from [2, 9].

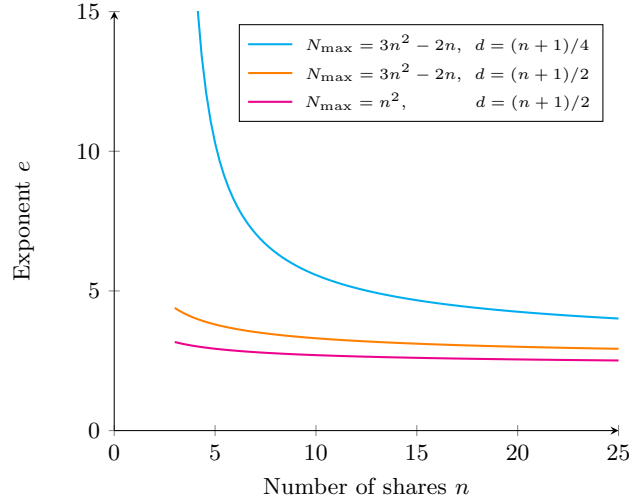


Fig. 2: Evolution of the complexity exponent $e = \log(N_{\max})/\log(d)$ with respect to the number of shares n . The blue curve matches the instantiation with the ISW-based gadgets; the orange curve assumes the optimal amplification order (*i.e.* an improvement of the multiplication gadget); the pink curve assumes a better complexity for addition and copy gadgets (so that $N_{\max}$ matches $N_{m,m} = n^2$).

Towards a Better Complexity. Choosing gadgets which attain the upper bound $\min(t+1, n-t)$ on the amplification order from Lemma 1 allows the compiler to have the maximum amplification order $d = (n+1)/2$ and thus have the lowest complexity blowup. Our ISW-based copy and addition gadgets achieve this bound while the ISW multiplication gadget is limited to $(n+1)/4$ (Lemma 12). To reach the optimal amplification order, one would need a different multiplication gadget and in

particular a multiplication gadget which does not perform a direct product of shares (because of the bound from Lemma 2). We introduce such a multiplication gadget hereafter (see Section 5.5). Specifically, our new multiplication gadget achieves the upper bound on the amplification order $\min(t+1, n-t)$ by avoiding a direct product of shares using a prior refresh on the input sharings. The orange curve in Figure 2 shows the evolution of the value of the exponent when instantiating the expanding compiler with our previous addition and copy gadgets and this new multiplication gadget. For such an instantiation, the complexity exponent still slowly converges towards $e = 2$ but, as we can see from Figure 2, the exponent value is much better for small values of n . For example, we obtain $e \approx 3$ for $n = 20$.

Another possible direction for improvement would be to lower the complexity of the addition and copy gadgets, which is mainly dominated by the refreshing. Assume that we can design a (T)RPE refresh gadget in sub-quadratic complexity, *e.g.* as the refresh gadgets proposed in [20, 7, 15], then the eigenvalues of the matrix in (5) would also be sub-quadratic and the value of $N_{\max}$ from equation (5) would drop to $N_{\max} = n^2$ (if the multiplication gadget still requires n^2 multiplication gates). The pink curve in Figure 2 depicts the evolution of the exponent value under this assumption. We still have a slow convergence towards $e = 2$ but the exponent value is yet better for small values of n . For example, a complexity blowup of $\mathcal{O}(\kappa^{2.5})$ is obtained with 20 shares. We leave the task of finding such a sub-quadratic (T)RPE refresh gadget as an open question for further research.

The above analysis shows that the expanding compiler can theoretically approach a quadratic complexity at the cost of increasing the number of shares in the base gadgets. The downside of it is that the tolerated leakage probability is likely to decrease as the number of shares grow. For instance, the ISW construction is known to only tolerate a leakage probability $p = \mathcal{O}(1/n)$ [14]. The number of shares hence offers multiple trade-offs between the tolerated probability and the asymptotic complexity of the compiler. Starting from a target leakage probability p , one could determine the highest number of shares admissible from a generic construction (such as the ISW-based instantiation exhibited above) and thus deduce the best complexity exponent achievable. In Section 6, we exhibit concrete trade-offs that can be reached for small values of n .

5.5 Multiplication Gadget with Maximal Amplification Order

Constructing a multiplication gadget which achieves the upper bound on the amplification order from Lemma 1 is tricky. First, as a standard multiplication gadget (*i.e.* which computes the cross products of the input shares), the ISW multiplication cannot achieve the maximal amplification order (see Lemma 2). In order to reach the upper bound for two-input gadgets (see Corollary 2), we need a non-standard multiplication gadget, *i.e.* which does not perform a direct product between the input shares. As an additional observation, the addition, copy, and random gates are *virtually free* in a multiplication gadget since they do not impact the final complexity of the expanding compiler (see Section 2.5). This suggests that we can be greedy in terms of randomness to reach the maximal amplification order.

In the following, we will describe the construction of a new multiplication gadget which achieves the maximum amplification order $\min(t+1, n-t)$. We first describe our standard n -share multiplication gadget and then explain how we avoid the initial cross products of shares. First, the gadget

constructs the matrix of the cross product of input shares:

$$M = \begin{pmatrix} a_1 \cdot b_1 & a_1 \cdot b_2 & \cdots & a_1 \cdot b_n \\ a_2 \cdot b_1 & a_2 \cdot b_2 & \cdots & a_2 \cdot b_n \\ \vdots & \vdots & \ddots & \vdots \\ a_n \cdot b_1 & a_n \cdot b_2 & \cdots & a_n \cdot b_n \end{pmatrix}$$

Then, it picks n^2 random values which define the following matrix:

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \cdots & r_{1,n} \\ r_{2,1} & r_{2,2} & \cdots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n,1} & r_{n,2} & \cdots & r_{n,n} \end{pmatrix}$$

It then performs an element-wise addition between the matrices M and R :

$$P = M + R = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2} & \cdots & p_{n,n} \end{pmatrix}$$

At this point, the gadget randomizes each product of input shares from the matrix M with a single random value from R . In order to generate the correct output, the gadget adds all the columns of P into a single column V of n elements, and adds all the columns of the transpose matrix $R^\top$ into a single column X of n elements:

$$V = \begin{pmatrix} p_{1,1} + \cdots + p_{1,n} \\ p_{2,1} + \cdots + p_{2,n} \\ \vdots \\ p_{n,1} + \cdots + p_{n,n} \end{pmatrix}, \quad X = \begin{pmatrix} r_{1,1} + \cdots + r_{n,1} \\ r_{1,2} + \cdots + r_{n,2} \\ \vdots \\ r_{1,n} + \cdots + r_{n,n} \end{pmatrix}$$

The n -share output is finally defined as $(c_1, \dots, c_n) = V + X$.

In order to further increase the maximum amplification order attainable by the gadget, we need to avoid performing a direct product of shares (because of the bound proved in Lemma 2). For this, we add a pre-processing phase to the gadget using a refresh gadget G_{refresh} . Specifically, we refresh the input $(b_1, \dots, b_n)$ each time it is used. In other terms, each row of the matrix M uses a fresh copy of $(b_1, \dots, b_n)$ produced using the considered refresh gadget. This amounts to performing n independent refreshes of the input $(b_1, \dots, b_n)$. The matrix M is thus defined as

$$M = \begin{pmatrix} a_1 \cdot b_1^{(1)} & a_1 \cdot b_2^{(1)} & \cdots & a_1 \cdot b_n^{(1)} \\ a_2 \cdot b_1^{(2)} & a_2 \cdot b_2^{(2)} & \cdots & a_2 \cdot b_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ a_n \cdot b_1^{(n)} & a_n \cdot b_2^{(n)} & \cdots & a_n \cdot b_n^{(n)} \end{pmatrix}$$

where $(b_1^{(j)}, \dots, b_n^{(j)})$, $j \in [n]$, are the n independent refreshings of the input $(b_1, \dots, b_n)$.

With this refreshing scheme, we avoid using the same share more than once for one of the two input sharings. As a consequence, the double failure set of size $t + 1$ which is the reason behind the bound $(t + 1)/2$ in Lemma 2, becomes a simple failure set (*i.e.* provoking a failure on a single input sharing). In addition, the computational overhead of these additional n refreshes is negligible compared to the joint contribution of the copy and addition gadgets to the complexity of the expanding compiler.

For the sake of completeness, we present the full algorithm for this multiplication gadget in Algorithm 5.

Algorithm 5: Our multiplication gadget

Input : $(a_1, \dots, a_n), (b_1, \dots, b_n)$ input sharings, $\{r_{ij}\}_{1 \leq i \leq n, 1 \leq j \leq n}$ random values, refresh gadget G_{refresh}

Output: $(c_1, \dots, c_n)$ sharing of $a \cdot b$

```

for  $i \leftarrow 1$  to  $n$  do
   $(b_1^{(i)}, \dots, b_n^{(i)}) \leftarrow G_{\text{refresh}}(b_1, \dots, b_n);$ 
end
for  $i \leftarrow 1$  to  $n$  do
  for  $j \leftarrow 1$  to  $n$  do
     $p_{i,j} \leftarrow a_i \times b_j^{(i)} + r_{i,j};$ 
  end
end
 $(v_1, \dots, v_n) \leftarrow (0, \dots, 0);$ 
 $(x_1, \dots, x_n) \leftarrow (0, \dots, 0);$ 
for  $i \leftarrow 1$  to  $n$  do
  for  $j \leftarrow 1$  to  $n$  do
     $v_i \leftarrow v_i + p_{i,j};$ 
     $x_i \leftarrow x_i + r_{i,j};$ 
  end
end
for  $i \leftarrow 1$  to  $n$  do
   $c_i \leftarrow v_i + x_i;$ 
end
return  $(c_1, \dots, c_n);$ 

```

In the following lemma, we show that if the refresh gadget G_{refresh} achieves the TRPE1 property with the amplification order at least $d = \min(t + 1, n - t)$ for any t , then the multiplication gadget depicted in Algorithm 5 achieves TRPE with the maximum amplification orders. The proof is given in Appendix I.

Lemma 13. *Let $t \leq n - 1$. Let G_{refresh} be a (t, f') -TRPE1 refresh gadget for some function $f' : \mathbb{R} \rightarrow \mathbb{R}$, and G_{mult} the n -share multiplication gadget from Algorithm 5. If f' is of amplification order $d' \geq d = \min(t + 1, n - t)$, then G_{mult} achieves (t, f) -TRPE for some function $f : \mathbb{R} \rightarrow \mathbb{R}$ of amplification order $d = \min(t + 1, n - t)$.*

Corollary 6 then directly follows from Lemma 13 by applying Lemma 4 (TRPE $\Rightarrow$ RPE).

Corollary 6. *Let $t \leq n - 1$. Let G_{refresh} be a (t, f') -TRPE1 refresh gadget for some function $f' : \mathbb{R} \rightarrow \mathbb{R}$, and G_{mult} the n -share multiplication gadget from Algorithm 5. If f' is of amplification order $d' \geq d = \min(t + 1, n - t)$, then G_{mult} achieves (t, f) -RPE for some function $f : \mathbb{R} \rightarrow \mathbb{R}$ of the same amplification order $d = \min(t + 1, n - t)$.*

6 Efficient Small Gadgets

This section displays our new constructions of small gadgets for copy, addition, and multiplication operations with a low number of shares. As explained in [9], we cannot achieve RPE security with relevant amplification orders for gadgets of less than 3 shares. Then, as explained in Section 3.1, the highest amplification orders can only be achieved for gadgets with an odd number of shares. We therefore omit 4-share gadgets and display our best trade-offs in terms of RPE security and complexity for 3-share and 5-share gadgets. Each one of these gadgets is experimentally verified using the VRAPS verification tool from [9].

Addition and Copy Gadgets. For the construction of small 3-share and 5-share addition and copy gadgets, we use the generic constructions depicted in Algorithms 1 and 2 (in Section 5) which naturally use a refresh gadget as a building block. We hence start by looking for refresh gadgets that have a good complexity in terms of gates count, and achieve the upper bound on the amplification order for the specific case of 3-share and 5-share constructions (but not necessarily for a higher number of shares).

Multiplication gadget. For the construction of small 3-share and 5-share multiplication gadgets, we use the generic construction depicted in Algorithm 5 from Section 5.5 which, to the best of our knowledge, is the only multiplication gadget which achieves the maximum amplification order for any number of shares, and specifically for 3-share and 5-share constructions. As for the refresh gadget G_{refresh} which is used to perform n refreshes on the second input, we use the same scheme as for the construction of small addition and copy gadgets (and which shall satisfy the necessary condition on G_{refresh} from Corollary 6).

While the multiplication gadget from Section 5.5 achieves the desired amplification order, we add another pre-processing phase to the gadget in order to further improve the tolerated leakage probability. In addition to the n refreshes performed on the second input b (see Algorithm 5), we add another single refresh of the input $(a_1, \dots, a_n)$ before computing the cross-products, using the same refresh gadget G_{refresh} . Refreshing the input $(a_1, \dots, a_n)$ before usage experimentally shows a further increase in the maximum tolerated leakage probability, by adding more randomness to the input shares before computing the cross-product matrix M in Algorithm 5. And since the refresh gadget G_{refresh} achieves the maximum amplification order, the amplification order achieved by G_{mult} is not affected by adding another refresh to the first input a .

The above construction achieves the maximum amplification order for 3-share ($d = 2$) and 5-share ($d = 3$) gadgets based on natural refresh gadgets detailed hereafter.

6.1 3-share Gadgets

We start with the construction of 3-share gadgets for our three base operations.

Copy and Addition Gadgets. We build our copy and addition gadgets from the instantiation of the generic constructions of Section 5 (Algorithms 1 and 2) with 3 shares. However, we do not use the ISW refresh gadget but the following more efficient construction with only two random values (instead of three):

$$\begin{aligned} G_{\text{refresh}} : c_1 &\leftarrow r_1 + a_1 \\ c_2 &\leftarrow r_2 + a_2 \\ c_3 &\leftarrow (r_1 + r_2) + a_3. \end{aligned}$$

This refresh is sufficient to reach the upper bounds on the amplification orders (from Lemma 1). From this basis, we obtain the following 3-share addition gadget with four random values:

$$\begin{aligned} G_{\text{add}} : c_1 &\leftarrow (r_1 + a_1) + (r_3 + b_1) \\ c_2 &\leftarrow (r_2 + a_2) + (r_4 + b_2) \\ c_3 &\leftarrow ((r_1 + r_2) + a_3) + ((r_3 + r_4) + b_3) \end{aligned}$$

and the following 3-share copy gadget with also four random values:

$$\begin{aligned} G_{\text{copy}} : c_1 &\leftarrow r_1 + a_1; & d_1 &\leftarrow r_3 + a_1 \\ c_2 &\leftarrow r_2 + a_2; & d_2 &\leftarrow r_4 + a_2 \\ c_3 &\leftarrow (r_1 + r_2) + a_3; & d_3 &\leftarrow (r_3 + r_4) + a_3. \end{aligned}$$

Multiplication Gadget. The following construction is a 3-share instantiation of the multiplication gadget described in Section 5.5. For the input refreshing, we use the 3-share refresh gadget described above with two uniformly random values. The construction achieves the bound on the amplification order from Lemma 1 with 17 random values:

$$\begin{aligned} G_{\text{mult}} : i_{1,1} &\leftarrow r_1 + b_1; & i_{1,2} &\leftarrow r_2 + b_2; & i_{1,3} &\leftarrow (r_1 + r_2) + b_3 \\ i_{2,1} &\leftarrow r_3 + b_1; & i_{2,2} &\leftarrow r_4 + b_2; & i_{2,3} &\leftarrow (r_3 + r_4) + b_3 \\ i_{3,1} &\leftarrow r_5 + b_1; & i_{3,2} &\leftarrow r_6 + b_2; & i_{3,3} &\leftarrow (r_5 + r_6) + b_3 \\ a'_1 &\leftarrow r_7 + a_1; & a'_2 &\leftarrow r_8 + a_2; & a'_3 &\leftarrow (r_7 + r_8) + a_3 \\ c_1 &\leftarrow (a'_1 \cdot i_{1,1} + r_{1,1}) + (a'_1 \cdot i_{1,2} + r_{1,2}) + (a'_1 \cdot i_{1,3} + r_{1,3}) + (r_{1,1} + r_{2,1} + r_{3,1}) \\ c_2 &\leftarrow (a'_2 \cdot i_{2,1} + r_{2,1}) + (a'_2 \cdot i_{2,2} + r_{2,2}) + (a'_2 \cdot i_{2,3} + r_{2,3}) + (r_{1,2} + r_{2,2} + r_{3,2}) \\ c_3 &\leftarrow (a'_3 \cdot i_{3,1} + r_{3,1}) + (a'_3 \cdot i_{3,2} + r_{3,2}) + (a'_3 \cdot i_{3,3} + r_{3,3}) + (r_{1,3} + r_{2,3} + r_{3,3}). \end{aligned}$$

Results. Table 1 displays the results for the above gadgets obtained through the VRAPS tool. The second column gives the complexity, where N_a , N_c , N_m , N_r stand for the number of addition gates, copy gates, multiplication gates and random gates respectively. The third column provides the amplification order of the gadget. And the last column gives the maximum tolerated leakage probability. The last row gives the global complexity, amplification order, and maximum tolerated leakage probability for the expanding compiler using these three gadgets from the results provided in [9].

Table 1: Results for the 3-share gadgets for $(t = 1, f)$ -RPE, achieving the bound on the amplification order.

Gadget	Complexity (N_a, N_c, N_m, N_r)	Amplification order	$\log_2$ of maximum tolerated proba
G_{refresh}	(4, 2, 0, 2)	2	-5.14
G_{add}	(11, 4, 0, 4)	2	-4.75
G_{copy}	(8, 7, 0, 4)	2	-7.50
G_{mult}	(40, 29, 9, 17)	2	-7.41
Compiler	$\mathcal{O}(C \cdot \kappa^{3.9})$	2	-7.50

Remark 1. The copy gadget G_{copy} instantiated in [9] which uses a refresh scheme with 3 randoms for each output, also reaches the amplification order 2. It tolerates a better leakage probability (*i.e.* $2^{-5.9}$) than the one provided here, but with a higher complexity of (12, 9, 0, 6). If it is used to replace the 3-share copy gadget, the maximum tolerated leakage probability by the compiler from Table 1 would be of $2^{-7.4}$ slightly better than the current value of $2^{-7.5}$ but with a higher complexity of $\mathcal{O}(|C| \cdot \kappa^{4.08})$ instead of $\mathcal{O}(|C| \cdot \kappa^{3.9})$. Another copy gadget can be constructed by using the refresh scheme with 3 random values from [9] for one of the outputs, and the refresh scheme presented in this section with 2 random values for the second output. This gadget tolerates a maximum leakage probability of around $2^{-7.1}$ with a complexity of (10, 8, 0, 5). Using it would bring the complexity of the compiler from Table 1 to $\mathcal{O}(|C| \cdot \kappa^4)$, while tolerating a leakage probability of $2^{-7.4}$, the same as that of the used multiplication gadget.

6.2 5-share Gadgets

We now present our 5-share gadgets for our three base operations, which reach the optimal amplification order from Lemma 1.

Copy and Addition Gadgets. As for the 3-share case, we use the generic constructions from Section 5. Instead of using the ISW refresh gadget which would require 10 uniformly random values for a 5-share construction, we use the *circular refresh gadget* described in [4, 6] (a.k.a. *block refresh gadget*):

$$\begin{aligned}
G_{\text{refresh}} : c_1 &\leftarrow (r_1 + r_2) + a_1 \\
c_2 &\leftarrow (r_2 + r_3) + a_2 \\
c_3 &\leftarrow (r_3 + r_4) + a_3 \\
c_4 &\leftarrow (r_4 + r_5) + a_4 \\
c_5 &\leftarrow (r_5 + r_1) + a_5.
\end{aligned}$$

This gadget only uses n randoms for an n -share construction, and while it does not achieve enough security in the generic case (unless the refresh block is iterated on the input a certain number of times [4, 6]), it proves to be more than enough to achieve the necessary amplification order for our

5-share constructions. We use a variant of the original version (also suggested in [4]): we choose to sum the random values first (thus obtaining a sharing of 0) before adding them to the input shares. The idea is to avoid using the input shares in any of the intermediate variables, so that input shares only appear in the input variables $\{a_i\}_{1 \leq i \leq n}$ and the final output variables $\{c_i\}_{1 \leq i \leq n}$. Intuitively, this trick allows to have less failure tuples in the gadget because there are less variables that could leak information about the input. This is validated experimentally where we obtain better results in terms of amplification order and tolerated leakage probability for small gadgets.

From this circular refresh, we obtain an addition gadget with ten random values which reaches the upper bound on the amplification order:

$$\begin{aligned}
G_{\text{add}} : c_1 &\leftarrow ((r_1 + r_2) + a_1) + ((r_6 + r_7) + b_1) \\
c_2 &\leftarrow ((r_2 + r_3) + a_2) + ((r_7 + r_8) + b_2) \\
c_3 &\leftarrow ((r_3 + r_4) + a_3) + ((r_8 + r_9) + b_3) \\
c_4 &\leftarrow ((r_4 + r_5) + a_4) + ((r_9 + r_{10}) + b_4) \\
c_5 &\leftarrow ((r_5 + r_1) + a_5) + ((r_{10} + r_6) + b_5)
\end{aligned}$$

and a copy gadget with also ten random values and which also reaches the upper bound on the amplification order:

$$\begin{aligned}
G_{\text{copy}} : c_1 &\leftarrow (r_1 + r_2) + a_1; & d_1 &\leftarrow (r_6 + r_7) + a_1 \\
c_2 &\leftarrow (r_2 + r_3) + a_2; & d_2 &\leftarrow (r_7 + r_8) + a_2 \\
c_3 &\leftarrow (r_3 + r_4) + a_3; & d_3 &\leftarrow (r_8 + r_9) + a_3 \\
c_4 &\leftarrow (r_4 + r_5) + a_4; & d_4 &\leftarrow (r_9 + r_{10}) + a_4 \\
c_5 &\leftarrow (r_5 + r_1) + a_5; & d_5 &\leftarrow (r_{10} + r_6) + a_5.
\end{aligned}$$

Multiplication Gadget. The following construction is a 5-share instantiation of the multiplication gadget described in Section 5.5. For the input refreshing, we use the 5-share circular refresh gadget described above. The gadget advantageously achieves the optimal amplification order (given by

Lemma 1) with 55 random values:

$$G_{\text{mult}} : i_{1,1} \leftarrow (r_1 + r_2) + b_1; \quad i_{1,2} \leftarrow (r_2 + r_3) + b_2; \quad i_{1,3} \leftarrow (r_3 + r_4) + b_3; \\ i_{1,4} \leftarrow (r_4 + r_5) + b_4; \quad i_{1,5} \leftarrow (r_5 + r_1) + b_5$$

$$i_{2,1} \leftarrow (r_6 + r_7) + b_1; \quad i_{2,2} \leftarrow (r_7 + r_8) + b_2; \quad i_{2,3} \leftarrow (r_8 + r_9) + b_3; \\ i_{2,4} \leftarrow (r_9 + r_{10}) + b_4; \quad i_{2,5} \leftarrow (r_{10} + r_6) + b_5$$

$$i_{3,1} \leftarrow (r_{11} + r_{12}) + b_1; \quad i_{3,2} \leftarrow (r_{12} + r_{13}) + b_2; \quad i_{3,3} \leftarrow (r_{13} + r_{14}) + b_3; \\ i_{3,4} \leftarrow (r_{14} + r_{15}) + b_4; \quad i_{3,5} \leftarrow (r_{15} + r_{11}) + b_5$$

$$i_{4,1} \leftarrow (r_{16} + r_{17}) + b_1; \quad i_{4,2} \leftarrow (r_{17} + r_{18}) + b_2; \quad i_{4,3} \leftarrow (r_{18} + r_{19}) + b_3; \\ i_{4,4} \leftarrow (r_{19} + r_{20}) + b_4; \quad i_{4,5} \leftarrow (r_{20} + r_{16}) + b_5$$

$$i_{5,1} \leftarrow (r_{21} + r_{22}) + b_1; \quad i_{5,2} \leftarrow (r_{22} + r_{23}) + b_2; \quad i_{5,3} \leftarrow (r_{23} + r_{24}) + b_3; \\ i_{5,4} \leftarrow (r_{24} + r_{25}) + b_4; \quad i_{5,5} \leftarrow (r_{25} + r_{21}) + b_5$$

$$a'_1 \leftarrow (r_{26} + r_{27}) + a_1; \quad a'_2 \leftarrow (r_{27} + r_{28}) + a_2; \quad a'_3 \leftarrow (r_{28} + r_{29}) + a_3; \\ a'_4 \leftarrow (r_{29} + r_{30}) + a_4; \quad a'_5 \leftarrow (r_{30} + r_{26}) + a_5$$

$$c_1 \leftarrow (a'_1 \cdot i_{1,1} + r_{1,1}) + (a'_1 \cdot i_{1,2} + r_{1,2}) + (a'_1 \cdot i_{1,3} + r_{1,3}) + (a'_1 \cdot i_{1,4} + r_{1,4}) \\ + (a'_1 \cdot i_{1,5} + r_{1,5}) + (r_{1,1} + r_{2,1} + r_{3,1} + r_{4,1} + r_{5,1}) \\ c_2 \leftarrow (a'_2 \cdot i_{2,1} + r_{2,1}) + (a'_2 \cdot i_{2,2} + r_{2,2}) + (a'_2 \cdot i_{2,3} + r_{2,3}) + (a'_2 \cdot i_{2,4} + r_{2,4}) \\ + (a'_2 \cdot i_{2,5} + r_{2,5}) + (r_{1,2} + r_{2,2} + r_{3,2} + r_{4,2} + r_{5,2}) \\ c_3 \leftarrow (a'_3 \cdot i_{3,1} + r_{3,1}) + (a'_3 \cdot i_{3,2} + r_{3,2}) + (a'_3 \cdot i_{3,3} + r_{3,3}) + (a'_3 \cdot i_{3,4} + r_{3,4}) \\ + (a'_3 \cdot i_{3,5} + r_{3,5}) + (r_{1,3} + r_{2,3} + r_{3,3} + r_{4,3} + r_{5,3}) \\ c_4 \leftarrow (a'_4 \cdot i_{4,1} + r_{4,1}) + (a'_4 \cdot i_{4,2} + r_{4,2}) + (a'_4 \cdot i_{4,3} + r_{4,3}) + (a'_4 \cdot i_{4,4} + r_{4,4}) \\ + (a'_4 \cdot i_{4,5} + r_{4,5}) + (r_{1,4} + r_{2,4} + r_{3,4} + r_{4,4} + r_{5,4}) \\ c_5 \leftarrow (a'_5 \cdot i_{5,1} + r_{5,1}) + (a'_5 \cdot i_{5,2} + r_{5,2}) + (a'_5 \cdot i_{5,3} + r_{5,3}) + (a'_5 \cdot i_{5,4} + r_{5,4}) \\ + (a'_5 \cdot i_{5,5} + r_{5,5}) + (r_{1,5} + r_{2,5} + r_{3,5} + r_{4,5} + r_{5,5}).$$

Results. Table 2 gives the results for the above gadgets obtained through the VRAPS tool.

Table 2: Results for the 5-share gadgets for $(t = 2, f)$ -RPE, achieving the bound on the amplification order.

Gadget	Complexity	Amplification order	$\log_2$ of maximum tolerated proba
G_{refresh}	(10, 5, 0, 5)	3	-4.83
G_{add}	(25, 10, 0, 10)	3	$[-6.43, -3.79]$
G_{copy}	(20, 15, 0, 10)	3	$[-6.43, -5.78]$
G_{mult}	(130, 95, 25, 55)	3	$[-12.00, -6.03]$
Compiler	$\mathcal{O}(C \cdot \kappa^{3.23})$	3	$[-12.00, -6.03]$

From Tables 1 and 2, we observe that the asymptotic complexity is better for the instantiation based on 5-share gadgets as they provide a better amplification order with limited overhead. While this result can seem to be counterintuitive, it actually comes from the fact that each gadget will be expended less in the second scenario. We stress that we could only obtain an interval $[2^{-12}, 2^{-6}]$ for the tolerated leakage probability because it was computationally too expensive to obtain a tighter interval from the **VRAPS** tool, but this could probably be improved in the future. Meanwhile, we can consider that our best complexity $\mathcal{O}(|C| \cdot \kappa^{3.2})$ comes at the price of a lower tolerated leakage probability of 2^{-12} (5-share gadget) compared to the $\mathcal{O}(|C| \cdot \kappa^{3.9})$ complexity and $2^{-7.5}$ tolerated leakage probability obtained for our 3-share instantiation.

In comparison, the previous instantiation of the expanding compiler [9] could only achieve a complexity of $\mathcal{O}(|C| \cdot \kappa^{7.5})$ for maximum tolerated probabilities of 2^{-8} , and the instantiation of the expanding approach with a multi-party computation protocol [2], could only achieve a complexity of $\mathcal{O}(|C| \cdot \kappa^{8.2})$ for maximum tolerated probabilities of 2^{-26} .

Acknowledgments. This work is partly supported by the French FUI-AAP25 VeriSiCC project.

References

1. Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 715–724, San Jose, CA, USA, June 6–8, 2011. ACM Press.
2. Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Heidelberg, Germany.
3. Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.
4. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standardt, and Pierre-Yves Strub. Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. *Journal of Cryptographic Engineering*, 10(1):17–26, April 2020.
5. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl,

- Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016: 23rd Conference on Computer and Communications Security*, pages 116–129, Vienna, Austria, October 24–28, 2016. ACM Press.
6. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, Paris, France, April 30 – May 4, 2017. Springer, Heidelberg, Germany.
 7. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. Cryptology ePrint Archive, Report 2016/540, 2016. <https://eprint.iacr.org/2016/540>.
 8. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648, Vienna, Austria, May 8–12, 2016. Springer, Heidelberg, Germany.
 9. Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 339–368, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Heidelberg, Germany.
 10. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO’99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer, Heidelberg, Germany.
 11. Jean-Sébastien Coron. Higher order masking of look-up tables. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 441–458, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.
 12. Jean-Sébastien Coron, Aurélien Greuet, and Rina Zeitoun. Side-channel masking with pseudo-random generator. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology – EUROCRYPT 2020, Part III*, volume 12107 of *Lecture Notes in Computer Science*, pages 342–375, Zagreb, Croatia, May 10–14, 2020. Springer, Heidelberg, Germany.
 13. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424, Singapore, March 11–13, 2014. Springer, Heidelberg, Germany.
 14. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer, Heidelberg, Germany.
 15. Stefan Dziembowski, Sebastian Faust, and Karol Zebrowski. Simple refreshing in the noisy leakage model. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 315–344, Kobe, Japan, December 8–12, 2019. Springer, Heidelberg, Germany.
 16. Louis Goubin and Jacques Patarin. DES and differential power analysis (the “duplication” method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES’99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer, Heidelberg, Germany.
 17. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer, Heidelberg, Germany.
 18. Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO’96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer, Heidelberg, Germany.
 19. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159, Athens, Greece, May 26–30, 2013. Springer, Heidelberg, Germany.
 20. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225

of *Lecture Notes in Computer Science*, pages 413–427, Santa Barbara, CA, USA, August 17–20, 2010. Springer, Heidelberg, Germany.

A Random Probing Expandability 1 & 2

Definition 7 (Random Probing Expandability 1). Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -RPE1 if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$, such that $|J| \leq t$, and for every $p \in [0, 1]$, the random experiment

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2) &\leftarrow \text{Sim}_1^G(\mathcal{W}, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J, \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (|I_1| > t)$ and $\mathcal{F}_2 \equiv (|I_2| > t)$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \quad (8)$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. the output distribution satisfies

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, (\hat{x}, \hat{y})), \hat{z}|_J) \quad (9)$$

where $\hat{z} = G(\hat{x}, \hat{y})$.

Definition 8 (Random Probing Expandability 2). Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -RPE2 if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every $p \in [0, 1]$, the random experiment

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2, J) &\leftarrow \text{Sim}_1^G(\mathcal{W}) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J, \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (|I_1| > t)$ and $\mathcal{F}_2 \equiv (|I_2| > t)$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \quad (10)$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. J is such that $J \subseteq [n]$ with $|J| = n - 1$

3. the output distribution satisfies

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, (\hat{x}, \hat{y})), \hat{z}|_J) \quad (11)$$

where $\hat{z} = G(\hat{x}, \hat{y})$.

B Tight Random Probing Expandability

Definition 9 (Tight Random Probing Expandability). Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -tight random probing expandable (TRPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the random experiment

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2, J') &\leftarrow \text{Sim}_1^G(\mathcal{W}, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J', \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (|I_1| > \min(t, |\mathcal{W}|))$ and $\mathcal{F}_2 \equiv (|I_2| > \min(t, |\mathcal{W}|))$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \quad (12)$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. J' is such that $J' = J$ if $|J| \leq t$ and $J' \subseteq [n]$ with $|J'| = n - 1$ otherwise,
3. the output distribution satisfies

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, (\hat{x}, \hat{y})), \hat{z}|_{J'}) \quad (13)$$

where $\hat{z} = G(\hat{x}, \hat{y})$,

Definition 10 (Tight Random Probing Expandability 1). Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -tight random probing expandable (TRPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$, such that $|J| \leq t$, and for every $p \in [0, 1]$, the random experiment

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2) &\leftarrow \text{Sim}_1^G(\mathcal{W}, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J, \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (|I_1| > \min(t, |\mathcal{W}|))$ and $\mathcal{F}_2 \equiv (|I_2| > \min(t, |\mathcal{W}|))$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \quad (14)$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. the output distribution satisfies

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, (\hat{x}, \hat{y})), \hat{z}|_J) \quad (15)$$

where $\hat{z} = G(\hat{x}, \hat{y})$,

Definition 11 (Tight Random Probing Expandability 2). Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -tight random probing expandable (TRPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every $p \in [0, 1]$, the random experiment

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2, J) &\leftarrow \text{Sim}_1^G(\mathcal{W}) \\ \text{out} &\leftarrow \text{Sim}_2^G(\mathcal{W}, J, \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (|I_1| > \min(t, |\mathcal{W}|))$ and $\mathcal{F}_2 \equiv (|I_2| > \min(t, |\mathcal{W}|))$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \quad (16)$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. J is such that $J \subseteq [n]$ with $|J| = n - 1$
3. the output distribution satisfies

$$\text{out} \stackrel{\text{id}}{=} (\text{AssignWires}(G, \mathcal{W}, (\hat{x}, \hat{y})), \hat{z}|_J) \quad (17)$$

where $\hat{z} = G(\hat{x}, \hat{y})$,

C Proof of Lemma 7

Proof. To prove that G_{copy} is TRPE achieving the same amplification order d as the underlying refresh gadget G_{refresh} , we need to prove that any set of leaking wires $\mathcal{W}$ such that $|\mathcal{W}| \leq d - 1$ can be perfectly simulated together with any sets of outputs wires $J_1, J_2 \subseteq [n]$ (such that J_1 refers to the first output e and J_2 to the second output f) from a set of input wires I such that $|I| \leq \min(t, |\mathcal{W}|)$. In addition, we know from Lemma 1 that the maximal amplification order achievable in the TRPE setting is $d_{\text{max}} \leq \min(t + 1, 2(n - t))$. Since we consider sets $\mathcal{W}$ of size at most $|\mathcal{W}| \leq d - 1 \leq \min(t + 1, 2(n - t)) - 1 \leq t$ then we need to prove that $|I| \leq \min(t, |\mathcal{W}|) = |\mathcal{W}|$.

The leaking set $\mathcal{W}$ can be split into two distinct subsets $\mathcal{W}_1$ and $\mathcal{W}_2$ such that $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2$ where $\mathcal{W}_1$ (resp. $\mathcal{W}_2$) is the set of leaking wires of G_{refresh} for the output e (resp. f). Let $J_1, J_2 \subseteq [n]$. We consider four cases:

- $|J_1| \leq t, |J_2| \leq t$: since $|\mathcal{W}| \leq d - 1$, then $|\mathcal{W}_1|, |\mathcal{W}_2| \leq d - 1$. Since G_{refresh} achieves an amplification order d , then by definition of TRPE, the sets $\mathcal{W}_1$ and J_1 can be simulated with a set of input shares I_1 such that $|I_1| \leq \min(|\mathcal{W}_1|, t)$. Similarly, the sets $\mathcal{W}_2$ and J_2 can be simulated with a set of input shares I_2 such that $|I_2| \leq \min(|\mathcal{W}_2|, t)$. As a consequence, set I defined as $I = I_1 \cup I_2$ is enough to simulate $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2$ and both output shares J_1 and J_2 . Furthermore, we have

$$|I| \leq |I_1| + |I_2| \leq \min(|\mathcal{W}_1|, t) + \min(|\mathcal{W}_2|, t) \leq |\mathcal{W}| = \min(|\mathcal{W}|, t)$$

- $|J_1| > t, |J_2| > t$: in this case, we need to prove the existence of a set of input shares I such that $|I| \leq \min(t, |\mathcal{W}|) = |\mathcal{W}|$ (since $|\mathcal{W}| \leq d - 1 \leq t$) for which we can perfectly simulate $\mathcal{W}$ together with two chosen output sets J'_1 and J'_2 such that $|J'_1| = |J'_2| = n - 1$. Since we have $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2$ such that $|\mathcal{W}_1| \leq d - 1, |\mathcal{W}_2| \leq d - 1$, then by definition of TRPE, there exists $J'_1, |J'_1| = n - 1$ such that $\mathcal{W}_1$ and J'_1 can be perfectly simulated from a set of inputs shares I_1 such that $|I_1| \leq \min(|\mathcal{W}_1|, t)$. Similarly, there exists $J'_2, |J'_2| = n - 1$ such that $\mathcal{W}_2$ and J'_2 can be perfectly simulated from a set of inputs shares I_2 such that $|I_2| \leq \min(|\mathcal{W}_2|, t)$. By choosing such sets J'_1, J'_2 , the overall simulation of G_{copy} can be done with the set of input shares $I = I_1 \cup I_2$, and we have

$$|I| \leq |I_1| + |I_2| \leq \min(|\mathcal{W}_1|, t) + \min(|\mathcal{W}_2|, t) \leq |\mathcal{W}| = \min(|\mathcal{W}|, t)$$

- $|J_1| \leq t, |J_2| > t$: Since $|J_1| \leq t$, by definition of TRPE, $\mathcal{W}_1$ and J_1 can be perfectly simulated from a set of input shares I_1 such that $|I_1| \leq \min(|\mathcal{W}_1|, t)$. In addition, for $|J_2| > t$, we also know that we can choose a set J'_2 such that $|J'_2| = n - 1$ that can be perfectly simulated with $\mathcal{W}_2$ from a set of input shares I_2 with $|I_2| \leq \min(|\mathcal{W}_2|, t)$. By choosing such a set J'_2 , the overall simulation of G_{copy} can be achieved with the set of input wires $I = I_1 \cup I_2$, and we have

$$|I| \leq |I_1| + |I_2| \leq \min(|\mathcal{W}_1|, t) + \min(|\mathcal{W}_2|, t) \leq |\mathcal{W}| = \min(|\mathcal{W}|, t)$$

- $|J_1| > t, |J_2| \leq t$: the proof is exactly the reflect of the previous one.

Since in the four cases, there is no failure tuple $\mathcal{W}$ of size $|\mathcal{W}| < d$, then the gadget G_{copy} achieves an amplification order d . Lemma 1 finally completes the proof. $\square$

D Proof of Lemma 8

Proof. We need to prove that when G_{refresh} is (t, f) -RPE (resp. (t, f') -TRPE) of amplification order d , then G_{add} is (t, f') -RPE (resp. (t, f') -TRPE) of amplification order at least $\lfloor \frac{d}{2} \rfloor$. We will prove the property for the RPE setting, and the proof for the TRPE setting will be exactly the same except for the notion of failure event which changes. This amounts to proving that:

1. Any set of leaking wires $\mathcal{W}$ such that $|\mathcal{W}| < \lfloor \frac{d}{2} \rfloor$ can be simulated together with any set of outputs wires $J \subseteq [n]$ from sets of input wires I_1 on a and I_2 on b such that $|I_1| \leq t$ and $|I_2| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |\mathcal{W}|)$ and $|I_2| \leq \min(t, |\mathcal{W}|)$).
2. Any set of leaking wires such that $\lfloor \frac{d}{2} \rfloor \leq |\mathcal{W}| < d$ can be simulated together with any set of outputs wires $J \subseteq [n]$ from sets of input wires I_1, I_2 such that $|I_1| \leq t$ or $|I_2| \leq t$ (because of the double failure, *i.e* failure on both inputs) (for TRPE we would have $|I_1| \leq \min(t, |\mathcal{W}|)$ or $|I_2| \leq \min(t, |\mathcal{W}|)$).

We proceed by building the necessary simulators for G_{add} from the simulators that already exist for G_{refresh} . Concretely, we split each set $\mathcal{W}$ of leaking wires, into four subsets $\mathcal{W} = \mathcal{W}_1^r \cup \mathcal{W}_1^a \cup \mathcal{W}_2^r \cup \mathcal{W}_2^a$ where $\mathcal{W}_1^r$ (resp. $\mathcal{W}_2^r$) is the set of leaking wires during the computation of $G_{\text{refresh}}(a_1, \dots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \dots, b_n)$), and $\mathcal{W}_1^a$ (resp. $\mathcal{W}_2^a$) is the set of leaking wires of $(e_1, \dots, e_n)$ (resp. $(f_1, \dots, f_n)$).

From these notations, we build a leaking set $\mathcal{W}'$ which contains $\mathcal{W}_1^r$ and $\mathcal{W}_2^r$ and also each input or pair of inputs of gates whose output is a wire in $\mathcal{W}_1^a$ or $\mathcal{W}_2^a$. We have that

$$|\mathcal{W}'| \leq |\mathcal{W}_1^r| + |\mathcal{W}_2^r| + 2|\mathcal{W}_1^a| + 2|\mathcal{W}_2^a| \leq 2|\mathcal{W}|.$$

The new set $\mathcal{W}'$ can be split into two subsets $\mathcal{W}'_1$ and $\mathcal{W}'_2$ such that $\mathcal{W}'_1$ (resp. $\mathcal{W}'_2$) contains only leaking wires during the computation of $G_{\text{refresh}}(a_1, \dots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \dots, b_n)$). We now demonstrate how we can simulate $\mathcal{W}'$ when the output set J is of size less than t ((T)RPE1) and when it is of size strictly more than t ((T)RPE2).

– if $|J| \leq t$ ((T)RPE1): we prove both properties 1 and 2:

1. we assume that $|\mathcal{W}| < \lfloor \frac{d}{2} \rfloor$. Then we consider the set $\mathcal{W}' = \mathcal{W}'_1 \cup \mathcal{W}'_2$ (as previously defined) such that

$$|\mathcal{W}'| \leq 2|\mathcal{W}| < 2\lfloor \frac{d}{2} \rfloor \leq d$$

and hence,

$$|\mathcal{W}'_1| < d \quad \text{and} \quad |\mathcal{W}'_2| < d.$$

From the (t, f) -RPE property of G_{refresh} and its amplification order, there exists an input set of shares of a I_1 such that $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |\mathcal{W}|)$) and I_1 perfectly simulates $\mathcal{W}'_1$ and any set J_1 of up to t variables e_i . Similarly, there exists an input set of shares of b I_2 such that $|I_2| \leq t$ (for TRPE we would have $|I_2| \leq \min(t, |\mathcal{W}|)$) and I_2 perfectly simulates $\mathcal{W}'_2$ and any set J_2 of up to t variables f_i . J_1 and J_2 are chosen as the inputs e_i and f_i respectively of wires $e_i + f_i$ in set J . Namely $|J_1| = |J_2| = |J|$.

From these definitions, I_1 and I_2 together perfectly simulate $\mathcal{W}'$ and J and are both of size less than t (less than $\min(t, |\mathcal{W}|)$ for TRPE), which proves the first property in this scenario.

2. we now assume that $\lfloor \frac{d}{2} \rfloor \leq |\mathcal{W}| < d$. Then we consider the set $\mathcal{W}' = \mathcal{W}'_1 \cup \mathcal{W}'_2$ such that

$$|\mathcal{W}'| \leq 2|\mathcal{W}| < 2d$$

and hence,

$$|\mathcal{W}'_1| < d \quad \text{or} \quad |\mathcal{W}'_2| < d.$$

Without loss of generality, let us consider that $|\mathcal{W}'_1| < d$ (the proof is similar in the opposite scenario). From the (t, f) -RPE property of G_{refresh} and its amplification order, there exists an input set of shares of a I_1 such that $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |\mathcal{W}|)$) and I_1 perfectly simulates $\mathcal{W}'_1$ and any set J_1 of up to t variables e_i . There also exists an input set of shares of b I_2 which perfectly simulates $\mathcal{W}'_2$ and any set J_2 of up to t variables f_i but not necessarily of size less than t (less than $\min(t, |\mathcal{W}|)$ for TRPE). If J_1 and J_2 are chosen as the inputs e_i and f_i respectively of wires $e_i + f_i$ in set J , then sets I_1 and I_2 together perfectly simulate $\mathcal{W}'$ and J . In this case, we only have a failure on at most one of the inputs (b in this case), which concludes the proof for the second property.

At this point, we proved that G_{add} achieves an amplification order greater than or equal to $\lfloor \frac{d}{2} \rfloor$ for RPE1 (for TRPE1 in the TRPE setting).

– if $|J| > t$ ((T)RPE2): we prove both properties 1 and 2:

1. we assume that $|\mathcal{W}| < \lfloor \frac{d}{2} \rfloor$. Then we consider the set $\mathcal{W}' = \mathcal{W}'_1 \cup \mathcal{W}'_2$ (as previously defined) such that

$$|\mathcal{W}'| \leq 2|\mathcal{W}| < d.$$

$\mathcal{W}'_1$ and $\mathcal{W}'_2$ both point to leaking wires in instances of G_{refresh} . We denote by $\mathcal{W}''_1$ the set of leaking wires on the first instance of G_{refresh} (on input a) such that $\mathcal{W}''_1$ contains $\mathcal{W}'_1$ and all the wires that are leaking within the second instance of G_{refresh} (designated by $\mathcal{W}'_2$ in this

second instance). Hence, we have that $|\mathcal{W}_1''| \leq |\mathcal{W}_1' \cup \mathcal{W}_2'| < d$. From the (t, f) -RPE ((t, f) -TRPE in the TRPE setting) property of G_{refresh} and its amplification order, there exists an input set of shares of a I_1 such that $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |\mathcal{W}|)$) and a set of output shares e_i J_1' of size $n - 1$ such that the input shares of I_1 perfectly simulate the wires designated by $\mathcal{W}_1''$ and J_1' . Similarly, as both instances of G_{refresh} are identical, the same set I_2 but of input shares b perfectly simulates $\mathcal{W}_2''$ (defined as the equivalent of $\mathcal{W}_1''$ on the second instance) and J_2' which points to the same output shares than J_1 but on f_i instead of e_i . We thus have two input sets I_1 and I_2 of size less than t (less than $\min(t, |\mathcal{W}|)$ for TRPE2) whose shares perfectly simulate the wires $\mathcal{W}'$ and the elements $e_i + f_i$ of a set J' of size $n - 1$ with $i \in J_1' = J_2'$. That concludes the proof for the first property.

2. we now assume that $\lfloor \frac{d}{2} \rfloor \leq |\mathcal{W}| < d$. Then we consider the set $\mathcal{W}' = \mathcal{W}_1' \cup \mathcal{W}_2'$ such that

$$|\mathcal{W}'| \leq 2|\mathcal{W}| < 2d.$$

Without loss of generality, let us consider that $|\mathcal{W}_1'| < d$ (the proof is similar in the opposite scenario). From the (t, f) -RPE ((t, f) -TRPE in the TRPE setting) property of G_{refresh} and its amplification order, there exists a set J_1' such that $|J_1'| = n - 1$ and a set of input shares I_1 such that I_1 perfectly simulates $\mathcal{W}_1'$ and J_1' and $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |\mathcal{W}|)$). Thus, we can select a set J' of outputs of G_{add} such that J' corresponds to the outputs of J_1 (for each element e_i designated by J_1 , $e_i + f_i$ is pointed by J). Then, by choosing $I_2 = [n]$, we have two input sets I_1 and I_2 which perfectly simulate $\mathcal{W}'$ and an output set J' of size $n - 1$ such that $|I_1| \leq t$ (for TRPE we would have $|I_1| \leq \min(t, |\mathcal{W}|)$). That concludes the proof for the second property.

We thus proved that G_{add} achieves an amplification order greater than or equal to $\lfloor \frac{d}{2} \rfloor$ for RPE2 (for TRPE2 in the TRPE setting).

Since G_{add} has an amplification order greater than or equal to $\lfloor \frac{d}{2} \rfloor$ for RPE1 and RPE2 (resp. TRPE1 and TRPE2), then G_{add} is a (t, f') -RPE (resp. (t, f') -TRPE) addition gadget for some function f' of amplification order $d' \geq \lfloor \frac{d}{2} \rfloor$, which concludes the proof. $\square$

E Proof of Lemma 9

Proof. We start by proving the first property of the lemma, i.e the amplification order d_1 for TRPE1. The n -share ISW refresh gadget was proven to be $(n - 1)$ -SNI [5], hence it follows from Lemma 6 that $d_1 \geq \min(t + 1, n - t)$. In addition, we know from the proof of Lemma 1 and as explained in section 4.1 that $d_1 \leq t + 1$. It remains to show that $d_1 \leq n - t$. We thus have to exhibit a simulation failure by carefully choosing $n - t$ leaking variables (the leaking set $\mathcal{W}$) together with t leaking output variables (indexed by the set J). Consider the set of output shares indexed by $J = \{1, \dots, t\}$, which corresponds to the first t shares $c_1, \dots, c_t$ of the output. Next, we construct the set of leaking wires $\mathcal{W}$ of size $n - t$. First, observe that the partial sums of the output shares are of the form

$$c_{i,j} = \begin{cases} a_i + r_{1,i} + \dots + r_{j,i} & \text{if } j < i \\ a_i + r_{1,i} + \dots + r_{i-1,i} + r_{i,i+1} + \dots + r_{i,j} & \text{otherwise.} \end{cases}$$

Then, let $\mathcal{W} = \{c_{t+1,t}, \dots, c_{n,t}\}$. We can prove that the constructed set $\mathcal{W}$ along with the set of indexes of output shares $J = \{1, \dots, t\}$ cannot be perfectly simulated with at most $\min(t, |\mathcal{W}|)$

input shares. For this, we consider a variable $s = c_1 + \dots + c_t + c_{t+1,t} + \dots + c_{n,t}$, the sum of the t output shares indexed in J , and the leaking variables from $\mathcal{W}$. Each of the output shares $\{c_i\}_{1 \leq i \leq t}$ is the sum of exactly one input share a_i and $n - 1$ random values. Each of the leaking variables $\{c_{i,t}\}_{t+1 \leq i \leq n}$ is the sum of exactly one input share a_i and t random values. In addition, it can be observed that each random value appears exactly twice in the set of expressions of the variables $\{c_i\}_{1 \leq i \leq t} \cup \{c_{i,t}\}_{t+1 \leq i \leq n}$, so all of the random values are eliminated in the expression of the variable s , which is the sum of all of these variables. Since each of the variables has one input share a_i appearing in its expression, then we have $s = a_1 + \dots + a_n = a$. Thus, simulating the variable s requires the knowledge of the full input, and hence the leaking variables indexed by $\mathcal{W}$ and J cannot be perfectly simulated without the knowledge of the full input. Hence, the set $\mathcal{W}$ of size $n - t$ represents a failure set with respect to TRPE1, and so the function f_1 cannot be of amplification order higher than $n - t$, that is $d_1 \leq n - t$. From the three inequalities $d_1 \geq \min(t + 1, n - t)$, $d_1 \leq t + 1$ and $d_1 \leq n - t$, we obtain $d_1 = \min(t + 1, n - t)$.

Next, we demonstrate the second part of the lemma, *i.e.* the amplification order d_2 for TRPE2. Let $\mathcal{W}$ be a set of leaking wires such that $|\mathcal{W}| < t + 1$. We aim to prove that there exists a set J indexing $n - 1$ output shares such that the leaking variables indexed by both $\mathcal{W}$ and J can be perfectly simulated with the input shares indexed by a set I such that $|I| \leq \min(t, |\mathcal{W}|) = |\mathcal{W}|$. First, we observe that the leaking wires in $\mathcal{W}$ are of the following forms:

1. input share a_i
2. random variable r_{ij} ($i < j$)
3. partial sum $c_{ij} = \begin{cases} a_i + r_{1,i} + \dots + r_{j,i} & \text{if } j < i \\ a_i + r_{1,i} + \dots + r_{i-1,i} + r_{i,i+1} + \dots + r_{i,j} & \text{otherwise.} \end{cases}$

We then build I from an empty set as follows. For every wire in $\mathcal{W}$ of the first or third form, we add index i to I . For every wire in $\mathcal{W}$ of the second form (r_{ij}), if $i \in I$, we add j to I , otherwise we add i to I . By construction we have $|I| \leq |\mathcal{W}| \leq t$. Moreover, the wires in the set $\mathcal{W}$ only depends of the input shares a_i with $i \in I$ which implies that we can perfectly simulate the variables indexed by $\mathcal{W}$ from the input shares indexed by I . We then build the set J as the union of two subsets J_1 and J_2 such that $J_1 = I$ and J_2 is any set satisfying $|J_2| = n - 1 - |I|$ and $J_1 \cap J_2 = \emptyset$. Now, we aim to show that the output shares determined by the indexes in $J = J_1 \cup J_2$ can be further perfectly simulated from the input shares indexed by I (namely given the previous simulation of the variables from $\mathcal{W}$). The simulation works as follows:

- each output share c_i such that $i \in J_1$ can be perfectly simulated with a_i (since $i \in I$) and $n - 1$ uniformly random variables (the same generated r_{ij} can be reused for several c_i);
- for each output share c_i such that $i \in J_2$, we have $i \notin I$ and hence a_i is not available. Since by construction of J_1 , all the variables observed through the set $\mathcal{W}$ are included in the set of variables observed through J_1 , and since $|J_1| \leq |\mathcal{W}| \leq t \leq n - 2$ and $|J_2| = n - 1 - |J_1|$, then each output wire c_i indexed in J_2 has at least one random value that does not appear in any other observation from $\mathcal{W}$ or J_1 , so c_i can be assigned to a fresh random value. This produces a perfect simulation of all output wires indexed in J_2 .

We thus obtain a perfect simulation of the output shares indexed by $J = J_1 \cup J_2$, such that $|J| = n - 1$, together with the variables indexed by $\mathcal{W}$, from the input shares indexed by a set I of size $|I| \leq |\mathcal{W}| \leq t$, so $|I| \leq \min(t, |\mathcal{W}|)$. Hence the ISW refresh gadget is (t, f) -TRPE2 with an amplification order $d_2 \geq t + 1$. In addition, we know from the proof of Lemma 1 and as explained in Section 4.1 that $d_2 \leq t + 1$, hence $d_2 = t + 1$ which concludes the proof. $\square$

F Proof of Lemma 10

Proof. In order to prove that the amplification order d of G_{copy} instantiated with the ISW refresh gadget is equal to $\min(t+1, n-t)$, we first demonstrate that $d \geq \min(t+1, n-t)$ and then we show the existence of failure tuples to argue that $d \leq \min(t+1, n-t)$.

In fact, we already know that the ISW refresh gadget is (t, f_1) -TRPE of amplification order $d_1 = \min(t+1, n-t)$. Then from Lemma 7, we know that G_{copy} instantiated with ISW refresh is also (t, f_2) -TRPE of amplification order $d_2 = d_1 = \min(t+1, n-t)$. Then, from Lemma 4 we have that G_{copy} is (t, f) -RPE of amplification order $d \geq d_2 = \min(t+1, n-t)$. Next, we need to prove that $d \leq \min(t+1, n-t)$. In addition, we know from Lemma 1 that $d \leq t+1$. Hence, it remains to show that it is also upper bounded by $n-t$.

We know from the proof of Lemma 9 that, for the ISW refresh gadget, we can construct a set of leaking wires $\mathcal{W}$ of size $n-t$ along with a set of t indexes of output shares J such that a perfect simulation of both sets $\mathcal{W}$ and J requires the knowledge of the full input sharing *i.e.* $I = [n]$. Then, in the case of the copy gadget G_{copy} , let $\mathcal{W}$ be the set of leaking wires and $J_1, J_2 \subseteq [n]$ be the sets of output shares on the outputs e and f respectively. Then, we can split $\mathcal{W}$ into two distinct subsets $\mathcal{W}_1$ and $\mathcal{W}_2$ such that $\mathcal{W} = \mathcal{W}_1 \cup \mathcal{W}_2$, where $\mathcal{W}_1$ (resp. $\mathcal{W}_2$) is the set of leaking wires of ISW G_{refresh} for the output e (resp. f). Then, in the case where $|J_1| \leq t$, we can construct the set $\mathcal{W} = \mathcal{W}_1$ of size $n-t$ ($\mathcal{W}_2 = \emptyset$) in the exact same way as in the proof of Lemma 9, such that we have simulation failure of $\mathcal{W}_1$ along with the output shares indexed in J_1 on the input of the gadget. Otherwise, in the case where $|J_2| \leq t$, we can construct the set $\mathcal{W} = \mathcal{W}_2$ of size $n-t$ ($\mathcal{W}_1 = \emptyset$) in the exact same way, such that we have simulation failure of $\mathcal{W}_2$ along with the output shares indexed in J_2 on the input of the gadget. Hence, the amplification order d of G_{copy} is upper bounded by $n-t$.

From the three inequalities $d \geq \min(t+1, n-t)$, $d \leq t+1$ and $d \leq n-t$, we conclude that the copy gadget instantiated with ISW refresh is (t, f) -RPE of amplification order $d = \min(t+1, n-t)$. $\square$

G Proof of Lemma 11

Proof. We proceed similarly to the proof of Lemma 8 to show that the function f_1 (resp. f_2) is of amplification order at least $\min(t+1, n-t)$ (resp. $(t+1)$). We split each set $\mathcal{W}$ of leaking wires, into four subsets $\mathcal{W} = \mathcal{W}_1^r \cup \mathcal{W}_1^a \cup \mathcal{W}_2^r \cup \mathcal{W}_2^a$ where $\mathcal{W}_1^r$ (resp. $\mathcal{W}_2^r$) is the set of leaking wires during the computation of $G_{\text{refresh}}(a_1, \dots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \dots, b_n)$), and $\mathcal{W}_1^a$ (resp. $\mathcal{W}_2^a$) is the set of leaking wires of $(e_1, \dots, e_n)$ (resp. $(f_1, \dots, f_n)$). Then, we prove using previous lemmas that the amplification order for TRPE1 (resp. for TRPE2) at most $\min(t+1, n-t)$ (resp. $(t+1)$) by exhibiting failure tuples. Hence the final equalities.

– if $|J| \leq t$ (TRPE1): we prove two properties like in Lemma 8:

1. we assume that $|\mathcal{W}| < \min(t+1, n-t)$, in particular $|\mathcal{W}_1^r| + |\mathcal{W}_1^a| < \min(t+1, n-t) \leq n-t$ and $|\mathcal{W}_2^r| + |\mathcal{W}_2^a| < \min(t+1, n-t) \leq n-t$. Since we have $|J| \leq t$, then $|\mathcal{W}_1^r| + |\mathcal{W}_1^a| + |J| \leq n-1$ and $|\mathcal{W}_2^r| + |\mathcal{W}_2^a| + |J| \leq n-1$. Then from the $(n-1)$ -SNI property of the ISW refresh gadget, and by choosing J_1 and J_2 as the inputs e_i and f_i respectively of the wires $e_i + f_i$ in set J ($|J_1| = |J_2| = |J|$), we know that there exists an input set of shares of a I_1 such that $|I_1| \leq |\mathcal{W}_1^r| \leq |\mathcal{W}| \leq t$ and I_1 perfectly simulates $\mathcal{W}_1^r$, $\mathcal{W}_1^a$ and J_1 . And there exists an

input set of shares of b I_2 such that $|I_2| \leq |\mathcal{W}_2^r| \leq |\mathcal{W}| \leq t$ and I_2 perfectly simulates $\mathcal{W}_2^r$, $\mathcal{W}_2^a$ and J_2 . Thus I_1 and I_2 together perfectly simulate $\mathcal{W}$ and J and are both of size less than $|\mathcal{W}| \leq t$ so less than $\min(t, |\mathcal{W}|)$. Hence, there is no failure on the inputs for any set of leaking wires $\mathcal{W}$ of size strictly less than $\min(t+1, n-t)$ along with a set J of at most t output shares.

2. we now assume that $\min(t+1, n-t) \leq |\mathcal{W}| < 2 \cdot \min(t+1, n-t)$. Without loss of generality, let us consider that $|\mathcal{W}_1^r| + |\mathcal{W}_1^a| < \min(t+1, n-t)$. We consider the set of input shares of b $I_2 = [n]$, which trivially simulates all of the wires in $\mathcal{W}_2^r$, $\mathcal{W}_2^a$ and the inputs f_i of the wires $e_i + f_i$ in set J . Next, since $|\mathcal{W}_1^r| + |\mathcal{W}_1^a| < \min(t+1, n-t) \leq n-t$, by choosing J_1 as the inputs e_i of the wires $e_i + f_i$ in set J , we know that $|\mathcal{W}_1^r| + |\mathcal{W}_1^a| + |J_1| \leq n-1$ and by the $(n-1)$ -SNI property of ISW refresh gadget, there exists a set of input shares of a I_1 such that $|I_1| \leq |\mathcal{W}_1^r| \leq t$ so $|I_1| \leq \min(t, |\mathcal{W}_1^r|) \leq \min(t, |\mathcal{W}|)$ and I_1 perfectly simulates $\mathcal{W}_1^r$, $\mathcal{W}_1^a$ and J_1 . Then I_1 and I_2 together perfectly simulate $\mathcal{W}$ and J , and we only have a failure on input b with $|I_2| = n > t$. Thus, for any set of leaking wires $\mathcal{W}$ such that $\min(t+1, n-t) \leq |\mathcal{W}| < 2 \cdot \min(t+1, n-t)$, we have a failure on at most one of the inputs.

From the above properties, we have that G_{add} is of amplification order $d_1 \geq \min(t+1, n-t)$ for TRPE1. Then, from the proof of Lemma 1, we know that there exists an immediate failure tuple of size $t+1$ (on the input shares), hence $d_1 \leq t+1$. From the same proof of Lemma 1, we also know that there exists a failure tuple of size $2(n-t)$ (with a set of t output shares). Since G_{add} has two inputs, then it results in the following lower bound: $d_1 \geq n-t$. From these three inequalities on d_1 , we obtain $d_1 = \min(t+1, n-t)$.

– if $|J| > t$ (TRPE2): we also prove both properties 1 and 2:

1. we assume that $|\mathcal{W}| < t+1$ with $\mathcal{W} = \mathcal{W}_1^r \cup \mathcal{W}_1^a \cup \mathcal{W}_2^r \cup \mathcal{W}_2^a$. We need to prove that there exists a set J of $n-1$ output wires such that $\mathcal{W}$ and J can be perfectly simulated with sets of input shares I_1 and I_2 such that $|I_1| \leq \min(t, |\mathcal{W}|) = |\mathcal{W}|$ and $|I_2| \leq \min(t, |\mathcal{W}|) = |\mathcal{W}|$. Recall that all of the wires in $\mathcal{W}_1^r$ (resp. $\mathcal{W}_2^r$) are of the following forms:

- (a) input share a_i (resp. b_i).
- (b) random variable r_{ij} (resp. r'_{ij}) with $i < j$.
- (c) partial sum $e_{ij} = \begin{cases} a_i + r_{1,i} + \dots + r_{j,i} & \text{if } j < i \\ a_i + r_{1,i} + \dots + r_{i-1,i} + r_{i,i+1} + \dots + r_{i,j} & \text{otherwise.} \end{cases}$
resp. $f_{ij} = \begin{cases} b_i + r'_{1,i} + \dots + r'_{j,i} & \text{if } j < i \\ b_i + r'_{1,i} + \dots + r'_{i-1,i} + r'_{i,i+1} + \dots + r'_{i,j} & \text{otherwise.} \end{cases}$

In addition, the wires in $\mathcal{W}_1^a$ (resp. $\mathcal{W}_2^a$) are output wires of G_{refresh} of the form e_i (resp. f_i). We build I_1 and I_2 from empty sets as follows. For every wire in $\mathcal{W}_1^a \cup \mathcal{W}_2^a$, we add index i to I_1 and I_2 . Next, for every wire in $\mathcal{W}_1^r \cup \mathcal{W}_2^r$ of the first or third form, we add index i both to I_1 and I_2 . For every wire in $\mathcal{W}_1^r \cup \mathcal{W}_2^r$ of the second form, if $i \in I_1 (= I_2)$, we add j to I_1 and I_2 . Otherwise, we add i to I_1 and I_2 . It is clear that $I_1 = I_2$, and that $|I_1| = |I_2| \leq |\mathcal{W}| \leq t$. Following the t -SNI proof of the ISW refresh gadget, we can show that $\mathcal{W}_1^a$ and $\mathcal{W}_1^r$ are perfectly simulated using shares of indexes in I_1 . Respectively, $\mathcal{W}_2^a$ and $\mathcal{W}_2^r$ are perfectly simulated using shares of indexes in I_2 . So all wires in $\mathcal{W}$ are perfectly simulated using shares of indexes in I_1 and I_2 . We then build the set J of $n-1$ indexes of output shares from two subsets J_1 and J_2 . We define $J_1 = I_1 (= I_2)$, and J_2 as any set such that $J_2 = n-1 - |J_1|$ and $J_1 \cap J_2 = \emptyset$. We now show that the output shares determined by the indexes in $J = J_1 \cup J_2$ can be perfectly simulated from I_1 and I_2 :

- each output share $e_i + f_i$ such that $i \in J_1 \subseteq J$ can be perfectly simulated from e_i and f_i . Precisely, e_i can be perfectly simulated with a_i (since $i \in I_1$) and $n - 1$ uniformly random variables. And each f_i can be perfectly simulated with b_i (since $i \in I_2$) and $n - 1$ uniformly random variables.
- for each output share $e_i + f_i$ such that $i \in J_2$ so $i \notin I_1$, $i \notin I_2$, we show that we can still perfectly simulate e_i and f_i . By construction of the set J_1 , all the variables observed through the set $\mathcal{W}$ are included in the set of variables observed through e_j and f_j for $j \in J_1$, and since $|J_1| = |I_1| \leq |\mathcal{W}| \leq t \leq n - 2$ and $|J_2| = n - 1 - |J_1|$, then each of the wires e_i and f_i for which $e_i + f_i$ is indexed in J_2 has at least one random value that does not appear in any other observation from $\mathcal{W}$ or wires e_i and f_i for which $e_i + f_i$ is indexed in J_1 . So e_i can be assigned to a fresh random value, and f_i can be assigned to a fresh random value. Thus $e_i + f_i$ is also assigned to a random value. This produces a perfect simulation of all output wires indexed in J_2 .

Having a perfect simulation of J_1 and J_2 , we conclude that we can perfectly simulate J along with $\mathcal{W}$ from the sets I_1 and I_2 with $|I_1| = |I_2| \leq |\mathcal{W}|$. So for every set of leaking wires $\mathcal{W}$ of size at most t , there exists a set of $n - 1$ output wires J which can be perfectly simulated along with $\mathcal{W}$ from sets of input shares I_1 and I_2 of sizes at most $|\mathcal{W}| = \min(|\mathcal{W}|, t)$.

2. we now assume that $t + 1 \leq |\mathcal{W}| < 2(t + 1)$. Without loss of generality, let us consider that $|\mathcal{W}_1^r \cup \mathcal{W}_1^a| < t + 1$ (the proof is similar in the opposite scenario). We consider the set of input shares of b $I_2 = [n]$ which trivially simulates all of the wires in $\mathcal{W}_2^r \cup \mathcal{W}_2^a$ and all of the inputs f_i of the output wires $e_i + f_i$. We construct the set I_1 for input shares of a similarly to the earlier construction (when $|\mathcal{W}| < t + 1$), while only considering the sets $\mathcal{W}_1^r$ and $\mathcal{W}_1^a$. The corresponding set I_1 will produce a perfect simulation of $\mathcal{W}_1^r \cup \mathcal{W}_1^a$. So I_1 and I_2 perfectly simulate the set $\mathcal{W}$. Now we choose the set J from two subsets J_1 and J_2 such that $J_1 = I_1$ and J_2 as any set such that $J_2 = n - 1 - |J_1|$ and $J_1 \cap J_2 = \emptyset$. We now show that the output shares determined by the indexes in $J = J_1 \cup J_2$ can be perfectly simulated from I_1 and I_2 :

- each output share $e_i + f_i$ such that $i \in J_1 \subseteq J$ can be perfectly simulated from e_i and f_i . Precisely, e_i can be perfectly simulated with a_i (since $i \in I_1$) and $n - 1$ uniformly random variables. And each f_i can be perfectly simulated with b_i (since $I_2 = [n]$) and $n - 1$ uniformly random variables.
- for each output share $e_i + f_i$ such that $i \in J_2$ so $i \notin I_1$, the input wire f_i can be perfectly simulated with b_i (since $I_2 = [n]$) and $n - 1$ uniformly random variables. In addition, by construction of the set J_1 , all the variables observed through the set $\mathcal{W}_1^r \cup \mathcal{W}_1^a$ are included in the set of variables observed through e_j for $j \in J_1$, and since $|J_1| = |I_1| \leq |\mathcal{W}| \leq t \leq n - 2$ and $|J_2| = n - 1 - |J_1|$, then each of the wires e_i for which $e_i + f_i$ is indexed in J_2 has at least one random value that does not appear in any other observation from $\mathcal{W}_1^r \cup \mathcal{W}_1^a$ or wires e_i for which $e_i + f_i$ is indexed in J_1 . So e_i can be assigned to a fresh random value. Thus $e_i + f_i$ is perfectly simulated from e_i and f_i . This produces a perfect simulation of all output wires indexed in J_2 .

Having a perfect simulation of J_1 and J_2 , we conclude that we can perfectly simulate J along with $\mathcal{W}$ from the sets I_1 and I_2 with $|I_1| \leq t$ so $|I_1| \leq \min(t, |\mathcal{W}|) = t$ (recall that $|\mathcal{W}| \geq t + 1$) and $|I_2| = n$, which is only a failure on one of the inputs b .

From the proof of both properties 1 and 2 for TRPE2, we thus have that G_{add} instantiated with ISW refresh achieves the amplification order $d_2 \geq t + 1$ for TRPE2. In addition, we know from the proof of Lemma 1 and as explained in section 4.1 that $d_2 \leq t + 1$. Hence $d_2 = t + 1$.

We finally proved both amplification orders d_1 and d_2 for TRPE1 and TRPE2 respectively for G_{add} displayed in Algorithm 2 and instantiated with the n -share ISW refresh gadget, which concludes the proof. $\square$

H Proof of Lemma 12

Proof. We start by proving the first property of the lemma. Since the n -share ISW multiplication gadget is $(n-1)$ -SNI [5], then we know from Lemma 6 that

$$d_1 \geq \frac{\min(t+1, n-t)}{2}.$$

In addition, we know from the proof of Lemma 2 that

$$d_1 \leq \frac{t+1}{2}.$$

It remains to show that $d_1 \leq (n-t)/2$. In this purpose, we exhibit a simulation failure on both inputs by carefully choosing $n-t$ leaking variables, with t output variables. Consider the set of indexes of output shares $J = \{1, \dots, t\}$, which corresponds to the first t output shares $c_1, \dots, c_t$. Next, we construct the set of leaking wires $\mathcal{W}$ of size $n-t$. First, observe that the partial sums of the output shares are of the form

$$c_{i,j} = \begin{cases} a_i \cdot b_i + r_{i,1} + \dots + r_{i,j} & \text{if } j < i \\ a_i \cdot b_i + r_{i,1} + \dots + r_{i,i-1} + r_{i,i+1} + \dots + r_{i,j} & \text{otherwise.} \end{cases}$$

Then, let $\mathcal{W} = \{c_{t+1,t}, \dots, c_{n,t}\}$. We can prove that the constructed set $\mathcal{W}$ along with the set of output shares $J = \{1, \dots, t\}$ cannot be perfectly simulated with at most t input shares. For this, we consider a variable $s = c_1 + \dots + c_t + c_{t+1,t} + \dots + c_{n,t}$, the sum of the t output shares indexed in J , and the leaking variables from $\mathcal{W}$. Each of the output shares $\{c_i\}_{1 \leq i \leq t}$ is the sum of

- one product of input shares $a_i \cdot b_i$
- $n-1$ random values,
- at most $n-1$ pairs of input shares products: $(a_i \cdot b_j, a_j \cdot b_i)$ with $i \neq j$.

Each of the leaking variables $\{c_{i,t}\}_{t+1 \leq i \leq n}$ is the sum of

- one product of input shares $a_i \cdot b_i$,
- t random values,
- at most t pairs of input shares products: $(a_i \cdot b_j, a_j \cdot b_i)$ with $i \neq j$.

In addition, each random value appears exactly twice in the set of expressions of the variables $\{c_i\}_{1 \leq i \leq t} \cup \{c_{i,t}\}_{t+1 \leq i \leq n}$, so all the random values are eliminated from the expression of the variable s , which is the sum of all of these variables. Hence, $s = a_1 \cdot b_1 + \dots + a_n \cdot b_n + C$ where C is a variable containing other products of input shares of the form $a_i \cdot b_j$ and $a_j \cdot b_i$ with $i \neq j$. Thus, simulating the variable s requires the knowledge of the full inputs a and b . Since s is constructed from the set of leaking wires $\mathcal{W}$ and the output shares indexed in J , then $\mathcal{W}$ and J cannot be perfectly simulated without the knowledge of the full inputs a and b . Hence, the set $\mathcal{W}$ of size $n-t$ represents a failure tuple on both inputs, and so the function f_1 for RPE1 cannot be of amplification order higher than $(n-t)/2$. Thus, $d_1 \leq (n-t)/2$.

From the three inequalities $d_1 \geq \frac{\min(t+1, n-t)}{2}$, $d_1 \leq (t+1)/2$ and $d_1 \leq (n-t)/2$, we conclude that

$$d_1 = \frac{\min(t+1, n-t)}{2}.$$

Next, we demonstrate the second part of the lemma. Let $\mathcal{W}$ be a set of leaking wires such that $|\mathcal{W}| \leq t$. We aim to prove that there exists a set J of $n-1$ output wires such that $\mathcal{W}$ and J can be perfectly simulated with sets of input shares I_1 on a and I_2 on b such that $|I_1| \leq t$, $|I_2| \leq t$. First, observe that the leaking wires in $\mathcal{W}$ are of the following forms :

1. input shares a_i, b_i , product of shares $a_i \cdot b_i$.
2. partial sum $c_{i,j} = \begin{cases} a_i \cdot b_i + r_{i,1} + \dots + r_{i,j} & \text{if } j < i \\ a_i \cdot b_i + r_{i,1} + \dots + r_{i,i-1} + r_{i,i+1} + \dots + r_{i,j} & \text{otherwise.} \end{cases}$
3. random variable r_{ij} for $i < j$, variable $r_{ji} = a_i \cdot b_j + r_{ij} + a_j \cdot b_i$ for $j > i$.
4. product of shares $a_i \cdot b_j$, or variable $a_i \cdot b_j + r_{ij}$ with $i \neq j$.

We build sets I_1 and I_2 from empty sets as follows. For every wire in $\mathcal{W}$ of the first or second form, we add index i to I_1 and I_2 . For every wire in $\mathcal{W}$ of the third or fourth form, if $i \in I_1$, we add j to I_1 , otherwise we add i to I_1 , and if $i \in I_2$, we add j to I_2 , otherwise we add i to I_2 . Since $\mathcal{W}$ is of size at most t , then $|I_1| \leq t$ and $|I_2| \leq t$. Following the t -SNI property proof from [5], we can show that $\mathcal{W}$ is perfectly simulated using shares of indexes in I_1 and I_2 . We now build the set J of $n-1$ indexes of output shares from two subsets J_1 and J_2 . We define $J_1 = \{i \mid c_{i,j} \text{ is observed in } \mathcal{W}\}$. Next, we define J_2 as any set such that $|J_2| = n-1-|J_1|$ and $J_1 \cap J_2 = \emptyset$. Now, we show that the output shares determined by the indexes in $J = J_1 \cup J_2$ can be perfectly simulated from I_1 and I_2 :

- First consider the output wires indexed in J_1 , which have a partial sum observed. For each such variable c_i , the biggest partial sum which is observed is already simulated. For the remaining r_{ij} in c_i , if $i < j$, then r_{ij} is assigned to a fresh random value. Otherwise, if r_{ji} enters in the computation of any other internal observation, then $i, j \in I_1$ and $i, j \in I_2$, and so r_{ji} can be perfectly simulated from the input shares. If not, then r_{ji} is replaced by the random value r_{ij} . So all output wires indexed in J_1 are perfectly simulated from I_1 and I_2 .
- Now consider the output wires indexed in J_2 . None of the c_i indexed in J_2 has a partial sum observed. Meanwhile, each c_i indexed in J_2 is composed of $n-1$ random values, and at most one of them can enter in the expression of each other output wire c_j . Since by construction of J_1 , all the variables observed through the set $\mathcal{W}$ are included in the set of variables observed through J_1 , and since $|J_1| \leq |\mathcal{W}| \leq t \leq n-2$ and $|J_2| = n-1-|J_1|$, then each output wire c_i indexed in J_2 has at least one random value that does not appear in any other observation from $\mathcal{W}$ or J_1 , so c_i can be assigned to a fresh random value. This produces a perfect simulation of all output wires indexed in J_2 .

We conclude that the set J of $n-1$ wires is perfectly simulated along with $\mathcal{W}$ from the constructed sets I_1 and I_2 of sizes $|I_1| \leq |\mathcal{W}| \leq t$ and $|I_2| \leq |\mathcal{W}| \leq t$. So there is no failure set of observations of size at most t for RPE2 on any of the inputs. Hence $d_2 \geq (t+1)/2$. In addition, we know from the proof of Lemma 2 and as explained in section 4.1 that $d_2 \leq (t+1)/2$. Hence, $d_2 = (t+1)/2$, which concludes the proof for RPE2. $\square$

I Proof of Lemma 13

Recall the procedure of the gadget. We consider that we have an n -share (t, f') -TRPE refresh gadget G_{refresh} achieving the amplification order $d \geq \min(t + 1, n - t)$. First, the gadget G_{mult} performs n executions of the gadget G_{refresh} on the input sharing $(b_1, \dots, b_n)$ to produce:

$$\begin{aligned} (b_1^{(1)}, \dots, b_n^{(1)}) &\leftarrow G_{\text{refresh}}(b_1, \dots, b_n) \\ &\dots \\ (b_1^{(n)}, \dots, b_n^{(n)}) &\leftarrow G_{\text{refresh}}(b_1, \dots, b_n) \end{aligned}$$

then, the gadget constructs the matrix of the cross product of input shares using the refreshed input shares of b :

$$M = \begin{pmatrix} a_1 \cdot b_1^{(1)} & a_1 \cdot b_2^{(1)} & \dots & a_1 \cdot b_n^{(1)} \\ a_2 \cdot b_1^{(2)} & a_2 \cdot b_2^{(2)} & \dots & a_2 \cdot b_n^{(2)} \\ \vdots & \vdots & \ddots & \vdots \\ a_n \cdot b_1^{(n)} & a_n \cdot b_2^{(n)} & \dots & a_n \cdot b_n^{(n)} \end{pmatrix}.$$

Then, it picks n^2 random values which define the following matrix:

$$R = \begin{pmatrix} r_{1,1} & r_{1,2} & \dots & r_{1,n} \\ r_{2,1} & r_{2,2} & \dots & r_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ r_{n,1} & r_{n,2} & \dots & r_{n,n} \end{pmatrix}.$$

It then performs an element-wise addition between the matrices M and R :

$$P = M + R = \begin{pmatrix} p_{1,1} & p_{1,2} & \dots & p_{1,n} \\ p_{2,1} & p_{2,2} & \dots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{n,1} & p_{n,2} & \dots & p_{n,n} \end{pmatrix}.$$

At this point, the gadget randomized each product of input shares from the matrix M with a single random value from R . In order to generate the correct output, the gadget adds all the columns of P into a single column V of n elements, and adds all the columns of the transpose matrix R^T into a single column X of n elements:

$$V = \begin{pmatrix} p_{1,1} + \dots + p_{1,n} \\ p_{2,1} + \dots + p_{2,n} \\ \vdots \\ p_{n,1} + \dots + p_{n,n} \end{pmatrix}, \quad X = \begin{pmatrix} r_{1,1} + \dots + r_{n,1} \\ r_{1,2} + \dots + r_{n,2} \\ \vdots \\ r_{1,n} + \dots + r_{n,n} \end{pmatrix}$$

The n -share output is finally defined as $(c_1, \dots, c_n)^T = V + X$ such that

$$\begin{aligned} c_1 &= V_1 + X_1 \\ &\dots \\ c_n &= V_n + X_n. \end{aligned}$$

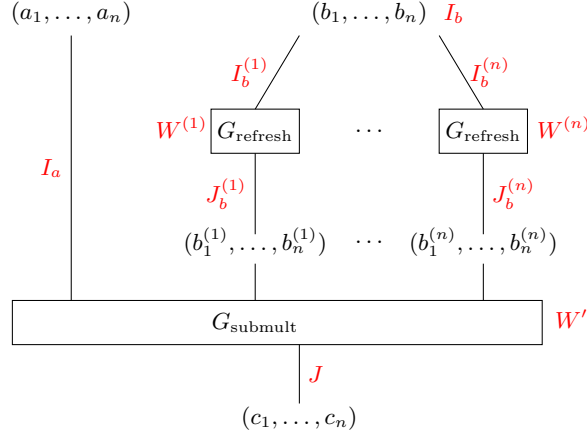


Fig. 3: G_{mult} gadget from Section 5.5.

Figure 3 represents the G_{mult} gadget from a high-level, composed of several blocks. First, a refresh gadget G_{refresh} is executed n independent times on the input sharing of b to produce n fresh copies $b^{(1)}, \dots, b^{(n)}$. Then, the gadget G_{submult} takes as input $(a_1, \dots, a_n)$ and the outputs of the refreshing gadgets $b^{(1)}, \dots, b^{(n)}$ to produce the output of G_{mult} .

In the following proofs, we will denote W to be any set of probes on the global gadget G_{mult} , then W can be split as $W = W' \cup W^{(1)} \cup \dots \cup W^{(n)}$ where $W^{(i)}$ is the set of probes on the internal wires of the execution of G_{refresh} for the fresh sharing $b^{(i)}$ of b , and W' is the set of probes on the internal wires of G_{submult} . We will also denote J to be any set of output wires of G_{mult} (which are the output wires of G_{submult}), and $J_b^{(i)}$ (resp. $I_b^{(i)}$) any set of output wires (resp. input wires) of the execution of G_{refresh} for the fresh sharing $b^{(i)}$ of b . Observe that any probe on the output wires of G_{refresh} for any sharing $b^{(i)}$ can be obtained through internal probes in W' on G_{submult} , so in the beginning we always consider that $J_b^{(i)} = \emptyset$ for all $i \in [n]$.

Observe that any probe in the set W' on the internal wires of G_{submult} is of one of the following forms:

- (a) $a_i, b_j^{(i)}, a_i \cdot b_j^{(i)}, r_{i,j}, p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$,
- (b) $V_{i,j}$ partial sum of the first j terms of V_i . Observe that $V_{i,n} = V_i$,
- (c) $X_{i,j}$ partial sum of the first j terms of X_i . Observe that $X_{i,n} = X_i$.

Also observe that each random value $r_{i,j}$ only appears in the expression of the wires $r_{i,j}, p_{i,j}, V_{i,j}$, or $X_{j,i}$ (so also $c_i = V_i + X_i$ and $c_j = V_j + X_j$), and does not appear anywhere else in the wires.

We will first start by proving some simple claim.

Claim 1 *Let J be a set of output shares of G_{mult} and $W = W' \cup W^{(1)} \cup \dots \cup W^{(n)}$ be a set of leaking wires as described above such that $|J| + |W'| \leq n - 1$ (we only consider the set W' of probes on the internal wires to G_{submult}). Then, for any $i \in J$ such that $V_{i,j} \notin W$ for any $j \in [n]$, the output wire c_i can be perfectly simulated by generating a uniform random value without knowing any of the input shares.*

Proof. Let $i \in J$ such that $V_{i,j} \notin W'$ for any $j \in [n]$. Then we know that the expression of V_i in $c_i = V_i + X_i$ contains $n - 1$ random values since $V_i = p_{i,1} + \dots + p_{i,n}$ and each $p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$

(without counting the random $r_{i,i}$ because it is cancelled out in c_i as it appears in V_i and X_i and $c_i = V_i + X_i$). Observe that each random value $r_{i,k}$ in V_i appears in exactly one other output share $c_k = V_k + X_k$ that comes from the expression of $X_k = r_{1,k} + \dots + r_{i,k} + \dots + r_{n,k}$. In other terms, each output share c_k has exactly one random value in common with V_i in c_i . Then, by probing $|J|$ output shares in J including c_i , there are at least $n - |J|$ remaining random values in V_i that do not appear in any other expression of the output shares. In addition, observe that any probed variable in W' can have in its expression at most one random value in common with V_i (because each random value $r_{i,j}$ appears exactly once in each of the wires $p_{i,j}$, $r_{i,j}$ or X_j). Then, since $|W'| \leq n - |J| - 1$ (because $|J| + |W'| \leq n - 1$), there is at least $n - |J| - (n - |J| - 1) = 1$ remaining random value $r_{i,\ell}$ where $\ell \in [n]$ in V_i , that does not appear in any other expression of the probed values in W' or J . So $c_i = V_i + X_i$ can be perfectly simulated by generating the uniform random value $r_{i,\ell}$, which concludes the proof.

In the following, we will separately prove the TRPE1 then the TRPE2 property on G_{mult} via Lemmas 14 and 17 to demonstrate Lemma 13.

I.1 Proof for TRPE1 property

Lemma 14. *The multiplication gadget G_{mult} is (t, f_1) -TRPE1 of amplification order $d = \min(t + 1, n - t)$*

Proof. We proceed in two steps through the following two lemmas 15 and 16, considering the leaking wires in two distinct ranges.

Lemma 15. *Let J be a set of at most t output shares of G_{mult} . Let W be a set of leaking wires as described above such that $|W| \leq d - 1 \leq t$. Then W and J can be perfectly simulated from at most $\min(t, |W|) = |W|$ shares of each of the inputs a and b .*

Proof. Let J be the set of t output shares of G_{mult} (i.e of G_{submult}), and let $W = W' \cup W^{(1)} \cup \dots \cup W^{(n)}$ with $|W| \leq d - 1 \leq t$ be the set of probes on the global gadget G_{mult} and decomposed as explained earlier. We organize the proof in two steps:

1. We first identify the set of input shares I_a and the sets $J_b^{(i)}$ for $i \in [n]$ which are necessary to perfectly simulate J and W' in G_{submult} .
2. Then, we show that we can perfectly simulate the sets $J_b^{(i)}$ and $W^{(i)}$ for $i \in [n]$ using the simulator of the gadget G_{refresh} . This will determine the sets $I_b^{(i)}$ necessary for each of the n simulations of G_{refresh} , and thus determine the set I_b of input shares on b as $I_b = I_b^{(1)} \cup \dots \cup I_b^{(n)}$.

Using I_b , we will be able to perfectly simulate $J_b^{(i)}$ and $W^{(i)}$ for $i \in [n]$. Then using I_a and $J_b^{(i)}$ for $i \in [n]$, we will be able to perfectly simulate W' and J . This will lead to a perfect simulation of all probes W and output shares in J on the global gadget G_{mult} .

We first start by constructing the set of input shares indices I_a and the sets $J_b^{(k)}$ for $k \in [n]$ depending on the probes in the set W' as follows³:

- (a) For probes of form (a), we add index i to I_a , and index j to $J_b^{(k)}$ for $k \in [n]$.

³ We consider that all $J_b^{(k)}$ are empty at first since all the output shares of G_{refresh} can be probed directly in W' .

- (b) For probes of form (b), we add index i to I_a and to $J_b^{(k)}$ for $k \in [n]$.
- (c) For probes of form (c), we add index i to $J_b^{(k)}$ for $k \in [n]$.

Observe that since $|W| \leq d - 1$, then in particular $|W'| \leq d - 1 \leq t$, then $|I_a| \leq |W'| \leq |W| \leq \min(t, |W|)$ so we have no failure on the input a . We also have $|J_b^{(k)}| \leq |W'| \leq t$.

Simulation of W' : probes of the form (a) can be perfectly simulated from the corresponding input shares in I_a and $J_b^{(k)}$, and by generating uniformly random values $r_{i,j}$ when necessary. Probes of the form (c) are also perfectly simulated by simply generating uniformly random values, since $X_{i,j} = r_{1,i} + \dots + r_{j,i}$. As for probes of the form (b), we know that $i \in I_a$ and $i \in J_b^{(i)}$, then we look at each of the terms $p_{i,j'}$ for $j' \in [j]$ in $V_{i,j} = p_{i,1} + \dots + p_{i,j}$. In particular, if $j \geq i$, the term $p_{i,i}$ is in the partial sum $V_{i,j}$ and is perfectly simulated using the input shares a_i and $b_i^{(i)}$ and by generating the random value $r_{i,i}$. Next, for each $p_{i,j'}$ such that $j' \neq i$, if $j' \in J_b^{(i)}$, then $p_{i,j'}$ can be perfectly simulated from the corresponding input shares and by generating uniformly at random $r_{i,j'}$. Otherwise, if $j' \notin J_b^{(i)}$, then that means that the wires $p_{i,j'}, r_{i,j'}$ and $X_{j'}$ are not probed in W' because otherwise j' would have been added to all $J_b^{(k)}$ for $k \in [n]$. Since the random value $r_{i,j'}$ only appears in the expression of the wires $p_{i,j'}, r_{i,j'}$ and $X_{j'}$ (besides $V_{i,j}$ which is already probed), and of the output wire $c_{j'} = V_{j'} + X_{j'}$, we need to consider two cases:

- $j' \notin J$: in this case, the random value $r_{i,j'}$ can be used to mask the expression of $p_{i,j'}$ in the partial sum $V_{i,j}$, perfectly simulating it without the need to the share $b_{j'}^{(i)}$.
- $j' \in J$: $c_{j'} = V_{j'} + X_{j'}$, and $r_{i,j'}$ is the one of the summed terms in the expression of $X_{j'}$. We know that $V_{j',k} \notin W'$ for any $k \in [n]$ since otherwise j' would have been added to $J_b^{(i)}$. Since in addition we have $|J| + |W| \leq t + d - 1 \leq t + n - t - 1 \leq n - 1$, by claim 1, the output share $c_{j'}$ can be masked by some random value $r_{j',\ell}$. Thus, $X_{j'}$ is masked and $r_{i,j'}$ does not appear anymore in $c_{j'}$. So $r_{i,j'}$ can be used to mask the expression of $p_{i,j'}$ in the partial sum $V_{i,j}$. This brings us to a perfect simulation of $p_{i,j'}$ simply by generating at random $r_{i,j'}$.

By perfectly simulating each of the terms $p_{i,j'}$ for $j' \in [j]$ in the probed wire $V_{i,j}$ independently, we can perfectly simulate their sum and thus perfectly simulate $V_{i,j}$. This brings us to a perfect simulation of the set W' .

Simulation of J : Let $i \in J$.

- if $V_{i,j} \notin W'$ for any $j \in [n]$, then by claim 1, c_i is perfectly simulated by simply generating a uniform random value $r_{i,\ell}$ for some $\ell \in [n]$.
- if $V_{i,j} \in W'$ for at least one $j \in [n]$, then let $V_{i,j'}$ be the largest of the probed partial sums. All of the partial sums including $V_{i,j'}$ are perfectly simulated as described earlier. Then, let us consider $c_i + V_{i,j'} = p_{i,j'+1} + \dots + p_{i,n} + X_i$. The wire X_i can be perfectly simulated by generating uniform random values. As for each of the terms $p_{i,j'+1}, \dots, p_{i,n}$, they can each be perfectly simulated in the exact same way each of the terms in $V_{i,j'}$ are simulated independently.

In the particular case where $j' \leq i$ then the term $p_{i,i} = a_i \cdot b_i^{(i)} + r_{i,i}$ appears in the expression of $c_i + V_{i,j'}$, and in this case, the random value $r_{i,i}$ is cancelled out in the expression of $c_i + V_{i,j'}$ since it appears in both $p_{i,i}$ and X_i , and $c_i + V_{i,j'} = p_{i,j'+1} + \dots + p_{i,i} + \dots + p_{i,n} + X_i$. So to simulate the term $p_{i,i}$ in $c_i + V_{i,j'}$ we need both input shares a_i and $b_i^{(i)}$. This is already the case by construction because we assume that $V_{i,j'} \in W'$.

Thus, by perfectly simulating $V_{i,j'}$ and $c_i + V_{i,j'}$, the output share c_i is also perfectly simulated.

Also, since $|J_b^{(k)}| \leq |W'| \leq t$ and $|W^{(k)}| \leq d - 1$, and since G_{refresh} is (t, f') -TRPE achieving the amplification order d , then we can perfectly simulate sets $J_b^{(k)}$ and $W^{(k)}$ from the set of input shares $I_b^{(k)}$ such that $|I_b^{(k)}| \leq |W^{(k)}| \leq t$ for $k \in [n]$. Thus, we can let $I_b = I_b^{(1)} \cup \dots \cup I_b^{(n)}$ and we have $|I_b| \leq |W^{(1)}| + \dots + |W^{(n)}| \leq |W| \leq \min(|W|, t)$, so we have no failure on the input b either. Until now, we have shown that we can simulate all sets $W^{(k)}$ and $J_b^{(k)}$ from I_b of size at most $\min(|W|, t)$. It remains to show that we can also perfectly simulate the sets W' and J from I_a and $J_b^{(k)}$ for $k \in [n]$.

We have shown that we can perfectly simulate any set of t output shares J and any set of probes W of size at most $d - 1$, with at most $\min(|W|, t)$ shares of each of the inputs a and b . This concludes the proof of Lemma 15. $\square$

Remark 2. We can observe that for this lemma to apply on G_{mult} , we do not need the pre-processing phase of the refresh on input b . In fact, we can see that during the construction of the sets $J_b^{(k)}$, we add each index to all of the sets for all $k \in [n]$. However, executing n refreshings on the input b is necessary to prove the next result, specifically when we consider W such that $d \leq |W| \leq 2d - 1$.

To get back to the proof of Lemma 14, we also need the following result.

Lemma 16. *Let J be a set of at most t output shares of G_{mult} . Let W be a set of leaking wires as described above such that $d \leq |W| \leq 2d - 1$. Then W and J can be perfectly simulated from the sets of input shares I_a and I_b such that $|I_a| \leq \min(|W|, t)$ **or** $|I_b| \leq \min(|W|, t)$. In other terms, we have a simulation failure on at most one of the inputs a or b .*

Proof. Recall that the set W can be split into subsets $W = W' \cup W^{(1)} \cup \dots \cup W^{(n)}$ as described above. We can consider two cases.

Case 1: $|W'| \leq d - 1$. This case is similar to the case of Lemma 15, so we can construct the set I_a in the same way as in the proof of Lemma 15, and we can eventually consider $I_b = [n]$. We know that $|I_a| \leq |W'| \leq d - 1 \leq t$, so there is no failure on the input a . And all probes in W' can be simulated like in the proof of Lemma 15 with I_a and trivially with $I_b = [n]$. Also, all probes in $W^{(1)} \cup \dots \cup W^{(n)}$ can be trivially simulated since we have access to the full input b . As for output shares in J , whenever $i \in J \cap I_a$, then $c_i = V_i + X_i$ is easily simulated using $I_b = [n]$. If $i \in J$ but $i \notin I_a$, then $V_{i,j} \notin W'$ for any $j \in [n]$ and since $|J| + |W'| \leq t + d - 1 \leq t + n - t - 1 \leq n - 1$, c_i is perfectly simulated by a single random value thanks to claim 1. Thus, W and J are perfectly simulated with at most $|W'| \leq \min(|W|, t)$ shares of a and eventually n shares of b .

Case 2: $|W'| \geq d$ (and thus $|W^{(1)} \cup \dots \cup W^{(n)}| \leq d - 1$). In this case, we will construct the sets I_a and $J_b^{(k)}$ from empty sets, in a way that we will have a simulation failure on at most one of the inputs a or b , and we will be able to perfectly simulate W' and output shares in J using I_a and $J_b^{(k)}$. We will also show how to perfectly simulate all $J_b^{(k)}$ and $W^{(k)}$ using a set of input shares I_b .

First, we construct the sets I_a and $J_b^{(k)}$ depending on the probes in W' as follows:

- (a) For probes of form (a), we add index i to I_a , and index j only to $J_b^{(i)}$.
- (b) For probes of form (b), we add index i to I_a and only to $J_b^{(i)}$.
- (c) For probes of form (c), we add index i to $J_b^{(k)}$ for all $k \in [n]$.

In the rest of the proof, we will show that if we have a failure on one of the inputs, we can still perfectly simulate W and J without a failure on the other input. In this purpose, we will consider two cases: in the first case (2.1), we will have a failure on input a (i.e., more than $\min(t, |W|)$ shares of a are added to I_a) and in the second case (2.2), we won't have a failure on input a , and so we will eventually have a failure on input b .

Case 2.1: Simulation failure on input a . Notice that by construction we always have $|I_a| \leq |W'| \leq |W|$. Thus, a simulation failure on input a for TRPE1 means that the set I_a is of size $|I_a| \geq t+1 \geq d$. We will first start by showing that the sets $W^{(k)}$ and $J_b^{(k)}$ can be perfectly simulated using the simulator of G_{refresh} without a failure on the input b . Next, we will show that W' and output shares in J can be perfectly simulated using I_a and $J_b^{(k)}$.

Since we only add shares indices to I_a when we have probes of the form (a) or (b), this means that we have at least $t+1$ probes of these two forms with $t+1$ different values for the index i . In addition, since we have at least $t+1$ probes (a) or (b) with distinct values for the index i , then this also means that each of the sets $J_b^{(i)}$ built from these probes has at most one share of $b^{(i)}$ added to it by construction. In other terms, when we only consider probes of the form (a) and (b) with distinct i , we have $|J_b^{(k)}| \leq 1$ for each $k \in [n]$.

Now let us consider the remaining probes in W which are either in W' of the form (c), in W' of the form (a)/(b) for which $i \in I_a$ or in $W^{(1)} \cup \dots \cup W^{(n)}$. Since $|I_a| \geq t+1 \geq d$, then there are at most $d-1$ of these remaining probes. Without loss of generality, we consider that there are exactly $d-1$ instead of at most $d-1$ probes. Let m be the number of probes in $W^{(1)} \cup \dots \cup W^{(n)}$ and $d-1-m$ the remaining in W' of the form (c) or of the form (a)/(b) for which $i \in I_a$.

Since each wire in W' of the form (c) or of the form (a)/(b) for which $i \in I_a$ results in adding at most one more share index to each $J_b^{(k)}$ for $k \in [n]$, then we have $|J_b^{(k)}| \leq 1 + (d-1-m) = d-m$. And $|W^{(1)} \cup \dots \cup W^{(n)}| \leq m$, in particular $|W^{(k)}| \leq m$ for any $k \in [n]$.

- if $m = 0$, then $W^{(k)} = \emptyset$ for any $k \in [n]$, and $|J_b^{(k)}| \leq d \leq \min(t+1, n-t) \leq \lfloor \frac{n+1}{2} \rfloor \leq n-1$, so by the TRPE property of G_{refresh} for any $t \leq n-1$, all of the $J_b^{(k)}$ sets can be perfectly simulated with no knowledge of the input shares of b since $W^{(k)} = \emptyset$, so $I_b^{(k)} = \emptyset$. Hence, $I_b = I_b^{(1)} \cup \dots \cup I_b^{(n)} = \emptyset$ and we have no simulation failure on the input b .
- if $m > 0$, then $|J_b^{(k)}| \leq d-1 \leq t$ and $|W^{(1)} \cup \dots \cup W^{(n)}| \leq d-1$, in particular $|W^{(k)}| \leq d-1$ for each $k \in [n]$. Thus, by the (t, f) -TRPE property of the refresh gadget G_{refresh} achieving the amplification order d for any $t \leq n-1$, we can perfectly simulate both sets for each $k \in [n]$ with $I_b^{(k)}$ such that $|I_b^{(k)}| \leq |W^{(k)}|$. Thus, we can let $I_b = I_b^{(1)} \cup \dots \cup I_b^{(n)}$ so we can have $|I_b| \leq |W^{(1)} \cup \dots \cup W^{(n)}| \leq d-1 \leq t$, and we can perfectly simulate $W^{(1)} \cup \dots \cup W^{(n)}$ along with $J_b^{(1)} \cup \dots \cup J_b^{(n)}$ from the set I_b without a simulation failure on input b .

So far we proved that if we have $|I_a| \geq t+1$, then we must have $|I_b| \leq |W^{(1)} \cup \dots \cup W^{(n)}| \leq d-1 \leq t$, and $W^{(1)} \cup \dots \cup W^{(n)}$ can be perfectly simulated along with $J_b^{(1)} \cup \dots \cup J_b^{(n)}$ from the set I_b . Next we need to prove that we can perfectly simulate W' and J from these sets I_a and $J_b^{(1)} \cup \dots \cup J_b^{(n)}$.

Case 2.1.1: $I_a = [n]$. This only occurs by construction in the case where $|W| = 2d-1 = n$ so when $d = d_{\text{max}} = \lfloor \frac{n+1}{2} \rfloor$ for $t = \lceil \frac{n-1}{2} \rceil$. In this case, since $|W| \leq 2d-1 \leq (n+1)-1 \leq n$, then all probes in W are all in W' of the form (a) or (b) with n distinct values for the index i and so

$|J_b^{(i)}| \leq 1$ for all $i \in [n]$. In other words, for each $i \in [n]$ there is exactly one probe in W' of the form (a) or (b) and no probe of the form (c) i.e $X_{i,j}$ nor probes in $W^{(1)} \cup \dots \cup W^{(n)}$. We will prove that all the probes in W and in J can be perfectly simulated from these constructed sets I_a and $J_b^{(i)}$ for $i \in [n]$. For this, for each $i \in [n]$ we consider three cases:

- $V_{i,j} \notin W'$ for any $j \in [n]$, then we know that there exists a probe of the form (a) in W' with index i , in other terms, $a_i \in W'$, or $\exists! j \in [n]$ such that $b_j^{(i)}$ or $a_i \cdot b_j^{(i)}$ or $r_{i,j}$ or $p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$ is probed in W' . The corresponding probe is perfectly simulated by construction of the sets I_a and $J_b^{(i)}$.

If we also have $i \in J$, then we know that we only have one probe of the form (a) for the considered index i in W' and no probe of the form (b) or any probe of the form (c). And since there are t output shares probed in J , then there are at least $n - t - 1 > 1$ (since $t = \lceil \frac{n-1}{2} \rceil$) remaining random values which only appear in the expression of c_i , and any of them can be used to perfectly simulate c_i without the knowledge of the input shares (i.e., to mask c_i).

- $V_{i,n} \in W'$ then $V_{i,n}$ contains in its expression n random values $r_{i,1}, \dots, r_{i,n}$. Since there are no probes of the form (a) for the index i , and no probes of the form (c), then each of these random values appears at most once in each of the expressions of the probed outputs c_j in J . With t probed output shares, there are $n - t > 1$ (since $t = \lceil \frac{n-1}{2} \rceil$) remaining random values which only appear in the expression of $V_{i,n}$ and any of them can be used to perfectly simulate $V_{i,n}$, i.e., mask $V_{i,n}$.

If in addition we have $i \in J$, then the output share c_i is perfectly simulated by simulating $V_{i,n}$ and simulating $c_i + V_{i,n} = X_i$ which is perfectly simulated by generating uniform random values.

- $V_{i,j} \in W'$ for some $j \in [n]$ such that $1 < j < n$ ($j > 1$ because otherwise it would be the wire $p_{i,1}$ which is probed). Thus, $V_{i,j}$ is the sum of at least two wires p_{i,j_1} and p_{i,j_2} .
 - If $i \notin J$, then c_i is not probed and $V_{i,j}$ is the sum of at most $n - 1$ terms of the form $p_{i,1} = a_i \cdot b_1^{(i)} + r_{i,1}, \dots, p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$. We have that $i \in I_a$ by construction and $j \in J_b^{(i)}$. In fact we can reconstruct $J_b^{(i)}$ into $J_b^{(i)} = \{1, \dots, j\}$ such that $|J_b^{(i)}| \leq n - 1$ and since $W^{(i)} = \emptyset$, then by the (t, f) -TRPE1 property of G_{refresh} for any $t \leq n - 1$, we still have no failure on the input b and we still have $|I_b^{(i)}| \leq |W^{(i)}| = 0$. In addition, we can perfectly simulate this way all of the summed terms in $V_{i,j}$ by using the corresponding input shares and thus we can perfectly simulate $V_{i,j}$. Since we have no probes of the form (a) for this same index i , then reconstructing $J_b^{(i)}$ does not affect the simulation of the probes.
 - If $i \in J$, then we consider $V_{i,j}$ and $c_i + V_{i,j}$. Since we have no probes of the form (a) for the index i , then as proven before, with t probed output shares, there are at least $n - t > 1$ remaining random values which only appear in the expression of $V_{i,j}$ or $c_i + V_{i,j}$. Any of these random values can be used to mask the expression of $V_{i,j}$ or $c_i + V_{i,j}$. In the case where the expression of $V_{i,j}$ is masked, then we can reconstruct as before the set $J_b^{(i)}$ with at most $n - 1$ output shares of $b^{(i)}$ in order to perfectly simulate all the terms $p_{i,k}$ in $c_i + V_{i,j}$ including the shares of $b^{(i)}$ and thus perfectly simulate $c_i + V_{i,j}$ (the rest of the terms are just random values to be generated uniformly at random). In the other case where the expression of $c_i + V_{i,j}$ is masked, we can also reconstruct the set $J_b^{(i)}$ with at most $n - 1$ output shares of $b^{(i)}$ in order to perfectly simulate all the summed terms in $V_{i,j}$. In either case, by perfectly simulating one term ($V_{i,j}$ or $c_i + V_{i,j}$) masked by a random value, and perfectly simulating

the remaining one with $i \in I_a$ and the reconstructed set $J_b^{(i)}$, we can perfectly simulate both $V_{i,j}$ and $c_i + V_{i,j}$ and hence also perfectly simulate the output share c_i .

So we proved that we can perfectly simulate the sets W' and J from the constructed set I_a and from sets $J_b^{(i)}$ such that $|J_b^{(i)}| \leq n - 1$ for all $i \in [n]$. Furthermore, from the TRPE property of G_{refresh} for any $t \leq n - 1$ and the fact that $W^{(i)} = \emptyset$ for all $i \in [n]$, we have no simulation failure on the input b . This concludes the simulation of W and output shares in J for the case where $I_a = [n]$.

Case 2.1.2: $I_a \subset [n]$ with $|I_a| \leq n - 1$. In this case, we have at least one index $k \in [n] \setminus I_a$ for which there are no probes in W' of the form (a) or (b). In other terms, no partial sum of V_k is probed, no product of shares $a_k \cdot b_j^{(k)}$ or $p_{k,j}$ is probed, and no random value $r_{k,j}$ is probed since otherwise we would have $k \in I_a$ by construction.

On another hand, since $|I_a| \geq t + 1$, there are at most $d - 1 \leq n - t - 1$ remaining probes of the form (c) in W' , and since we have t output shares in the set J , there exists at least one wire X_ℓ such that $\ell \notin J$ and for which there is no partial sum $X_{\ell,j}$ probed in W' .

These two wires X_ℓ and V_k for $\ell, k \in [n]$ will be very important for the simulation of the sets W' and J . In particular, we need the two following claims.

Claim 2 *Let $i \in J$. Suppose that $i \notin I_a$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{i,\ell}$, in other terms $c_i \leftarrow r_{i,\ell}$.*

Proof. This claim can be proved easily, since we suppose that $i \notin I_a$ so the random value $r_{i,\ell}$ and $p_{i,\ell}$ are not probed in W' . In addition, since $\ell \notin J$ and $X_{\ell,j} \notin W'$ for all $j \in [n]$, then the random value $r_{i,\ell}$ does not appear in any other probed wire expression except in c_i , then c_i can be masked by the random value $r_{i,\ell}$. $\square$

Claim 3 *Let $i \in J$. Suppose that $X_{i,j} \notin W'$ for any $j \in [n]$. Suppose that $i \in I_a$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{k,i}$, in other terms $c_i \leftarrow r_{k,i}$.*

Proof. Since we suppose that $k \notin I_a$, then the random value $r_{k,i}$ or $p_{k,i}$ or $V_{k,j}$ for all $j \in [n]$ are not probed in W' . Then, if $k \notin J$, then the random value $r_{k,i}$ does not appear in the expression of any other probed wire in W' or J and c_i can be masked by the random value $r_{k,i}$. Otherwise, if $k \in J$, then by Claim 2, $c_k = V_k + X_k$ can be masked by $r_{k,\ell}$ and so c_i can also be masked by $r_{k,i}$ since $i \neq \ell$ (because $i \in J$ and $\ell \notin J$). $\square$

From these two claims, we are now ready to show that W' and J can be perfectly simulated with the sets I_a and $J_b^{(1)} \cup \dots \cup J_b^{(n)}$ as constructed earlier with respect to the probes in the set W' . Recall that all probes in $W^{(1)} \cup \dots \cup W^{(n)}$ and $J_b^{(1)} \cup \dots \cup J_b^{(n)}$ are perfectly simulated using I_b and the simulator of G_{refresh} .

Simulation of W' . Probes of the form (a) and (c) are trivially simulated by construction of the sets of input shares and by generating uniformly at random the necessary random values. Let us now check the probes of the form (b). Let $V_{i,j} = p_{i,1} + \dots + p_{i,j}$ be such a probe. Let us consider each of the terms $p_{i,j'}$ for $j' \in [j]$. if $j' = i$, then by construction $p_{i,i}$ is perfectly simulated using a_i and $b_i^{(i)}$ and by generating the random value $r_{i,i}$ if needed. Otherwise, let $j' \neq i$. If $j' \in J_b^{(i)}$ then the simulation of $p_{i,j'}$ is straightforward. Otherwise if $j' \notin J_b^{(i)}$, then we know that none of the wires

$r_{i,j'}$ or $p_{i,j'}$ or $X_{j',s}$ for all $s \in [n]$ are probed in W' . Thus, $r_{i,j'}$ can be eventually used to mask the expression of $p_{i,j'}$ without the need of the share $b_{j'}^{(i)}$ for the simulation. Meanwhile, we still need to check if $j' \in J$, since $X_{j'}$ appears in the expression of $c_{j'} = V_{j'} + X_{j'}$. Then we consider two cases:

- If $j' \notin I_a$, then by claim 2, $c_{j'}$ can be masked by the random value $r_{j',\ell}$ and so $r_{i,j'}$ does not appear in the expression of $X_{j'}$ in $c_{j'}$ anymore, and $r_{i,j'}$ can be used to mask $p_{i,j'}$.
- Otherwise, if $j' \in I_a$, then by claim 3, $c_{j'}$ can be masked by the random value $r_{k,j'}$ and so $r_{i,j'}$ does not appear in the expression of $X_{j'}$ in $c_{j'}$ anymore, and $r_{i,j'}$ can be used to mask $p_{i,j'}$ (since $i \notin k$).

Thus, each term $p_{i,j'}$ in $V_{i,j}$ can be perfectly simulated and thus $V_{i,j} = p_{i,1} + \dots + p_{i,j}$ can be perfectly simulated. This concludes the simulation of the set W' .

Simulation of J . Let $i \in J$. If $i \notin I_a$, then by claim 2, c_i is perfectly simulated by generating the random value $r_{i,\ell}$. Otherwise, let $i \in I_a$. If $X_{i,j} \notin W$ for any $j \in [n]$, then by claim 3, c_i is perfectly simulated by generating the random value $r_{k,i}$. Otherwise, we can show that we can perfectly simulate each term in $c_i = V_i + X_i$. In particular, each term in X_i can be simulated by generating the underlying random value uniformly. For each term in the sum V_i , we know in particular that $a_i \cdot b_i^{(i)}$ is perfectly simulated since $X_{i,j} \in W'$ for at least one $j \in [n]$ so $i \in J_b^{(i)}$ by construction. For the other terms in V_i , they can be perfectly simulated in the exact same way as we simulated the probes $V_{i,j}$ of the form (b) in the set W' . So c_i is perfectly simulated by summing all the perfectly simulated terms. This concludes the simulation proof for the set J .

Up until now, we have concluded that if we have a constructed set I_a of size at least $t + 1$, then we can perfectly simulate the sets W and J without having a simulation failure on the input b . In the rest of the proof, we will consider that $|I_a| \leq t$ (along with $|I_a| \leq |W|$ by construction meaning that we have no failure on input a), and we will prove that we can perfectly simulate W and J with at most a simulation failure on b . Recall that we are also considering that $|W'| \geq d$ and $|W^{(1)} \cup \dots \cup W^{(n)}| \leq d - 1$.

Case 2.2: $|I_a| \leq t$. This means that the number of probes of the form (a) or (b) in W' with distinct values for the index i is at most t .

First, let us consider that $|I_a| \geq d$ (this is the case where $d = n - t \leq t + 1$). Then, as proved earlier, and with t additional output shares in J of the form $c_i = V_i + X_i$, there are at least one X_ℓ remaining such that $\ell \notin J$ and $X_{\ell,j} \notin W$ for all $j \in [n]$. In this case, we can set $I_b = [n]$ and I_a as constructed with respect to the probes in W' . It is clear that all probes in $W^{(1)} \cup \dots \cup W^{(n)}$ and $J_b^{(1)} \cup \dots \cup J_b^{(n)}$ are trivially simulated using $I_b = [n]$. In addition, all probes in W' are also perfectly simulated by construction of the set I_a and using $I_b = [n]$ and generating the necessary random values. This means that we can perfectly simulate all of the set of probes $W = W' \cup W^{(1)} \cup \dots \cup W^{(n)}$. As for the set of output shares indexed in J . Let $i \in J$. If $i \in I_a$, then c_i is perfectly simulated using the share a_i and $I_b = [n]$, and by generating the necessary random values. Otherwise, if $i \notin I_a$, then in the same way as in claim 2, c_i can be masked by the random value $r_{i,\ell}$ (because $\ell \notin J$ and $X_{\ell,j} \notin W$ for all $j \in [n]$), so a_i is not needed for the simulation of c_i . This proves that we can perfectly simulate the output shares in J with I_a and $I_b = [n]$.

In the rest, we suppose that $|I_a| \leq d - 1 \leq n - t - 1$, i.e., the number of probes of the form (a) or (b) in W' with distinct values for the index i is at most $d - 1 \leq n - t - 1$. In this case, and with t

additional output shares, we have at least one index k such that $k \notin J$ and for which there are no probes in W' of the form (a) or (b). In other terms, no partial sum of V_k is probed, no product of shares $a_k \cdot b_j^{(k)}$ or $p_{k,j}$ is probed, and no random value $r_{k,j}$ is probed. Now we reason on the number of probes of the form (c) in W' :

- We first consider the special case where the number of $X_{i,j}$ probed (of form (c) in W') for distinct values of i is equal to n . In other terms, we have probes $X_{1,j_1}, \dots, X_{n,j_n}$ for certain values $j_1, \dots, j_n$. Since the set of probes W satisfies $|W| \leq n$ (because $2d - 1 \leq n$), then this means that there are no remaining probes in the set W except for the n probes of the form (c) in W' . This is an easy case since we can let $I_b = [n]$ and $I_a = J$ (always without a failure on a since in the case where $|W| = n$, we have $d = t + 1 = n - t$ so $|I_a| = |J| \leq \min(t, |W|)$ where $t \leq |W|$). This allows us to trivially simulate all output wires indexed in J , and since the remaining wires in W are just sums of random values, we can simulate them by generating the corresponding random values.
- Next, we consider that there is at least one index ℓ such that $X_{\ell,j} \notin W$ for all $j \in [n]$ (in other terms, the number of probes of the form $X_{i,j}$ for distinct values of i is at most $n - 1$). Notice that this case is slightly different than the case of claims 2 and 3, since ℓ can be in the set J . In this case, we can let $I_b = [n]$ so that we can perfectly simulate all wires in $W^{(1)} \cup \dots \cup W^{(n)}$ and $J_b^{(1)} \cup \dots \cup J_b^{(n)}$ using $I_b = [n]$, and we can perfectly simulate all wires in W' using I_a by construction and $I_b = [n]$ and generating the necessary random values. Next, we need to prove that we can perfectly simulate all output shares in J . Let $i \in J$. If $i \in I_a$, then c_i is perfectly simulated using a_i and $I_b = [n]$ and generating the necessary random values. Next, if $i \notin I_a$, then if $\ell \notin J$, we can use claim 2 to prove that we can replace the expression of $c_i = V_i + X_i$ by the random value $r_{i,\ell}$ and so the share a_i is not needed for the simulation of c_i (even if $X_{i,j}$ for a certain j is probed, the expression of V_i is still masked by $r_{i,\ell}$ and a_i is not needed to simulate X_i which is a sum of random values). Meanwhile, if $\ell \in J$, then we cannot directly use the random value $r_{i,\ell}$ to mask the expression of c_i . But since $X_{\ell,j} \notin W$ for all $j \in [n]$, and since $r_{k,\ell} \notin W$ because $k \notin I_a$ by assumption, then c_ℓ can be masked by the random value $r_{k,\ell}$, i.e $c_\ell = V_\ell + X_\ell \leftarrow r_{k,\ell}$. Since $i \in J$ and $k \notin J$, then $i \neq k$ and the random value $r_{i,\ell}$ does not appear anymore in X_ℓ in the expression of c_ℓ . Since $i \notin I_a$ then $r_{i,\ell}$ can be used to mask the expression of the output share c_i indexed in J and so the share a_i is not needed for the simulation of c_i . This proves that we can perfectly simulate all shares in J with the constructed sets I_a and $I_b = [n]$.

We managed to show that whenever the construction of the set I_a gives $|I_a| \leq t$, then we can perfectly simulate the sets W and J with at most a failure on input b and while still having $|I_a| \leq t$ and $|I_a| \leq |W|$.

By considering both cases $|I_a| \geq t + 1$ and $|I_a| \leq t$, we covered all the cases for the simulation, and we proved that we can always perfectly simulate the set of probes W along with the set of output shares J while having a failure on at most one of the inputs. This concludes the proof of Lemma 16. $\square$

I.2 Proof for TRPE2 property

Lemma 17. *The above multiplication gadget is (t, f_2) -TRPE2 of amplification order $d \geq \min(t + 1, n - t)$*

Proof. To prove the lemma, we proceed in two steps through the following two lemmas 18 and 19.

Lemma 18. *Let W be a set of leaking wires as described above such that $|W| < \min(t+1, n-t)$. Then there exists a set J of $n-1$ output shares, such that W and J can be perfectly simulated from at most $\min(|W|, t) = |W|$ shares of each of the inputs a and b .*

Proof. We will construct the set of input shares indices I_a and the sets of output shares $J_b^{(k)}$ for $k \in [n]$ depending on the probes in the set W' (recall that $W = W' \cup W^{(1)} \cup \dots \cup W^{(n)}$) as follows (we consider that all $J_b^{(k)}$ are empty at first since all the output shares of G_{refresh} can be probed directly in W'):

- (a) For probes of form (a), we add index i to I_a , and index j to $J_b^{(k)}$ for $k \in [n]$.
- (b) For probes of form (b), we add index i to I_a and to $J_b^{(k)}$ for $k \in [n]$.
- (c) For probes of form (c), we add index i to $J_b^{(k)}$ for $k \in [n]$.

Observe that since $|W| < \min(t+1, n-t)$, then in particular $|W'| \leq \min(t+1, n-t) - 1 \leq t$, then $|I_a| \leq |W'| \leq |W| \leq t$ so we have no failure on the input a . Also, since $|J_b^{(k)}| \leq |W'| \leq t$ and $|W^{(k)}| < \min(t+1, n-t)$, then by the (t, f') -TRPE1 property of G_{refresh} , we will be able to simulate sets $J_b^{(k)}$ and $W^{(k)}$ from the set of input shares $I_b^{(k)}$ such that $|I_b^{(k)}| \leq |W^{(k)}| \leq t$ for $k \in [n]$. Thus, we can let $I_b = I_b^{(1)} \cup \dots \cup I_b^{(n)}$ and we have $|I_b| \leq |W^{(1)} \cup \dots \cup W^{(n)}| \leq |W| \leq t$, so we have no failure on the input b either. Until now, we have shown that we can simulate all sets $W^{(k)}$ and $J_b^{(k)}$ from I_b of size at most $\min(|W|, t) = |W|$. It remains to show that we can also perfectly simulate the set W' and a well chosen set J of $n-1$ output shares, from I_a and $J_b^{(k)}$ for $k \in [n]$. We will choose the set J from two subsets $J = J_1 \cup J_2$, where $J_1 = \{i \mid i \in J_b^{(k)} \text{ for any } k \in [n]\}$, and $J_2 \subset [n]$ is any set such that $J_1 \cap J_2 = \emptyset$ and $|J_1 \cup J_2| = n-1$. Let $\ell \in [n]$ be the index such that $\ell \notin J$. Since $|W| \leq \min(t+1, n-t) - 1 \leq n-1$, then by construction of the sets $J_b^{(k)}$, we have that $|J_b^{(1)} \cup \dots \cup J_b^{(n)}| \leq n-1$, then for the index ℓ , we have that $\ell \notin J_b^{(k)}$ for all $k \in [n]$, then $X_{\ell,j} \notin W$ for any $j \in [n]$ by construction of the sets $J_b^{(k)}$. The value of X_ℓ will be useful to use the following claim.

Claim 4 *Let $i \in J$. Suppose that $V_{i,j} \notin W$ for all $j \in [n]$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{i,\ell}$, in other terms $c_i \leftarrow r_{i,\ell}$.*

Proof. The proof of this claim is quite straightforward since we suppose that $V_{i,j} \notin W$ for all $j \in [n]$, so none of the partial sums $V_{i,j}$ has been probed. Then V_i in c_i contains $n-1$ random values. In particular, we know that $r_{i,\ell}$ and $p_{i,\ell}$ only appear in the expression of the probed output c_i , because if they were probed in W then we would have $\ell \in J_b^{(i)}$ by construction, but we suppose that $\ell \notin J_b^{(k)}$ for all $k \in [n]$. In addition, since $X_{\ell,j} \notin W$ for all $j \in [n]$ (because otherwise then by construction $\ell \in J_b^{(k)}$ which does not hold), then $r_{i,\ell}$ does not appear in any other expression of the probed wires in W , so we can simply use it to perfectly simulate c_i . $\square$

We can now show that the sets W' and J can be perfectly simulated from the constructed sets I_a and $J_b^{(k)}$.

Simulation of W' . Probes of the form (a) can be perfectly simulated from the corresponding input shares in I_a and $J_b^{(k)}$, and by generating uniformly random values $r_{i,j}$ when necessary. Probes of the form (c) are also perfectly simulated by simply generating uniformly random values, since $X_{i,j} = r_{1,i} + \dots + r_{j,i}$. As for probes of the form (b), we know that $i \in I_a$, then we look at each of the terms $p_{i,j'}$ for $j' \in [j]$ in $V_{i,j} = p_{i,1} + \dots + p_{i,j}$. For each $p_{i,j'}$, if $j' \in J_b^{(i)}$, then $p_{i,j'}$ can be perfectly simulated from the corresponding input shares and by generating uniformly at random $r_{i,j'}$. Otherwise, if $j' \notin J_b^{(i)}$, then that means that the wires $p_{i,j'}, r_{i,j'}$ and $X_{j'}$ are not probed in W' . That means that we can potentially replace $p_{i,j'}$ by a random value $r_{i,j'}$ since $r_{i,j'}$ does not appear in any other expression of the variables probed in W' . Meanwhile, we also need to check the case where $j' \in J$, since $c_{j'} = V_{j'} + X_{j'}$, and $r_{i,j'}$ is the one of the summed terms in the expression of $X_{j'}$:

- If $j' \notin J$, then we can replace $p_{i,j'}$ by a random value $r_{i,j'}$ since $r_{i,j'}$ does not appear in any other expression of the variables probed in W' and is not probed either through $c_{j'}$.
- If $j' \in J$, then we also know that $V_{j'} \notin W'$ (because otherwise we would have by construction $j' \in J_b^{(k)}$ for $k \in [n]$ which does not hold), then we know from claim 4 that $c_{j'}$ can be masked by the random value $r_{j',\ell}$, which masks $V_{j'} + X_{j'}$. Since $\ell \neq j'$ (because $\ell \notin J$ while $j' \in J$), then $r_{i,j'}$ does not appear anymore in any other wire expression of the probed variables in W or J except in the term $p_{i,j'}$ of $V_{i,j}$, so $r_{i,j'}$ can be used to mask the expression of $p_{i,j'}$.

By perfectly simulating each term $p_{i,j'}$ in $V_{i,j}$, we can perfectly simulate $V_{i,j}$. Thus, we can perfectly simulate all wires in W' .

Simulation of J . Let $i \in J$. Let us first consider the case where $V_{i,j} \notin W'$ for any $j \in [n]$, then by claim 4, the output share c_i can be masked by the random variable $r_{i,\ell}$, so c_i is perfectly simulated by generating a fresh random value. Otherwise, if $V_{i,j} \in W'$ for a certain $j \in [n]$, then we know that the value of $V_{i,j}$ is perfectly simulated as proven above. Now, let us check each term $p_{i,j'}$ for $j' \in [j+1, n]$. Actually, we can also perfectly simulate each of these terms like the terms $p_{i,j'}$ for $j' \in [j]$. Plus, the term $p_{i,i}$ is perfectly simulated by construction of the sets I_a and $J_b^{(i)}$ (because $V_{i,j} \in W'$). In addition, all terms in X_i in $c_i = V_i + X_i$ can be perfectly simulated by generating a fresh random value. Thus, c_i can be perfectly simulated by summing all of the perfectly simulated terms in it. This brings us to a perfect simulation of all output shares in J . We have shown that we can perfectly simulate any set of probes W of size at most $\min(t+1, n-t) - 1$ with a chosen set J of $n-1$ output shares, with at most $\min(|W|, t) = |W|$ shares of each of the inputs a and b . This concludes the proof of Lemma 18. $\square$

Remark 3. We can observe that for this lemma to apply on G_{mult} , we don't need the pre-processing phase of the refresh on input b . In fact, you can see that during the construction of the sets $J_b^{(k)}$, we add each index to all of the sets for all $k \in [n]$. However, executing n refreshings on the input b will be necessary for the proof of the next result, specifically when we consider W such that $\min(t+1, n-t) \leq |W| < 2 \cdot \min(t+1, n-t)$.

To get back to the proof of Lemma 17, we also need the following result.

Lemma 19. *Let W be a set of leaking wires as described above such that $\min(t+1, n-t) \leq |W| < 2 \cdot \min(t+1, n-t)$. Then there exists a set J of $n-1$ output shares such that W and J can be perfectly simulated from sets of input shares I_a and I_b such that $|I_a| \leq \min(|W|, t)$ **or** $|I_b| \leq \min(|W|, t)$. In other terms, we have a simulation failure on at most one of the inputs a or b .*

Proof. Recall that the set W can be split into subsets $W = W' \cup W^{(1)} \cup \dots \cup W^{(n)}$ as described above. We consider two cases.

Case 1: $|W'| < \min(t+1, n-t)$. This case is similar to the case of Lemma 18, so we can construct the set I_a in the same way as in the proof of Lemma 18, and we can eventually consider $I_b = [n]$. We know that $|I_a| \leq |W'| \leq \min(t+1, n-t) - 1 \leq t$, so there is no failure on the input a . And all probes in W' can be simulated like in the proof of Lemma 18 with I_a and trivially with $I_b = [n]$. Also, all probes in $W^{(1)} \cup \dots \cup W^{(n)}$ can be trivially simulated since we have access to the full input b . In addition, we choose the set J of size $n-1$ in the same way as in Lemma 18. Whenever $i \in J$ and $V_{i,j} \in W'$ for some $j \in [n]$, then $c_i = V_i + X_i$ is easily simulated using $I_b = [n]$ and the share a_i . If $i \in J$ but $V_{i,j} \notin W'$ for all $j \in [n]$, then as in the proof of Lemma 18, c_i in this case can be masked by the random value $r_{i,\ell}$ (because $|W'| < \min(t+1, n-t)$) and so simulating c_i amounts to generating uniformly at random the corresponding random value. Thus, W and J are perfectly simulated with at most $\min(|W|, t)$ shares of a and eventually the full input b .

Case 2: $|W'| \geq \min(t+1, n-t)$ (and thus $|W^{(1)} \cup \dots \cup W^{(n)}| < \min(t+1, n-t)$). In this case, we will construct the sets I_a and $J_b^{(k)}$ from empty sets, in a way that we will have a simulation failure on at most one of the inputs a or b . We construct the mentioned sets depending on the probes in W' as follows:

- (a) For probes of form (a), we add index i to I_a , and index j only to $J_b^{(i)}$.
- (b) For probes of form (b), we add index i to I_a and only to $J_b^{(i)}$.
- (c) For probes of form (c), we add index i to $J_b^{(k)}$ for all $k \in [n]$.

In the rest of the lemma, we will prove that if we have a failure on one of the inputs, we can still perfectly simulate W and a chosen set J of $n-1$ output shares without a failure on the other input. For this, we will consider two cases, the first where we have a failure on input a , the second where we don't have a failure on input a , and so we can eventually have a failure on input b .

Case 2.1: simulation failure on input a , i.e. $|I_a| > t$. This means that the set I_a is of size $|I_a| \geq t+1 \geq \min(t+1, n-t)$ (this is because by construction $|I_a| \leq |W|$, so to have $|I_a| > \min(|W|, t)$, we must have $|I_a| > t$). We will first start by showing that the sets $W^{(k)}$ and $J_b^{(k)}$ can be perfectly simulated using the simulator of G_{refresh} without a failure on the input b . Next, we will show that W' and a well chosen set of $n-1$ output shares in J can be perfectly simulated using I_a and $J_b^{(k)}$.

Since we only add shares indices to I_a , when we have probes of the form (a) or (b), this means that we have at least $t+1$ probes of these two forms with $t+1$ different values for the index i . In addition, since we have at least $t+1$ probes (a) or (b) with distinct values for the index i , then this also means that each of the sets $J_b^{(i)}$ has at most one share of $b^{(i)}$ added to it. In other terms, $|J_b^{(k)}| \leq 1$ for each $k \in [n]$ (from the probes (a) and (b) with distinct indices i).

Now let us consider the remaining probes in W which are either in W' of the form (c) or in $W^{(1)} \cup \dots \cup W^{(n)}$ or of the forms (a) or (b) with $i \in I_a$. Since $|I_a| \geq t+1 \geq \min(t+1, n-t)$, then there are at most $\min(t+1, n-t) - 1$ of these remaining probes. Without loss of generality, we consider that there are exactly $\min(t+1, n-t) - 1$ instead of at most $\min(t+1, n-t) - 1$ probes. Let m be the number of probes in $W^{(1)} \cup \dots \cup W^{(n)}$ and $d-1-m$ the remaining probes in W' of the form (c) or (a)/(b) with $i \in I_a$. Since each wire in W' of the form (c) or (a)/(b) with $i \in I_a$ results in adding at most one more share index to each $J_b^{(k)}$ for $k \in [n]$, then we have

$|J_b^{(k)}| \leq 1 + (\min(t+1, n-t) - 1 - m) = d - \min(t+1, n-t)$. And $|W^{(1)} \cup \dots \cup W^{(n)}| \leq m$, in particular $|W^{(k)}| \leq m$ for any $k \in [n]$.

- if $m = 0$, then $W^{(k)} = \emptyset$ for any $k \in [n]$, and $|J_b^{(k)}| \leq \min(t+1, n-t) \leq \lfloor \frac{n+1}{2} \rfloor \leq n-1$, so by the TRPE property of G_{refresh} for any $t \leq n-1$, all of the $J_b^{(k)}$ sets can be perfectly simulated with no knowledge of the input shares of b since $W^{(k)} = \emptyset$, so $I_b^{(k)} = \emptyset$. Hence, $I_b = I_b^{(1)} \cup \dots \cup I_b^{(n)} = \emptyset$ and we have no simulation failure on the input b .
- if $m > 0$, then $|J_b^{(k)}| \leq \min(t+1, n-t) - 1 \leq t$ and $|W^{(1)} \cup \dots \cup W^{(n)}| < \min(t+1, n-t)$, in particular $|W^{(k)}| \leq \min(t+1, n-t) - 1$ for each $k \in [n]$. Thus, by the (t, f) -TRPE property of the refresh gadget G_{refresh} achieving the amplification order d , we can perfectly simulate both sets for each $k \in [n]$ with $I_b^{(k)}$ such that $|I_b^{(k)}| \leq |W^{(k)}|$. Thus, we can let $I_b = I_b^{(1)} \cup \dots \cup I_b^{(n)}$ so we can have $|I_b| \leq |W^{(1)} \cup \dots \cup W^{(n)}| \leq \min(t+1, n-t) - 1 \leq t$, and we can perfectly simulate $W^{(1)} \cup \dots \cup W^{(n)}$ along with $J_b^{(1)} \cup \dots \cup J_b^{(n)}$ from the set I_b without a simulation failure on input b .

So far we proved that if we have $|I_a| > t$, then we must have $|I_b| \leq t$, and $W^{(1)} \cup \dots \cup W^{(n)}$ can be perfectly simulated along with $J_b^{(1)} \cup \dots \cup J_b^{(n)}$ from the set I_b . Next we need to prove that we can perfectly simulate W' and a chosen set J of $n-1$ output shares, from these sets I_a and $J_b^{(1)} \cup \dots \cup J_b^{(n)}$. We consider two sub-cases.

Case 2.1.1: $I_a = [n]$. In this case, since $|W| \geq n$ (from I_a) and $|W| < 2\min(t+1, n-t) \leq n+1$, then $|W| = n$ and all probes in W are all in W' of the form (a) or (b) with n distinct values for the index i . We neither have probes in $W^{(1)} \cup \dots \cup W^{(n)}$ nor in W' of the form (c). Thus, we can reconstruct each $|J_b^{(k)}|$ of size at most $n-1$ without having a failure on the input b (since G_{refresh} is (t', f') -TRPE for any $t' \leq n-1$ achieving $d' = \min(t'+1, n-t')$ and all $W^{(k)}$ are empty). We consider two cases:

- Suppose that for each $i \in [n]$, we have at least one probe in W' of the form $r_{k,i}$ or $p_{k,i}$ for some $k \in [n]$, note this probe $q_{k,i} \in \{r_{k,i}, p_{k,i}\}$. Since also $I_a = [n]$, this means that we have probes $q_{k_1,1}, \dots, q_{k_n,n}$, such that $k_1 \neq \dots \neq k_n$. Because $|W'| = n$, then all probes in W' are of the form (a) (specifically $q_{k,i}$), and we have no probes of the form $V_{i,j}$ for any $i, j \in [n]$. In this case, the simulation of the probes in W' is straightforward by construction of the sets I_a and $J_b^{(k)}$. As for the set J , we let $J \subset [n]$ such that $|J| = n-1$ (any set of $n-1$ shares works), and let $\ell \in [n]$ such that $\ell \notin J$. Observe that out of all the random values $r_{i,\ell}$ in X_ℓ in the expression of $c_\ell = V_\ell + X_\ell$, only the random value $r_{k_\ell,\ell}$ appears in the expression of the probe $q_{k_\ell,\ell}$ in the set W' , and all other random values $r_{i,\ell}$ for $i \neq k_\ell$ do not appear in any other probed variable in W' (since $W' = \{q_{k_1,1}, \dots, q_{k_\ell,\ell}, \dots, q_{k_n,n}\}$, such that $k_1 \neq \dots \neq k_n$). Then, for each $i \in J$ such that $i \neq k_\ell$, the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{i,\ell}$, so simulating c_i amounts to generating a fresh random value $r_{i,\ell}$. Now let's check $i = k_\ell \in J$. Since $q_{k_\ell,\ell}$ is probed, then we cannot mask the expression of c_{k_ℓ} using $r_{k_\ell,\ell}$. However, for each $i \in J$ with $i \neq k_\ell$, we have that c_i is masked by $r_{i,\ell}$. Since $c_i = V_i + X_i$, and the random value $r_{k_\ell,i}$ is one of the terms in X_i , then $r_{k_\ell,i}$ does not appear anymore in the expression of c_i . And since $W' = \{q_{k_1,1}, \dots, q_{k_\ell,\ell}, \dots, q_{k_n,n}\}$, such that $k_1 \neq \dots \neq k_n$ and $q_{k_\ell,\ell} \in W'$, then $q_{k_\ell,i} \notin W'$ and $r_{k_\ell,i}$ only appears in the expression of c_{k_ℓ} , so c_{k_ℓ} can be masked by the random value $r_{k_\ell,i}$. Thus, we proved that we can perfectly simulate the sets W' and J using the sets I_a and $J_b^{(k)}$.

– Next, we suppose that there exists $\ell \in [n]$ such that we have no probes in W' of the form $q_{k,\ell} \in \{r_{k,\ell}, p_{k,\ell}\}$. In this case, we choose $J = [n] \setminus \{\ell\}$. Next, we show that we can perfectly simulate all probes in W' and output shares in J for each $i \in I_a = [n]$. For this, first let $i \in [n] \setminus \{\ell\}$ (notice that we automatically have $i \in J$):

- if for the considered i , the probe in W' is of the form (a) i.e $a_i, b_j^{(i)}, a_i \cdot b_j^{(i)}, p_{i,j} = a_i \cdot b_j^{(i)} + r_{i,j}$, then the simulation of this probe is trivial by construction of the sets I_a and $J_b^{(i)}$. In addition, we know that $r_{i,\ell}$ and $p_{i,\ell}$ are not probed in W' by assumption, and since $X_{\ell,j} \notin W'$ for all $j \in [n]$, then the random value $r_{i,\ell}$ only appears in the expression of $c_i = V_i + X_i$ (specifically in V_i), and so can be used to mask c_i . So simulating c_i amounts to generating a fresh random value.
- if for the considered i , the probe in W' is of the form (b), i.e $V_{i,j} \in W'$ for a certain $j \in [n]$ (there is a unique probe of this form), then:
 - * either $j < \ell$, and so the random value $r_{i,\ell}$ can be used as before to mask the expression of $c_i + V_{i,j}$, and since in this case $V_{i,j}$ contains less than $n - 1$ terms $p_{i,j'}$, then we can add all the necessary shares of $b^{(i)}$ to $J_b^{(i)}$ without having a failure on b (recall that $W^{(i)} = \emptyset$). So we can perfectly simulate $V_{i,j}$ and $c_i + V_{i,j}$, and hence also simulate c_i .
 - * or $j \geq \ell$, and so the random value $r_{i,\ell}$ can be used in this case to mask the expression of $V_{i,j}$ so simulating $V_{i,j}$ amounts to generating a fresh random value, and since $V_{i,j}$ is the sum of at least two terms of the form $p_{i,j'}$, then $c_i + V_{i,j}$ can be simulated with at most $n - 1$ shares of $b^{(i)}$, so there is no simulation failure on input $b^{(i)}$. So we can perfectly simulate $V_{i,j}$ and $c_i + V_{i,j}$, and hence also simulate c_i .

Next we consider the case of the probe $V_{\ell,j}$:

- either $j < \ell$, and so in this case $V_{\ell,j}$ contains less than $n - 1$ terms $p_{\ell,j'}$, then we can add all the necessary shares of $b^{(\ell)}$ to $J_b^{(\ell)}$ without having a failure on b (recall that $W^{(\ell)} = \emptyset$). So we can perfectly simulate $V_{\ell,j}$ using the input share a_ℓ , the input shares of $b^{(\ell)}$ and by generating necessary random values.
- or $j \geq \ell$, and so the random value $r_{\ell,\ell}$ can be used in this case to mask the expression of $V_{\ell,j}$ so simulating $V_{\ell,j}$ amounts to generating a fresh random value.

Thus, also in this case, we can perfectly simulate W' and a chosen set of $n - 1$ output shares without a failure on input b , using $I_a = [n]$.

This concludes the simulation for the special case where $I_a = [n]$.

Case 2.1.2: $I_a \subset [n]$ such that $|I_a| \leq n - 1$. Let k such that $k \notin I_a$. Recall that $|I_a| \geq t + 1 \geq \min(t + 1, n - t)$ and $|W| < 2 \cdot \min(t + 1, n - t)$, then there are at most $\min(t + 1, n - t) - 1 \leq t \leq n - 1$ probes remaining either in $W^{(1)} \cup \dots \cup W^{(n)}$, of the form (c) in W' , or of the form (a)/(b) with $i \in I_a$. Thus, there exists at least one index $\ell \in [n]$ such that $X_{\ell,j} \notin W'$ for all $j \in [n]$. In this case, we choose $J = [n] \setminus \{\ell\}$. Next, we will prove that we can perfectly simulate the sets W' and J from the constructed sets I_a and $J_b^{(k)}$, using the following claims.

Claim 5 *Let $i \in J$. Suppose that $i \notin I_a$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{i,\ell}$, in other terms $c_i \leftarrow r_{i,\ell}$.*

Proof. This claim can be proved easily, since we suppose that $i \notin I_a$ so the random value $r_{i,\ell}$ and $p_{i,\ell}$ are not probed in W' . In addition, since $\ell \notin J$ and $X_{\ell,j} \notin W$ for all $j \in [n]$, then the random value $r_{i,\ell}$ does not appear in any other probed wire expression except in c_i , then c_i can be masked by the random value $r_{i,\ell}$. $\square$

Claim 6 *Let $i \in J$. Suppose that $X_{i,j} \notin W$ for any $j \in [n]$. Suppose that $i \in I_a$. Then the expression of $c_i = V_i + X_i$ can be masked by the random value $r_{k,i}$, in other terms $c_i \leftarrow r_{k,i}$.*

Proof. Since we suppose that $k \notin I_a$, then the random value $r_{k,i}$ or $p_{k,i}$ or $V_{k,j}$ for all $j \in [n]$ are not probed in W . Then, if $k \notin J$, then the random value $r_{k,i}$ does not appear in the expression of any other probed wire in W or J and c_i can be masked by the random value $r_{k,i}$ (Recall that $c_i = V_i + X_i$ and $X_i = r_{1,i} + \dots + r_{n,i}$). Otherwise, if $k \in J$, then by Claim 5, $c_k = V_k + X_k$ can be masked by $r_{k,\ell}$ and so c_i can also be masked by $r_{k,i}$ since $i \neq \ell$ (because $i \in J$ and $\ell \notin J$) and $i \neq k$ (because $i \in I_a$ and $k \notin I_a$). $\square$

Probes of the forms (a) or (c) in W' are trivially simulated using the constructed sets of input shares, and generating the necessary random values. Let us now check the probes of the form (b). Let $V_{i,j} = p_{i,1} + \dots + p_{i,j}$ be such a probe. Let us consider each of the terms $p_{i,j'}$ for $j' \in [j]$. if $j' = i$, then by construction $p_{i,i}$ is perfectly simulated using a_i and $b_i^{(i)}$ and by generating the random value $r_{i,i}$ if needed. Otherwise, let $j' \neq i$. If $j' \in J_b^{(i)}$ then the simulation of $p_{i,j'}$ is straightforward. Otherwise if $j' \notin J_b^{(i)}$, then we know that none of the wires $r_{i,j'}$ or $p_{i,j'}$ or $X_{j',s}$ for all $s \in [n]$ are probed in W' . Thus, $r_{i,j'}$ can be eventually used to mask the expression of $p_{i,j'}$ without the need of the share $b_{j'}^{(i)}$ for the simulation. Meanwhile, we still need to check if $j' \in J$, since $r_{i,j'}$ appears in $X_{j'}$ in the expression of $c_{j'} = V_{j'} + X_{j'}$.

- If $j' \in J$ and $j' \notin I_a$, then by claim 5, $c_{j'}$ can be masked by the random value $r_{j',\ell}$ and so $r_{i,j'}$ does not appear in the expression of $X_{j'}$ in $c_{j'}$ anymore, and $r_{i,j'}$ can be used to mask $p_{i,j'}$.
- Otherwise, if $j' \in J \cap I_a$, then by claim 6 $c_{j'}$ can be masked by the random value $r_{k,j'}$ and so $r_{i,j'}$ does not appear in the expression of $X_{j'}$ in $c_{j'}$ anymore, and $r_{i,j'}$ can be used to mask $p_{i,j'}$ (since $i \notin k$).

Thus, each term $p_{i,j'}$ in $V_{i,j}$ can be perfectly simulated and thus $V_{i,j} = p_{i,1} + \dots + p_{i,j}$ can be perfectly simulated. This concludes the simulation of the set W' .

We now focus on the simulation of J . Let $i \in J$. If $i \notin I_a$, then by claim 5, c_i is perfectly simulated by generating the random value $r_{i,\ell}$. Otherwise, let $i \in I_a$. If $X_{i,j} \notin W$ for any $j \in [n]$, then by claim 6, c_i is perfectly simulated by generating the random value $r_{k,i}$. Otherwise, we can show that we can perfectly simulate each term in $c_i = V_i + X_i$. In particular, each term in X_i can be simulated by generating the underlying random value uniformly. For V_i , we know in particular that $a_i \cdot b_i^{(i)}$ is perfectly simulated since $X_{i,j} \in W'$ for at least one $j \in [n]$ so $i \in J_b^{(i)}$ by construction. For the other terms in V_i , they can be perfectly simulated in the exact same way as we simulated the probes $V_{i,j}$ of the form (b) in the set W' . So c_i is perfectly simulated by summing all the perfectly simulated terms. This concludes the simulation proof for the set J .

Up until now, we have concluded that if we have a constructed set I_a of size at least $t + 1$, then we can perfectly simulate the sets W and a chosen set J of $n - 1$ output shares, without having a simulation failure on the input b . In the rest of the proof, we will consider that $|I_a| \leq t$, and we will prove that we can perfectly simulate W and J with at most a simulation failure on b . Recall that we are also considering that $|W'| \geq d$ and $|\mathbf{W}^{(1)} \cup \dots \cup \mathbf{W}^{(n)}| \leq \mathbf{d} - 1$.

Case 2.2: $|I_a| \leq t$. This means that the number of probes of the form (a) or (b) in W' with distinct values for the index i is at most t .

First, let us check the special case where the number of probes of the form (c) in W' with different values for the index i is equal to n (notice that this cannot occur when we have $|I_a| \geq t+1$). Since $|W| \leq 2 \cdot \min(t+1, n-t) - 1 \leq n$, then we have $W = \{X_{1,j_1}, \dots, X_{n,j_n}\}$ for certain $j_1, \dots, j_n \in [n]$. So we can let $J_b^{(k)} = [n]$ for all $k \in [n]$ and $I_b = [n]$, and by construction $I_a = \emptyset$. In this case, we choose $J = [n-1]$. The simulation of the set W is straightforward since all wires of the form (c) are just sums of random values. Then, let us consider the output shares in J .

- If for at least one $\ell \in J$, we have $X_{\ell,n} \in W$, we can mask the expression of $X_{\ell,n}$ by the random value $r_{n,\ell}$ (because there are no probes of the form (a) or (b) in W and $n \notin J$, so $r_{n,\ell}$ only appears in the expression of $X_{\ell,n}$). Recall that $X_{\ell,n} = r_{1,\ell} + \dots + r_{n,\ell}$, so $r_{n,\ell}$ masks all random values $r_{j,\ell}$ for $j \in [n-1]$. Each of the random values $r_{j,\ell}$ for $j \in [n-1] \setminus \{\ell\}$ can be used to mask the corresponding output share c_j for $j \in J$ because there are no probes of the form (a) or (b) in W and $X_{\ell,n}$ is already masked by $r_{n,\ell}$, so $r_{j,\ell}$ only appears in the expression of $c_j = V_j + X_j$, so $c_j \leftarrow r_{j,\ell}$. As for the output c_ℓ , we can let $I_a = \{\ell\}$ and we can perfectly simulate c_ℓ using a_ℓ and $I_b = [n]$. Since $|W| = n$, so $\min(t+1, n-t) > \frac{n}{2} \geq 1$, so we have no failure on the input a , and we can perfectly simulate the chosen set J and the set of probes W .
- Now we consider that for all $W = \{X_{1,j_1}, \dots, X_{n,j_n}\}$, we have $j_1 < n, \dots, j_n < n$. In this case, the set W is also trivially simulated by generating random values, and we let $J = [n-1]$. Since, $n \notin J$ and there are no probes of the form (a) or (b) in W , then the random values $r_{n,i}$ for $i \in [n-1]$ only appear in the expression of the output share $c_i = V_i + X_i$ each. And since all probes of the form $X_{i,j}$ are such that $j < n$, then we can let $r_{n,i}$ be used to mask the expression of $c_i + X_{i,j}$ because $r_{n,i}$ does not appear in $X_{i,j}$ for $j < n$, i.e $c_i + X_{i,j} \leftarrow r_{n,i}$. By perfectly simulating the masked expression of $c_i + X_{i,j}$ and the sum of random values $X_{i,j}$, we can perfectly simulate c_i . Thus, simulating all output shares in J amounts to generating random values uniformly. So we can perfectly simulate sets W and J from $I_a = \emptyset$ and $I_b = [n]$.

Next, we suppose that the number of probes of the form (c) in W' with different values for the index i is strictly smaller than n . So, there is at least one index ℓ such that $X_{\ell,j} \notin W'$ for all $j \in [n]$. We let $J = [n] \setminus \{\ell\}$. We also let $I_b = [n]$ and we keep the set I_a as constructed according to the probes in the set W' . Observe that all probes in W' are perfectly simulated by easily using the set I_a and $I_b = [n]$. As for the output shares in J , observe that for each $i \in J$ such that $i \notin I_a$, we can use claim 5 to mask the expression of c_i by $r_{i,\ell}$, and so the share a_i is not needed for the simulation of c_i . Otherwise, if $i \in J \cap I_a$, then c_i is perfectly simulated using a_i and $I_b = [n]$.

This proves that whenever $i \notin I_a$, the output share c_i can be simulated without the need of the share a_i . Since we suppose that $|I_a| \leq t$, then we conclude that we can perfectly simulate W and a chosen set of $n-1$ output shares J with at most a simulation failure on input b .

By considering both cases $|I_a| \geq t+1$ and $|I_a| \leq t$, we covered all the cases for the simulation, and we proved that we can always perfectly simulate the set of probes W along with a chosen set of $n-1$ output shares J while having a failure on at most one of the inputs. This concludes the proof of Lemma 19. $\square$

From Lemmas 18 and 19, we conclude that G_{mult} is (t, f_2) -TRPE2 of amplification order $d \geq \min(t+1, n-t)$. This concludes the proof of Lemma 17. $\square$

Appendix C

Dynamic Random Probing Expansion with Quasi Linear Asymptotic Complexity

Hereafter is appended the full version of our paper [\[BRTV21\]](#), joint work with Matthieu Rivain, Abdul Rahman Taleb, and Damien Vergnaud published at **ASIACRYPT 2021**.

Dynamic Random Probing Expansion with Quasi Linear Asymptotic Complexity

Sonia Belaïd¹, Matthieu Rivain¹, Abdul Rahman Taleb^{1,2}, and Damien Vergnaud^{2,3}

¹ CryptoExperts, France

² Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

³ Institut Universitaire de France, France

{sonia.belaïd,matthieu.rivain,abdul.taleb}@cryptoexperts.com

⁴ damien.vergnaud@lip6.fr

Abstract. The masking countermeasure is widely used to protect cryptographic implementations against side-channel attacks. While many masking schemes are shown to be secure in the widely deployed probing model, the latter raised a number of concerns regarding its relevance in practice. Offering the adversary the knowledge of a fixed number of intermediate variables, it does not capture the so-called horizontal attacks which exploit the repeated manipulation of sensitive variables. Therefore, recent works have focused on the *random probing model* in which each computed variable leaks with some given probability p . This model benefits from fitting better the reality of the embedded devices. In particular, Belaïd, Coron, Prouff, Rivain, and Taleb (CRYPTO 2020) introduced a framework to generate random probing circuits. Their compiler somehow extends base gadgets as soon as they satisfy a notion called *random probing expandability* (RPE). A subsequent work from Belaïd, Rivain, and Taleb (EUROCRYPT 2021) went a step forward with tighter properties and improved complexities. In particular, their construction reaches a complexity of $\mathcal{O}(\kappa^{3.9})$, for a κ -bit security, while tolerating a leakage probability of $p = 2^{-7.5}$.

In this paper, we generalize the random probing expansion approach by considering a dynamic choice of the base gadgets at each step in the expansion. This approach makes it possible to use gadgets with high number of shares –which enjoy better asymptotic complexity in the expansion framework– while still tolerating the best leakage rate usually obtained for small gadgets. We investigate strategies for the choice of the sequence of compilers and show that it can reduce the complexity of an AES implementation by a factor 10. We also significantly improve the asymptotic complexity of the expanding compiler by exhibiting new asymptotic gadget constructions. Specifically, we introduce RPE gadgets for linear operations featuring a quasi-linear complexity as well as an RPE multiplication gadget with linear number of multiplications. These new gadgets drop the complexity of the expanding compiler from quadratic to quasi-linear.

Keywords: Random probing model, masking, side-channel security, RPE

1 Introduction

Implementations of cryptographic algorithms may be vulnerable to the powerful *side-channel attacks*. The latter exploit the power consumption, the electromagnetic radiations or the temperature variations of the underlying device which may carry information on the manipulated data. Entire secrets can be recovered within a short time interval using cheap equipment.

Among the several approaches investigated by the community to counteract side-channel attacks, *masking* is one of the most deployed in practice. Simultaneously introduced by Chari, Jutla, Rao, and Rohatgi [12] and by Goubin and Patarin [16] in 1999, it consists in splitting the sensitive variables into n random shares, among which any combination of $n - 1$ shares does not reveal any secret information. When the shares are combined by bitwise addition, the masking is said to be *Boolean*. In this setting, the linear operations can be very easily implemented by applying on each

share individually. Nevertheless, non-linear operations require additional randomness to ensure that any set of less than n intermediate variables is still independent from the original secret.

To reason on the security of masked implementations, the community has introduced so-called *leakage models*. They aim to define the capabilities of the attacker to formally counteract the subsequent side-channel attacks. Among them, the *probing model* introduced in 2003 by Ishai, Sahai, and Wagner [18] is probably the most widely used. In a nutshell, it assumes that an adversary is able to get the exact values of up to a certain number of intermediate variables. The idea is to capture the difficulty of learning information from the combination of noisy variables. Despite its wide use by the community [21, 20, 13, 7, 14], the probing model raised a number of concerns regarding its relevance in practice [5, 17]. It actually fails to capture the huge amount of information resulting from the leakage of all manipulated data. As an example, it typically ignores the repeated manipulation of identical values which would average the noise and remove uncertainty on secret variables (see horizontal attacks [5]). Another model, the *noisy leakage model* introduced by Prouff and Rivain and inspired from [12], offers an opposite trade-off. Although it captures well the reality of embedded devices by assuming that all the data leaks with some noise, it is not convenient to build security proofs. To get the best from both worlds, Duc, Dziembowski, and Faust proved in 2014 that a scheme secure in the probing model is also secure in the noisy leakage model [15]. Nevertheless, the reduction is not very tight in the standard probing model (considering a constant number of probes) since the security level decreases as the size of the circuit increases (*i.e.* a secure circuit C in the probing model is also secure in the noisy model but loses at least a factor $|C|$, where $|C|$ is the number of operations in the circuit).

The reduction from [15] relies on an intermediate leakage model, referred to as *random probing model*. The latter benefits from a tight reduction with the noisy leakage model which becomes independent of the size of the circuit. In a nutshell, it assumes that every wire in the circuit leaks with some constant leakage probability. This leakage probability is somehow related to the amount of side-channel noise in practice. A masked circuit is secure in the random probing model whenever its random probing leakage can be simulated without knowledge of the underlying secret data with a negligible simulation failure. In addition to the attacks already captured by the probing model, the random probing model further encompasses the powerful *horizontal attacks* which exploit the repeated manipulations of variables in an implementation.

To the best of our knowledge, five constructions tolerate a constant leakage probability so far [1, 4, 3, 9, 10]. The two former ones [1, 4] use expander graphs and do not make their tolerated probability explicit. In the third construction [3], Ananth, Ishai, and Sahai develop an expansion strategy on top of multi-party computation protocols. According to the authors of [9], their construction tolerates a leakage probability of around 2^{-26} for a complexity of $\mathcal{O}(\kappa^{8.2})$ with respect to the security parameter κ . Finally, the two more recent constructions [9, 10] follow an expansion strategy on top of masking gadgets achieving the so-called *random probing expandability* (RPE) notion. In a nutshell, every gate in the original circuit is replaced by a corresponding gadget for some chosen number of shares. The operation is repeated until the desired security level is achieved. The improved gadgets of [10] make it possible to tolerate a leakage probability of $2^{-7.5}$ for a complexity of $\mathcal{O}(\kappa^{3.9})$.

Our contributions. In this paper, we push the random probing expansion strategy one step further by analyzing a dynamic choice of the base gadgets. While the expanding compiler considered in [9, 10] consists in applying a compiler CC composed of base RPE gadgets a given number of times, say k , to the input circuit: $\hat{C} = CC^{(k)}(C)$, we consider a dynamic approach in which a new compiler

is selected at each step of the expansion from a family of base compilers $\{\text{CC}_i\}_i$. This approach is motivated by the generic gadget constructions introduced in [10] which achieve the RPE property for any number of shares n . While the asymptotic complexity of the expanding compiler decreases with n , the tolerated leakage probability p also gets smaller with n , which makes those constructions only practical for small values of n . We show that using our dynamic approach we can get the best of both worlds: our dynamic expanding compiler enjoys the best tolerated probability as well as the best asymptotic complexity from the underlying family of RPE compilers $\{\text{CC}_i\}_i$. We further illustrate how this approach can reduce the complexity of a random probing secure AES implementation by a factor 10 using a dynamic choice of the gadgets from [10].

This first contribution further motivates the design of asymptotic RPE gadgets achieving better complexity. While the asymptotic constructions introduced in [10] achieve a quadratic complexity, we introduce new constructions achieving quasi-linear complexity. We obtain this result by showing that the quasi-linear refresh gadget from Battistello, Coron, Prouff, and Zeitoun [6] achieves a *strong random probing expandability* (SRPE) which makes it a good building block for linear RPE gadgets (addition, copy, multiplication by constant). We thus solve a first issue left open in [10]. With such linear gadgets, the complexity bottleneck of the expanding compiler becomes the number of multiplications in the multiplication gadget, which is quadratic in known RPE constructions. We then provide a new generic construction of RPE multiplication gadget featuring a linear number of multiplications. We obtain this construction by tweaking the probing-secure multiplication gadget from Belaïd, Benhamouda, Passelègue, Prouff, Thillard, and Vergnaud [8]. As in the original construction, our RPE gadget imposes some constraint on the underlying finite field. We demonstrate that for any number of shares there exist a (possibly large) finite field on which our construction can be instantiated and we provide some concrete instantiations for some (small) number of shares.

Using our new asymptotic gadget constructions with the dynamic expansion approach we obtain random probing security for a leakage probability of $2^{-7.5}$ with asymptotic complexity of $\mathcal{O}(\kappa^2)$. Moreover, assuming that the constraint on the finite field from our multiplication gadget is satisfied, we can make this asymptotic complexity arbitrary close to $\mathcal{O}(\kappa)$ which is optimal. In practice, this means that securing circuits defined on large field against random probing leakage can be achieved at a sub-quadratic nearly-linear complexity.

2 Preliminaries

Along the paper, we shall use similar notations and formalism as [9]. In particular, $\mathbb{K}$ shall denote a finite field. For any $n \in \mathbb{N}$, we shall denote $[n]$ the integer set $[n] = [1, n] \cap \mathbb{Z}$. For any tuple $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{K}^n$ and any set $I \subseteq [n]$, we shall denote $\mathbf{x}|_I = (x_i)_{i \in I}$. Any two probability distributions D_1 and D_2 are said ε -close, denoted $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε , that is

$$\text{SD}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon ,$$

where $p_{D_1}(\cdot)$ and $p_{D_2}(\cdot)$ denote the probability mass functions of D_1 and D_2 .

2.1 Linear Sharing, Circuits, and Gadgets

In the following, the n -linear decoding mapping, denoted LinDec , refers to the function $\mathbb{K}^n \rightarrow \mathbb{K}$ defined as

$$\text{LinDec} : (x_1, \dots, x_n) \mapsto x_1 + \dots + x_n ,$$

for every $n \in \mathbb{N}$ and $(x_1, \dots, x_n) \in \mathbb{K}^n$. We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $(\hat{x}_1, \dots, \hat{x}_\ell) \in (\mathbb{K}^n)^\ell$ the n -linear decoding mapping acts as

$$\text{LinDec} : (\hat{x}_1, \dots, \hat{x}_\ell) \mapsto (\text{LinDec}(\hat{x}_1), \dots, \text{LinDec}(\hat{x}_\ell)) .$$

Definition 1 (Linear Sharing). Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an n -linear sharing of x is a random vector $\hat{x} \in \mathbb{K}^n$ such that $\text{LinDec}(\hat{x}) = x$. It is said to be uniform if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $\hat{x}_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. A n -linear encoding is a probabilistic algorithm LinEnc which on input a tuple $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{K}^\ell$ outputs a tuple $\hat{\mathbf{x}} = (\hat{x}_1, \dots, \hat{x}_\ell) \in (\mathbb{K}^n)^\ell$ such that $\hat{x}_i$ is a uniform n -sharing of x_i for every $i \in [\ell]$.

An *arithmetic circuit* on a field $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations on $\mathbb{K}$. We consider circuits composed of gates from some base $\mathbb{B} = \{g : \mathbb{K}^\ell \rightarrow \mathbb{K}^m\}$, e.g., addition gates, $(x_1, x_2) \mapsto x_1 + x_2$, multiplication gates, $(x_1, x_2) \mapsto x_1 \cdot x_2$, and copy gates, $x \mapsto (x, x)$. A *randomized arithmetic circuit* is equipped with an additional random gate which outputs a fresh uniform random value of $\mathbb{K}$.

In the following, we shall call an $(n\text{-share}, \ell\text{-to-}m)$ *gadget*, a randomized arithmetic circuit that maps an input $\hat{\mathbf{x}} \in (\mathbb{K}^n)^\ell$ to an output $\hat{\mathbf{y}} \in (\mathbb{K}^m)^m$ such that $\mathbf{x} = \text{LinDec}(\hat{\mathbf{x}}) \in \mathbb{K}^\ell$ and $\mathbf{y} = \text{LinDec}(\hat{\mathbf{y}}) \in \mathbb{K}^m$ satisfy $\mathbf{y} = g(\mathbf{x})$ for some function g .

Definition 2 (Circuit Compiler). A circuit compiler is a triplet of algorithms $(\text{CC}, \text{Enc}, \text{Dec})$ defined as follows:

- **CC** (circuit compilation) is a deterministic algorithm that takes as input an arithmetic circuit C and outputs a randomized arithmetic circuit $\hat{C}$,
- **Enc** (input encoding) is a probabilistic algorithm that maps an input $\mathbf{x} \in \mathbb{K}^\ell$ to an encoded input $\hat{\mathbf{x}} \in \mathbb{K}^{\ell'}$,
- **Dec** (output decoding) is a deterministic algorithm that maps an encoded output $\hat{\mathbf{y}} \in \mathbb{K}^{m'}$ to a plain output $\mathbf{y} \in \mathbb{K}^m$,

which satisfy the following properties:

- **Correctness:** For every arithmetic circuit C of input length ℓ , and for every $\mathbf{x} \in \mathbb{K}^\ell$, we have

$$\Pr(\text{Dec}(\hat{C}(\hat{\mathbf{x}})) = C(\mathbf{x}) \mid \hat{\mathbf{x}} \leftarrow \text{Enc}(\mathbf{x})) = 1 , \text{ where } \hat{C} = \text{CC}(C).$$

- **Efficiency:** For some security parameter $\kappa \in \mathbb{N}$, the running time of $\text{CC}(C)$ is $\text{poly}(\kappa, |C|)$, the running time of $\text{Enc}(\mathbf{x})$ is $\text{poly}(\kappa, |\mathbf{x}|)$ and the running time of $\text{Dec}(\hat{\mathbf{y}})$ is $\text{poly}(\kappa, |\hat{\mathbf{y}}|)$, where $\text{poly}(\kappa, \ell) = \mathcal{O}(\kappa^{e_1} \ell^{e_2})$ for some constants e_1, e_2 .

2.2 Random Probing Security

Let $p \in [0, 1]$ be some constant leakage probability parameter, a.k.a. the *leakage rate*. In the p -random probing model, an evaluation of a circuit C leaks the value carried by each wire with a probability p , all the wire leakage events being mutually independent.

As in [9], we formally define the random-probing leakage of a circuit from the two following probabilistic algorithms:

- The *leaking-wires sampler* takes as input a randomized arithmetic circuit C and a probability $p \in [0, 1]$, and outputs a set W , denoted as

$$W \leftarrow \text{LeakingWires}(C, p) ,$$

where W is constructed by including each wire label from the circuit C with probability p to W (where all the probabilities are mutually independent).

- The *assign-wires sampler* takes as input a randomized arithmetic circuit C , a set of wire labels W (subset of the wire labels of C), and an input $\mathbf{x}$, and it outputs a $|W|$ -tuple $\mathbf{w} \in \mathbb{K}^{|W|}$, denoted as

$$\mathbf{w} \leftarrow \text{AssignWires}(C, W, \mathbf{x}) ,$$

where $\mathbf{w}$ corresponds to the assignments of the wires of C with label in W for an evaluation on input $\mathbf{x}$.

Definition 3 (Random Probing Leakage). *The p -random probing leakage of a randomized arithmetic circuit C on input $\mathbf{x}$ is the distribution $\mathcal{L}_p(C, \mathbf{x})$ obtained by composing the leaking-wires and assign-wires samplers as*

$$\mathcal{L}_p(C, \mathbf{x}) \stackrel{id}{=} \text{AssignWires}(C, \text{LeakingWires}(C, p), \mathbf{x}) .$$

Definition 4 (Random Probing Security). *A randomized arithmetic circuit C with $\ell \cdot n \in \mathbb{N}$ input gates is (p, ε) -random probing secure with respect to encoding Enc if there exists a simulator Sim such that for every $\mathbf{x} \in \mathbb{K}^\ell$:*

$$\text{Sim}(C) \approx_\varepsilon \mathcal{L}_p(C, \text{Enc}(\mathbf{x})) . \quad (1)$$

2.3 Random Probing Expansion

In [3], Ananth, Ishai and Sahai proposed an *expansion* approach to build a random-probing-secure circuit compiler from a secure multi-party protocol. This approach was later revisited by Belaïd, Coron, Prouff, Rivain, and Taleb who formalize the notion of *expanding compiler* [9].

The principle of the expanding compiler is to recursively apply a base compiler, denoted CC and which simply consists in replacing each gate of $\mathbb{B}$ in the input circuit by the corresponding gadget. Assume we have n -share gadgets G_g for each gate g in $\mathbb{B}$. The base compiler CC simply consists in replacing each gate g in these gadgets by G_g and by replacing each wire by n wires carrying a sharing of the value. We thus obtain n^2 -share gadgets by simply applying CC to each gadget: $G_g^{(2)} = \text{CC}(G_g)$. This process can be iterated an arbitrary number of times, say k , to an input circuit C :

$$C \xrightarrow{\text{CC}} \hat{C}_1 \xrightarrow{\text{CC}} \dots \xrightarrow{\text{CC}} \hat{C}_k .$$

The first output circuit $\hat{C}_1$ is the original circuit in which each gate g is replaced by a base gadget G_g . The second output circuit $\hat{C}_2$ is the original circuit C in which each gate is replaced by an n^2 -share gadget $G_g^{(2)}$. Equivalently, $\hat{C}_2$ is the circuit $\hat{C}_1$ in which each gate is replaced by a base gadget. In the end, the output circuit $\hat{C}_k$ is hence the original circuit C in which each gate has been replaced by a k -expanded gadget and each wire has been replaced by n^k wires carrying an (n^k) -linear sharing of the original wire.

The expanding compiler achieves random probing security if the base gadgets verify a property called *random probing expandability* [9]. We recall hereafter the original definition of the random probing expandability (RPE) property for 2-to-1 gadgets.

Definition 5 (Random Probing Expandability [9]). Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share 2-to-1 gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -random probing expandable (RPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^n \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the random experiment

$$\begin{aligned} W &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2, J') &\leftarrow \text{Sim}_1^G(W, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(W, J', \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (|I_1| > t)$ and $\mathcal{F}_2 \equiv (|I_2| > t)$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \quad (2)$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. J' is such that $J' = J$ if $|J| \leq t$ and $J' \subseteq [n]$ with $|J'| = n - 1$ otherwise,
3. the output distribution satisfies

$$\text{out} \stackrel{id}{=} (\text{AssignWires}(G, W, (\hat{x}, \hat{y})), \hat{z}|_{J'}) \quad (3)$$

where $\hat{z} = G(\hat{x}, \hat{y})$.

The RPE notion can be simply extended to gadgets with 2 outputs: the Sim_1^G simulator takes two sets $J_1 \subseteq [n]$ and $J_2 \subseteq [n]$ as input and produces two sets J'_1 and J'_2 satisfying the same property as J' in the above definition (w.r.t. J_1 and J_2). The Sim_2^G simulator must then produce an output including $\hat{z}_1|_{J'_1}$ and $\hat{z}_2|_{J'_2}$ where $\hat{z}_1$ and $\hat{z}_2$ are the output sharings. The RPE notion can also be simply extended to gadgets with a single input: the Sim_1^G simulator produces a single set I so that the failure event $(|I| > t)$ occurs with probability ε (and the Sim_2^G simulator is then simply given $\hat{x}|_I$ where $\hat{x}$ is the single input sharing). We refer the reader to [9] for the formal definitions of these variants.

Although the requirement of mutual independence for the failure events might seem strong, it can be relaxed which leads to the notion of *weak random probing expandability*. It is shown in [9] that this weaker notion actually implies the RPE notion for some ε which is derivable from the (joint) probability of the failure events.

The authors of [10] eventually introduced a tighter version the RPE security property, namely the tight random probing expandability (TRPE). In this setting, the failure events are re-define as $\mathcal{F}_j \equiv (|I_j| > \min(t, W))$. Both RPE and TRPE notions can be split into two sub-notions (that are jointly equivalent to the original one) corresponding to the two possible properties of J' in Definition 5. Specifically, in (T)RPE1, the set J is constrained to satisfy $|J| \leq t$ and $J' = J$, while in (T)RPE2, J' is chosen by the simulator such that $J' \subseteq [n]$ and $|J'| = n - 1$.

2.4 Complexity of the Expanding Compiler

Consider circuits with base of gates $\mathbb{B} = \{g_1, \dots, g_\beta\}$ for which we have n -share RPE gadgets $\{G_g\}_{g \in \mathbb{B}}$. Further denote G_{random} the n -share random gadget which generates n independent random values as a random n -sharing as well as CC the circuit compiler based from those gadgets. To

each gadget a complexity vector is associated $N_G = (N_{g_1}, \dots, N_{g_\beta}, N_r)^\top$ where N_{g_i} stands for the number of gates g_i and N_r for the number of random gates in the gadget G . Then the *compiler complexity matrix* M_{CC} is the $(\beta + 1) \times (\beta + 1)$ matrix defined as

$$M_{CC} = (N_{g_1} \mid \dots \mid N_{g_\beta} \mid N_{G_{\text{random}}}) \quad \text{with} \quad N_{G_{\text{random}}} = (0, \dots, 0, n)^\top.$$

Given a circuit C with complexity vector N_C (which is defined as the gate-count vector as for gadgets), compiling it with the base gadgets gives a circuit $\hat{C}$ of complexity vector $N_{\hat{C}} = M_{CC} \cdot N_C$. It follows that the k th power of the matrix M gives the gate counts for the level- k gadgets as:

$$M_{CC}^k = \underbrace{M_{CC} \cdots M_{CC}}_{k \text{ times}} = (N_{g_1}^{(k)} \mid \dots \mid N_{g_\beta}^{(k)} \mid N_{G_{\text{random}}}^{(k)}) \quad \text{with} \quad N_{G_{\text{random}}}^{(k)} = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ n^k \end{pmatrix}$$

where $N_{g_i}^{(k)}$ are the gate-count vectors for the level- k gadgets $G_{g_i}^{(k)}$. Let us denote the eigen decomposition of M_{CC} as $M_{CC} = Q \cdot \Lambda \cdot Q^{-1}$, we get

$$M_{CC}^k = Q \cdot \Lambda^k \cdot Q^{-1} \quad \text{with} \quad \Lambda^k = \begin{pmatrix} \lambda_1^k & & \\ & \ddots & \\ & & \lambda_{\beta+1}^k \end{pmatrix}$$

where λ_i are the eigenvalues of M_{CC} . We then obtain an asymptotic complexity of

$$|\hat{C}| = \mathcal{O}(|C| \cdot \sum_{i=1}^{\beta+1} |\lambda_i|^k) = \mathcal{O}(|C| \cdot \max(|\lambda_1|, \dots, |\lambda_{\beta+1}|)^k)$$

for a compiled circuit $\hat{C} = CC^{(k)}(C)$.

The complexity of the expanding compiler can be further expressed in terms of the target random probing security level κ . This complexity is related to the notion of *amplification order* that we recall hereafter.

Definition 6 (Amplification Order).

- Let $f : \mathbb{R} \rightarrow \mathbb{R}$ which satisfies

$$f(p) = c_d p^d + \mathcal{O}(p^{d+\varepsilon})$$

as p tends to 0, for some $c_d > 0$ and $\varepsilon > 0$. Then d is called the *amplification order* of f .

- Let $t > 0$ and G a gadget. Let d be the maximal integer such that G achieves (t, f) -RPE for $f : \mathbb{R} \rightarrow \mathbb{R}$ of amplification order d . Then d is called the *amplification order* of G (with respect to t). We will sometimes denote f_G as the function f corresponding to the gadget G for which G achieves (t, f_G) -RPE.

We stress that the amplification order of a gadget G is defined with respect to the RPE threshold t . Namely, different RPE thresholds t are likely to yield different amplification orders d for G (or equivalently d can be thought of as a function of t).

As shown in [9], the complexity of the expanding compiler relates to the (minimum) amplification order of the gadgets composing the base compiler CC . If the latter achieve (t, f) -RPE with an

amplification order d , the expanding compiler achieves $(p, 2^{-\kappa})$ -random probing security with an expansion level k such that $f^{(k)}(p) \leq 2^{-\kappa}$, which yields a complexity blowup of

$$|\widehat{C}| = \mathcal{O}(|C| \cdot \kappa^e) \quad \text{with} \quad e = \frac{\log N_{\max}}{\log d} \quad (4)$$

where

$$N_{\max} = \max |\text{eigenvalues}(M_{CC})|, \quad (5)$$

where $\text{eigenvalues}(\cdot)$ returns the tuple of eigenvalues (or modules of eigenvalues in case of complex numbers) of the input matrix.

Let us slightly explicit the complexity with the 3-gate base $\mathbb{B} = \{\text{add}, \text{mult}, \text{copy}\}$ as used in [9, 10]. Considering that multiplication gates are solely used in the multiplication gadget ($N_{G_{\text{add}},m} = N_{G_{\text{copy}},m} = 0$) which is the case in the constructions of [9, 10], it can be checked that (up to some permutation) the eigenvalues satisfy

$$(\lambda_1, \lambda_2) = \text{eigenvalues}(M_{ac}), \quad \lambda_3 = N_{G_{\text{mult}},m} \quad \text{and} \quad \lambda_4 = n$$

where M_{ac} is the top left 2×2 block matrix of M_{CC}

$$M_{ac} = \begin{pmatrix} N_{G_{\text{add}},a} & N_{G_{\text{copy}},a} \\ N_{G_{\text{add}},c} & N_{G_{\text{copy}},c} \end{pmatrix}$$

where $N_{x,y}$ denotes the number of gates x in a gadget y , with m for the multiplication, a for the addition, and c for the copy. We finally get

$$|\widehat{C}| = \mathcal{O}(|C| \cdot N_{\max}^k) \quad \text{with} \quad N_{\max} = \max(|\text{eigenvalues}(M_{ac})|, N_{G_{\text{mult}},m}, n). \quad (6)$$

As an illustration, the expanding compiler from [10] satisfies $N_{\max} = 3n^2 - 2n$ and $d = \frac{\min(t+1, n-t)}{2}$ which yields an asymptotic complexity of $\mathcal{O}(\kappa^e)$ with

$$e = \frac{\log(3n^2 - 2n)}{\log(\lfloor (n+1)/4 \rfloor)}$$

which tends to 2 as n grows. In comparison, in this work, we shall achieve a quasi-linear complexity, *i.e.*, $N_{\max} = \mathcal{O}(n \log n)$.

2.5 Tolerated Leakage Rate

Finally, we recall the notion of *tolerated leakage rate* which corresponds to the maximum value p for which we have $f(p) < p$. This happens to be a necessary and sufficient condition for the expansion strategy to apply with (t, f) -RPE gadgets.

In practice, the tolerated leakage rate should be measured on concrete devices and fixed accordingly. Hence the motivation to exhibit gadgets which tolerate a high probability to cover any setting. So far, the asymptotic constructions provide a trade-off between tolerated leakage rate and complexity. However, we only know how to compute the former for small numbers of shares and the bounds for larger values are not tight.

As an illustration, the instantiation proposed in [9] tolerates a leakage probability up to $2^{-7.80}$, while the instantiation of [10] tolerates $2^{-7.50}$, both for 3-share base gadgets.

3 Dynamic Random Probing Expansion

As recalled in Section 2, the principle of the expanding compiler is to apply a base circuit compiler CC which is composed of base gadgets –one per gate type in the circuit– several times, say k , to the input circuit: $\widehat{C} = \text{CC}^{(k)}(C)$. The level of expansion k is chosen in order to achieve a certain desired security level κ such that $f^{(k)}(p) \leq 2^{-\kappa}$.

In this section, we generalize this approach to choose the circuit compiler dynamically at the different steps of the expansion. Let $\{\text{CC}_i\}_i$ be a family of circuit compilers, the *dynamic expanding compiler* for this family with respect to the expansion sequence $k_1, \dots, k_\mu$, is defined as

$$\widehat{C} = \text{CC}_\mu^{k_\mu} \circ \text{CC}_{\mu-1}^{k_{\mu-1}} \circ \dots \circ \text{CC}_1^{k_1}(C) . \quad (7)$$

The idea behind this generalization is to make the most from a family of RPE compilers $\{\text{CC}_i\}_i$ which is defined with respect to the number of shares n_i in the base gadgets. If we assume that each compiler CC_i with n_i shares achieves the maximum amplification order $d_i = \frac{n_i+1}{2}$, then the benefit of using a compiler with higher number of shares is to increase the amplification order and thus reduce the number of steps necessary to achieve the desired security level κ . On the other hand, the tolerated leakage rate of existing constructions decreases with n_i . As we show hereafter, a dynamic increase of n_i can ensure both, the tolerated leakage rate of a small n_i and the better complexity of a high n_i .

3.1 Dynamic Expanding Compiler

We formally introduce the dynamic expanding compiler hereafter.

Definition 7 (RPE Compiler). Let $\mathbb{B} = \{g : \mathbb{K}^\ell \rightarrow \mathbb{K}^m\}$ be an arithmetic circuit basis. Let $n, t \in \mathbb{N}$, and let $\{G_g\}_{g \in \mathbb{B}}$ be a family of (t, f_{G_g}) -RPE n -share gadgets for the gate functionalities in $\mathbb{B}$. The RPE compiler CC associated to $\{G_g\}_{g \in \mathbb{B}}$ is the circuit compiler which consists in replacing each gate from a circuit over $\mathbb{B}$ by the corresponding gadget G_g . Moreover,

- the expanding function of CC is the function f defined as

$$f : p \mapsto \max_g f_{G_g}(p)$$

- the amplification order of CC is the integer d defined as

$$d = \min_g d_{G_g}$$

where d_{G_g} is the amplification order of f_{G_g} ,

- the gadget complexity of CC is the integer s defined as

$$s = \max_g |G_g|$$

where $|G_g|$ denotes the number of wires in the gadget G_g ,

- the tolerated leakage rate of CC is the real number $q \in [0, 1)$ such that $f(p) < p$ for every $p < q$.

In the following, we state the security and asymptotic complexity of the dynamic expanding compiler. We will consider a family of different RPE compilers where each compiler is indexed by an index i , *i.e.* a family of different RPE compilers is denoted as $\{\text{CC}_i\}_i$ for different number of shares $\{n_i\}_i$. We start with a formal definition of the dynamic compiler:

Definition 8 (Dynamic Expanding Compiler). *Let $\{\text{CC}_i\}_i$ be a family of RPE compilers with numbers of shares $\{n_i\}_i$. The dynamic expanding compiler for $\{\text{CC}_i\}_i$ with expansion levels $k_1, \dots, k_\mu$, is the circuit compiler $(\text{CC}, \text{Enc}, \text{Dec})$ where*

1. *The input encoding Enc is a $(\prod_{i=1}^\mu n_i^{k_i})$ -linear encoding.*
2. *The output decoding Dec is the $(\prod_{i=1}^\mu n_i^{k_i})$ -linear decoding mapping.*
3. *The circuit compilation is defined as*

$$\text{CC}(\cdot) = \text{CC}_\mu^{k_\mu} \circ \text{CC}_{\mu-1}^{k_{\mu-1}} \circ \dots \circ \text{CC}_1^{k_1}(\cdot) .$$

The following theorem states the random probing security of the dynamic expanding compiler. The proof of the theorem is very similar to the proof of RPE security (Theorem 2) from [9]. The main difference is that at each level of the expansion, we can use a different expanding compiler with different sharing orders. Besides that, the proof follows the same baselines as in [9]. The proof is provided in Appendix A.1.

Theorem 1 (Security). *Let $\{\text{CC}_i\}_i$ be a family of RPE compilers with expanding functions $\{f_i\}_i$. The dynamic expanding compiler for $\{\text{CC}_i\}_i$ with expansion levels $k_1, \dots, k_\mu$ is (p, ε) -random probing secure with*

$$\varepsilon = f_\mu^{k_\mu} \circ \dots \circ f_1^{k_1}(p) .$$

We now state the asymptotic complexity of the dynamic expanding compiler in the next theorem. The proof is given in Appendix A.2.

Theorem 2 (Asymptotic Complexity). *Let $\{\text{CC}_i\}_i$ be a family of circuit compilers with complexity matrices $\{M_{\text{CC}_i}\}_i$. For any input circuit C , the output circuit $\widehat{C} = \text{CC}_\mu^{k_\mu} \circ \dots \circ \text{CC}_1^{k_1}(C)$ is of size*

$$|\widehat{C}| = |C| \cdot \mathcal{O}\left(\prod_{i=1}^\mu |\lambda_i|^{k_i}\right) \quad \text{with } \lambda_i \text{ such that } |\lambda_i| := \max |\text{eigenvalues}(M_{\text{CC}_i})| . \quad (8)$$

In the following, we shall call λ_i as defined above, the *eigen-complexity* of the compiler CC_i . We shall further call the product $\prod_{i=1}^\mu |\lambda_i|^{k_i}$ the *complexity blowup* of the dynamic expanding compiler. We note that minimizing the complexity blowup is equivalent to minimizing the log complexity blowup, which is

$$\sum_{i=1}^\mu k_i \cdot \log_2(|\lambda_i|) . \quad (9)$$

3.2 General Bounds for Asymptotic Constructions

The following theorem introduces general bounds on the tolerated leakage rate and the expanding function of an RPE compiler with respect to its amplification order and gadget complexity. The proof of the theorem is given in the supplementary material (Appendix A.3).

Theorem 3. Let CC_i be an RPE circuit compiler of amplification order d_i and gadget complexity s_i . The tolerated leakage rate q_i of CC_i is lower bounded by

$$q_i \geq \bar{q}_i := \frac{1}{e} \left(\frac{1}{2e} \right)^{\frac{1}{d_i-1}} \left(\frac{d_i}{s_i} \right)^{1+\frac{1}{d_i-1}} \quad (10)$$

For any $p < \bar{q}_i$, the expanding function f_i of CC_i is upper bounded by

$$f_i(p) \leq 2 \binom{s_i}{d_i} p^{d_i} \leq 2 \left(\frac{e \cdot s_i}{d_i} \right)^{d_i} p^{d_i} . \quad (11)$$

The lower bound $\bar{q}_i$ on the tolerated leakage rate quickly converges to the ratio $e^{-1} \cdot d_i/s_i$ as d_i grows. In other words, an RPE compiler family $\{\text{CC}_i\}_i$ indexed by the number of shares n_i of its base gadgets tolerates a leakage probability which is linear in the ratio between its amplification order d_i and its complexity s_i . For known families of RPE compilers from [10] this ratio is in $\mathcal{O}(1/n_i)$.

From Theorem 3, we obtain the following bound for the composition $f_i^{(k)}$. The proof of the corollary is given in the supplementary material (Appendix A.4).

Corollary 1. Let CC_i be an RPE compiler of expanding function f_i , amplification order d_i and gadget complexity s_i . For any $p < \bar{q}_i$ as defined in (10), we have

$$f_i^{(k)}(p) \leq \left[2 \binom{s_i}{d_i} \right]^{(1+\frac{1}{d_i-1})d_i^{k-1}} p^{d_i^k} \leq \left[\left(\frac{2^{\frac{1}{d_i}} e s_i}{d_i} \right)^{(1+\frac{1}{d_i-1})} p \right]^{d_i^k} .$$

The following lemma gives an explicit lower bound on the expansion level $\{k_i\}_i$ to reach some arbitrary target probability $p_{out} = 2^{-\kappa_{out}}$ from a given input probability $p_{in} = 2^{-\kappa_{in}}$ by applying $\text{CC}_i^{(k_i)}$.

Lemma 1. Let $p_{in} = 2^{-\kappa_{in}} < q_i$ and $p_{out} = 2^{-\kappa_{out}} \in (0, 1]$. For any integer k_i satisfying

$$k_i \geq \log_{d_i}(\kappa_{out}) - \log_{d_i}(\kappa_{in} - \Delta_i)$$

with

$$\Delta_i := \left(1 + \frac{1}{d_i - 1} \right) \left(\frac{1}{d_i} + \log_2 \left(\frac{e s_i}{d_i} \right) \right)$$

we have

$$f_i^{(k_i)}(p_{in}) \leq p_{out} = 2^{-\kappa_{out}} .$$

In the above lemma, Δ_i represents a lower bound for κ_{in} which matches the upper bound $\bar{q}_i$ of $p_{in} = 2^{-\kappa_{in}}$. Assuming that s_i and d_i are both monotonically increasing with i , we get that the threshold Δ_i tends towards $\log_2 \left(\frac{e s_i}{d_i} \right)$.

From Lemma 1, we further get that the cost induced by the choice of the compiler CC_i to go from an input probability p_{in} to a target output probability p_{out} is

$$k_i \cdot \log_2(|\lambda_i|) \geq \frac{\log_2(|\lambda_i|)}{\log_2(d_i)} (\log_2(\kappa_{out}) - \log_2(\kappa_{in} - \Delta_i)) \quad (12)$$

(in terms of the log complexity blowup (9)). Note that this lower bound is tight: it could be replaced by an equality at the cost of ceiling the term between parentheses (*i.e.* the term corresponding to k_i). We further note that the above equation is consistent with the complexity analysis of the expanding compiler provided in [9]. Indeed going from a constant leakage probability $p_{in} = p$ to a target security level $p_{out} = 2^{-\kappa}$ by applying k_i times a single RPE compiler CC_i , we retrieve a complexity of $\mathcal{O}(\kappa^e)$ with $e = \frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$.

Equation (12) shows that using CC_i to go from input probability p_{in} to output probability p_{out} induces a log complexity cost close to

$$\frac{\log_2(|\lambda_i|)}{\log_2(d_i)} (\log_2(\kappa_{out}) - \log_2(\kappa_{in}))$$

provided that κ_{in} is sufficiently greater than Δ_i . So given the latter informal condition, it appears that the parameter i minimizing the ratio $\frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$ gives the best complexity.

Application. For the asymptotic construction introduced in [10], the RPE compiler CC_i features

- an amplification order $d_i = \mathcal{O}(n_i)$,
- a gadget complexity $s_i = \mathcal{O}(n_i^2)$,
- an eigen-complexity $|\lambda_i| = \mathcal{O}(n_i^2)$.

For such a construction, the ratio $\frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$ is decreasing and converging towards 2 as n_i grows. On the other hand, Δ_i tends to $\log_2(n_i)$ which implies that CC_i should only be applied to an input probability lower than $\frac{1}{n_i}$.

3.3 Selection of the Expansion Levels

In this section, we investigate the impact of the choice of the expansion levels k_i on the complexity of the dynamic expanding compiler. We first assess the asymptotic complexity obtained from a simple approach and then provide some application results for some given gadgets.

In the following CC_0 shall denote an RPE compiler with constant parameters while $\{\text{CC}_i\}_{i \geq 1}$ shall denote a family of RPE compilers indexed by a parameter i . We do this distinction since the goal of the CC_0 compiler shall be to tolerate the highest leakage rate and to transit from a (possibly high) leakage probability p to some lower failure probability p_i which is in turn tolerated by at least one compiler from $\{\text{CC}_i\}_i$.

A Simple Approach. We consider a simple approach in which the compiler CC_0 is iterated k_0 times and then a single compiler CC_i is iterated k_i times. The complexity blowup of this compiler is $|\lambda_0|^{k_0} |\lambda_i|^{k_i}$. The first expansion level k_0 is chosen to ensure that the intermediate probability $p_i := f_0^{(k_0)}(p)$ is lower than $\bar{q}_i$ (the lower bound on the tolerated leakage rate of CC_i from Theorem 3). Then k_i is chosen so that $f_i^{(k_i)} \leq 2^{-\kappa}$.

Concretely, we set $\kappa_i := \Delta_i + 1$ which, by Lemma 1, gives

$$k_0 = \lceil \log_{d_0}(\Delta_i + 1) - \log_{d_0}(\log_2(p) - \Delta_0) \rceil, \quad (13)$$

and

$$k_i = \lceil \log_{d_i}(\kappa) \rceil = \mathcal{O}(\log_{d_i}(\kappa)) . \quad (14)$$

For some constant leakage probability p and some start compiler CC_0 with constant parameters, we get $k_0 = \mathcal{O}(\log_{d_0}(\Delta_i))$ giving an asymptotic complexity blowup of

$$\mathcal{O}(|\lambda_0|^{k_0} |\lambda_i|^{k_i}) = \mathcal{O}(\Delta_i^{e_0} \kappa^{e_i}) \quad \text{with} \quad e_0 = \frac{\log_2(|\lambda_0|)}{\log_2(d_0)} \quad \text{and} \quad e_i = \frac{\log_2(|\lambda_i|)}{\log_2(d_i)} . \quad (15)$$

Then for any choice of i we get an asymptotic complexity blowup of $\mathcal{O}(\kappa^{e_i})$ which is the same asymptotic complexity as the standard expanding compiler with base compiler CC_i . On the other hand, our simple dynamic compiler $\text{CC}_i^{(k_i)} \circ \text{CC}_0^{(k_0)}$ tolerates the same leakage rate as CC_0 .

Using this simple approach we hence get the best of both worlds:

- a possibly inefficient RPE compiler CC_0 tolerating a high leakage rate q_0 ,
- a family of RPE compilers $\{\text{CC}_i\}_i$ with complexity exponent $e_i = \frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$ decreasing with i .

We stress that for monotonously increasing $|\lambda_i|$ and d_i , the asymptotic complexity of our simple approach is $\mathcal{O}(\kappa^e)$ where e can be made arbitrary close to $\lim_{i \rightarrow \infty} \frac{\log_2(|\lambda_i|)}{\log_2(d_i)}$.

Application. To illustrate the benefits of our dynamic approach, we simply get back to the experimentations on the AES implementation from [9]. The authors apply either a 3-share or 5-share compiler repeatedly until they reach their targeted security level. While using the 5-share compiler reduces the tolerated probability, we demonstrate that we can use both compilers to get the best tolerated probability as well as a better complexity.

Figure 1 illustrates the trade-offs in terms of achieved security level and complexity of the expansion strategy when using different compilers at each iteration of the expansion. Starting from a tolerated leakage probability p ($2^{-7.6}$ on the left and $2^{-9.5}$ on the right), the empty bullets ($\circ$) give this trade-off when only the 3-share compiler is iterated. In this case, the final security function ε from Theorem 1 is equal to $f_3^{(k_3)}(p)$ if we consider f_3 to be the failure function of the 3-share compiler, for a certain number of iterations k_3 which is written next to each empty bullet on the figure. On the other hand, the black bullets ($\bullet$) represent the trade-offs achieved in terms of complexity and security levels while combining both compilers with different numbers of iterations. In this case, we start the expansion with a certain number of iterations k_3 of the 3-share compiler, and then we continue with k_5 iterations of the 5-share compiler of failure function f_5 , the final compiled circuit is then random probing secure with $\varepsilon = f_5^{(k_5)}(f_3^{(k_3)}(p))$ for $p \in \{2^{-7.6}, 2^{-9.5}\}$. The number of iterations of the compilers is written next to each black bullet in the format k_3 - k_5 .

For instance, starting from the best tolerated probability $2^{-7.6}$, the static compiler from [9, 10] requires 11 applications of the 3-share compiler to achieve a security level of at least 80 bits. This effort comes with an overall complexity of $10^{17.52}$. Using our dynamic approach, we can combine the 3-share and the 5-share to achieve this 80 bits security level for the same tolerated probability but with a complexity of $10^{16.04}$. That would require 7 iterations of the 3-share compiler and 2 iterations of the 5-share compiler. Starting from the same leakage probability, a security level of at least 128 bits is achieved also with 11 applications of the 3-share compiler with a complexity of $10^{17.52}$. In order to achieve at least the same security, we would need more iterations of both compilers in the dynamic approach. With 7 iterations of the 3-share compiler and 3 iterations of

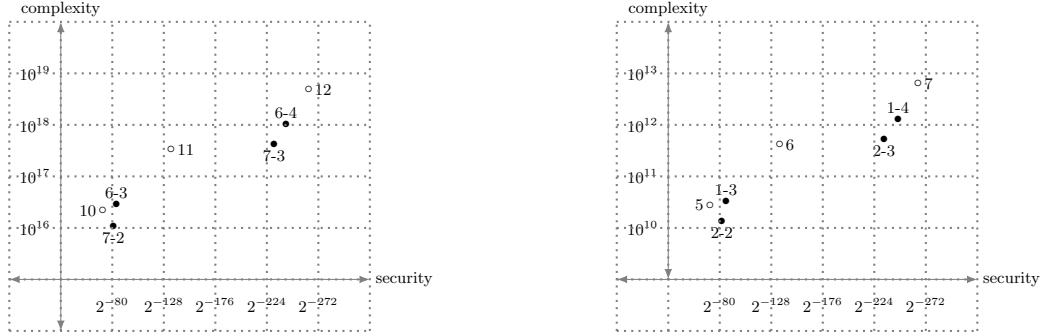


Fig. 1: Complexity of random probing AES for different security levels for a tolerated probability of $2^{-7.6}$ (left) or $2^{-9.5}$ (right).

the 5-share compiler, we get a complexity of $10^{17.62}$ which is very close to the complexity of the 3-share application alone, while achieving a security level of 231 bits. That is, we almost double the security level achieved using 11 iterations of the 3-share compiler with an almost equal complexity. For a tolerated probability of $2^{-7.6}$ and at least 128 bits of security, note that 11 applications of the 3-share compiler yield a security order of 2^{-135} while both other trade-offs directly yield security orders of 2^{-242} (6 iterations of 3-share and 4 iterations of 5-share) and 2^{-231} (7 iterations of 3-share and 3 iterations of 5-share), with one less iteration they would be below 128 bits, which explains their more important complexity. The same behavior can be observed with a starting tolerated leakage probability of $2^{-9.5}$ on the right.

The above results motivate the next contributions of this paper, namely finding RPE compilers which achieve the maximal amplification orders and which benefit from good asymptotic complexity (*i.e.* gadgets defined for any number of shares n with amplification order increasing with n) in order to optimize the security-efficiency trade-off and to tolerate the best possible leakage probability. We showed this far that the tolerated leakage probability decreases with an increasing number of shares n . So if we want to tolerate the best leakage probability, we would start with a few iterations of a compiler with a small number of shares and which tolerates a good leakage probability (which can be computed for instance with the verification tool VRAPS [9]), typically a 3-share construction. Meanwhile, after a few constant number of iterations, we can change to a different compiler which benefits from a better asymptotic complexity (as explained above with our simple approach). In the constructions from [10], the bottleneck in terms of asymptotic complexity was from the linear gadgets (addition and copy). Thanks to the quasilinear refresh gadget we introduce later in this paper, the bottleneck becomes the multiplication gadget (with n^2 multiplications), which we also improve in the following sections under some conditions on the base field.

4 Linear Gadgets with Quasi-Linear Complexity

In a first attempt, we aim to reduce the complexity of the linear gadgets that are to be used in our dynamic compiler.

In [10], the authors provide new constructions of generic addition and copy gadgets, using a refresh gadget G_{refresh} as a building block. The construction works for any number of shares and

the authors prove the RPE security of the gadgets based on the security of G_{refresh} . In a nutshell, given a n -share refresh gadget G_{refresh} , the authors construct a copy gadget G_{copy} which on input sharing $(a_1, \dots, a_n)$, outputs the sharings

$$\left(G_{\text{refresh}}(a_1, \dots, a_n), G_{\text{refresh}}(a_1, \dots, a_n) \right) \quad (16)$$

with two independent executions of G_{refresh} . The authors also construct an addition gadget G_{add} which, on input sharings $(a_1, \dots, a_n)$ and $(b_1, \dots, b_n)$, first refreshes the inputs separately, then outputs the sharewise sum of the results

$$\left(G_{\text{refresh}}(a_1, \dots, a_n) + G_{\text{refresh}}(b_1, \dots, b_n) \right). \quad (17)$$

If the refresh gadget G_{refresh} is TRPE of amplification order d , the authors show that G_{copy} is also TRPE of amplification order d , and G_{add} is TRPE of amplification order at least $\lfloor d/2 \rfloor$.

While the copy gadgets from [10] achieve an optimal amplification order, this is not the case yet for addition gadgets and we first aim to fill this gap. Precisely, we introduce a new property which, when satisfied by its inherent refresh gadget G_{refresh} , makes the addition gadget TRPE with the same amplification order as G_{refresh} . We then prove that this new property is actually satisfied by the refresh gadget from [6] which has quasi-linear complexity $\mathcal{O}(n \log n)$ in the sharing order n . Using this refresh gadget as a building block, we obtain linear gadgets G_{add} and G_{copy} with quasi-linear complexities.

Constructions of Linear Gadgets from a Stronger Building Block. We first define our new property (as a variant of properties defined in [9, 10]) which proves to be a useful requirement for refresh gadgets when used as a building block of linear gadgets.

Definition 9 (t -Strong TRPE2). *Let G be an n -share 1-input gadget. Then G is t -Strong TRPE2 (abbreviated t -STRPE2) if and only if for any set J' of output shares indices and any set W of internal wires of G such that $|W| + |J'| \leq t$, there exists a set J of output share indices such that $J' \subseteq J$ and $|J| = n - 1$ and such that the assignment of the wires indexed by W together with the output shares indexed by J can be perfectly simulated from the input shares indexed by a set I of cardinality satisfying $|I| \leq |W| + |J'|$.*

Remark 1. This new property directly implies the TRPE2 property with maximal amplification order introduced in [10]. Recall that G is t -TRPE2 with maximal amplification order if and only if for any set W of probed wires such that $|W| < \min(t + 1, n - t)$, there exists a set J of output shares indices such that $|J| = n - 1$ and such that an assignment of the wires indexed by W and the output shares indexed by J can be jointly perfectly simulated from input shares indexed in a set I such that $|I| \leq |W|$.

Having a refresh gadget which satisfies the property from Definition 9 results in tighter constructions for generic addition gadgets as stated in Lemma 2. Its proof is given in Appendix A.6.

Lemma 2. *Let G_{refresh} be an n -share refresh gadget and let G_{add} be the addition gadget described in Equation (17). Then if G_{refresh} is (t, f) -TRPE for any $t \leq n - 1$ of amplification order $d \geq \min(t + 1, n - t)$ and G_{refresh} is $(n - 1)$ -STRPE2, then G_{add} is (t, f') -RPE (resp. (t, f') -TRPE) for any $t \leq n - 1$ for some f' of amplification order $\min(t + 1, n - t)$.*

Instantiation of Linear Gadgets with Quasi-Linear Refresh Gadget. A refresh gadget with $\mathcal{O}(n \log n)$ complexity was introduced in [6]. In a nutshell, the idea is to add a linear number of random values on the shares at each step, to split the shares in two sets to apply the recursion, and then to add a linear number of random values again. For the sake of completeness, we provide the algorithmic description of this refresh gadget in Appendix A.7. It was proven to be $(n-1)$ -SNI in [6]. In Lemma 3, we show that this gadget is also (t, f) -TRPE of amplification order $\min(t+1, n-t)$ and that it satisfies $(n-1)$ -STRPE2. The proof is given in Appendix A.8.

Lemma 3. *Let G_{refresh} be the n -share refresh gadget described above from [6]. Then G_{refresh} is (t, f) -TRPE for some function $f : \mathbb{R} \rightarrow \mathbb{R}$ of amplification order $d \geq \min(t+1, n-t)$. G_{refresh} is additionally $(n-1)$ -STRPE2.*

Hence, we can instantiate the generic copy and addition gadgets described in (16) and (17) using the above refresh gadget as G_{refresh} . We thus obtain RPE gadgets G_{add} and G_{copy} enjoying optimal amplification order in quasi-linear complexity $\mathcal{O}(n \log n)$.

Regarding the asymptotic complexity of the expanding compiler, the eigenvalues λ_1, λ_2 from Section 2 are hence now both in $\mathcal{O}(n \log n)$. At this point, only the quadratic number of multiplications in the multiplication gadget still separates us from a compiler of quasi-linear complexity. We tackle this issue in the next section by constructing a generic multiplication gadget. We finally end up with a full expanding compiler with quasi-linear asymptotic complexity.

5 Towards Optimal Multiplication Gadgets

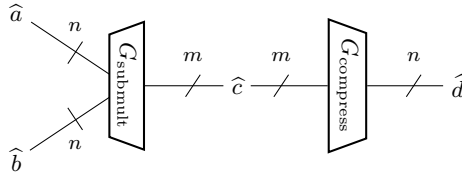


Fig. 2: n -share multiplication gadget G_{mult} from two subgadgets G_{submult} and G_{compress}

In what follows we should distinguish two types of multiplication gates: regular two-operand multiplications on $\mathbb{K}$, that we shall call bilinear multiplications, and multiplications by constant (or scalar multiplications) which have a single input operand and the constant scalar is considered as part of the gate description.

In previous works [9, 10], the number of bilinear multiplications is the prominent term of the expanding compiler's complexity. While the most deployed multiplication gadgets (*e.g.*, [18]) require a quadratic number of bilinear multiplications in the masking order, the authors of [8] exhibited a probing secure higher-order masking multiplication with only a linear number of bilinear multiplications. Their construction, which applies on larger fields, is built from the composition of two subgadgets G_{submult} and G_{compress} , as described in Figure 2. In a nutshell, on input sharings $\hat{a}$ and $\hat{b}$, the subgadget G_{submult} performs multiplications between the input shares of $\hat{a}$ and $\hat{b}$ as well as linear combinations of these products and it outputs a m -sharing $\hat{c}$ of the product $a \cdot b$ where

$m \geq n$ ⁵. Next, the compression gadget G_{compress} compresses the m -sharing $\hat{c}$ back into an n -sharing $\hat{d}$ of the product $a \cdot b$.

The authors of [8] instantiate this construction with a sub-multiplication gadget which performs only $\mathcal{O}(n)$ bilinear multiplications and with the compression gadget from [11]. In addition to bilinear multiplications, their sub-multiplication gadget additionally requires a quadratic number of linear operations (*i.e.*, addition, copy, multiplications by a constant) and random generation gates.

In the following, we rely on the construction [8] with its gadget G_{submult} which offers a linear number of bilinear multiplications to build a more efficient RPE multiplication gadget. In order to use it in our expanding compiler, we integrate an additional gate for the multiplication by a constant and discuss the resulting asymptotic complexity. We additionally demonstrate that the compression gadget of [8] is not $(n-1)$ -SNI as claimed in the paper, and show that we can rely on other simple and more efficient compression gadgets which satisfy the expected properties.

5.1 Global Multiplication Gadget

We first define two new properties that G_{submult} and G_{compress} will be expected to satisfy to form a (t, f) -RPE multiplication gadget with the maximum amplification order from the construction [8].

Contrary to the usual simulation notions, the first *partial*-NI property distinguishes the number of probes on the gadget, and the number of input shares that must be used to simulate them. It additionally tolerates a *simulation failure* on at most one of the inputs (*i.e.*, no limitation on the number of shares for the simulation).

Definition 10 ((s, t)-partial NI). Let G be a gadget with two input sharings $\hat{a}$ and $\hat{b}$. Then G is (s, t) -partial NI if and only if the assignment of any t wires of G can be perfectly simulated from shares $(a_i)_{i \in I_1}$ of $\hat{a}$ and $(b_i)_{i \in I_2}$ of $\hat{b}$ such that $|I_1| \leq s$ **or** $|I_2| \leq s$.

The second property is a variant of the classical TRPE property that we refer to as *comp-TRPE*.

Definition 11 ((t, f)-comp-TRPE). Let G be a 1-to-1 gadget with m input shares and n output shares such that $m > n$. Let $t \leq n-1$ and $d = \min(t+1, n-t)$. Then G is (t, f) -comp-TRPE if and only if for all sets of internal wires W of G with $|W| \leq 2d-1$, we have:

1. $\forall J, |J| \leq t$ a set of output share indices of G , the assignment of the wires indexed by W and the output shares indexed by J can be jointly perfectly simulated from the input shares of G indexed by a set I , such that $|I| \leq |W|$.
2. $\exists J', |J'| = n-1$ a set of output share indices of G , such that the assignment of the wires indexed by W and the output shares indexed by J' can be jointly perfectly simulated from the input shares of G indexed by a set I , such that $|I| \leq |W|$.

Similarly to what was done in [8] for the SNI property, we can prove that the composition of a gadget G_{submult} and G_{compress} which satisfy well chosen properties results in an overall multiplication gadget which is (t, f) -RPE specifically for any $t \leq n-1$ achieving the maximum amplification order $d = \min(t+1, n-t)$. This is formally stated in Lemma 4 which proof is given in Appendix A.9.

Lemma 4. Consider the n -share multiplication gadget of Figure 2 formed by a 2-to-1 multiplication subgadget G_{submult} of m output shares and a 1-to-1 compression gadget G_{compress} of m input shares such that $m > n$. Let $t \leq n-1$ and $d = \min(t+1, n-t)$. If

⁵ In case of a sharewise multiplication for instance, we would have $m = n^2$.

- G_{submult} is $(d-1)$ -NI and $(d-1, 2d-1)$ -partial NI,
- G_{compress} is (t, f) -comp-TRPE,

then the multiplication gadget G_{mult} is (t, f) -RPE of amplification order d .

5.2 Construction of G_{compress}

In a first attempt, we analyze the compression function that was introduced in [11] and used to build a multiplication gadget in [8]. As it turns out not to be SNI or meet our requirements for the expanding compiler, we exhibit a new and also more efficient construction in a second attempt.

G_{compress} from [8, 11]. The authors of [8] use the $[m : n]$ -compression gadget introduced in [11] for any input sharing m , using a $[2n : n]$ -compression subgadget as a building block. In a nutshell, it first generates an *ISW*-refresh of the zero n -sharing $(w_1, \dots, w_n)$. Then, these shares are added to the input ones $(c_1, \dots, c_n)$ to produce the sequence of output shares $(c_1 + w_1, \dots, c_n + w_n)$.

The compression gadget is claimed to be $(n-1)$ -SNI in [8]. However, we demonstrate that it is not with the following counterexample. Let $n > 2$ and $i \in [n]$. We consider the set composed of a single output share of the compression procedure $J = \{(c_i + w_i) + c_{n+i}\}$ and the set of probes on the internal wires $W = \{w_i\}$. For the compression to be 2-SNI, we must be able to perfectly simulate both the wires in W and J with at most $|W| = 1$ share of the input $\hat{c}$. However, we can easily observe that $(c_i + w_i) + c_{n+i} - w_i = c_i + c_{i+n}$ requires the two input shares c_i and c_{i+n} to be simulated, which does not satisfy the 2-SNI property. In conclusion, the above gadget is actually not SNI, and interestingly it is not sufficient either for our construction, *i.e.* it does not satisfy Definition 11. This observation motivates our need for a new compression gadget which satisfies the necessary property for our construction.

New Construction for G_{compress} . In Algorithm 1, we exhibit a new $[m : n]$ -compression technique using an m -share refresh gadget G_{refresh} as a building block. We demonstrate in Lemma 5 that this new compression gadget satisfies the necessary properties for our construction as long as $m \geq 2n$. The proof is given in Appendix A.10.

Algorithm 1: $[m : n]$ -compression gadget

Input : $(c_1, \dots, c_m)$ such that $m \geq 2n$, m -share refresh gadget G_{refresh}
Output: $(d_1, \dots, d_n)$ such that $\sum_{i=1}^n d_i = \sum_{i=1}^m c_i$
 $K \leftarrow \lfloor m/n \rfloor$;
 $(c'_1, \dots, c'_m) \leftarrow G_{\text{refresh}}(c_1, \dots, c_m)$;
 $(d_1, \dots, d_n) \leftarrow (c'_1, \dots, c'_n)$;
for $i = 1$ **to** $K - 1$ **do**
 $(d_1, \dots, d_n) \leftarrow (d_1 + c'_{1+i \cdot n}, \dots, d_n + c'_{n+i \cdot n})$;
end
for $i = 1$ **to** $m - K \cdot n$ **do**
 $d_i \leftarrow d_i + c'_{i+K \cdot n}$;
end
return $(d_1, \dots, d_n)$;

Lemma 5. *Let G_{compress} be the $[m : n]$ -compression gadget from Algorithm 1 such that $m \geq 2n$. If G_{refresh} is $(m-1)$ -SNI and $(m-1)$ -STRPE2, then G_{compress} is (t, f) -comp-TRPE (Definition 11).*

As shown in Section 4, the refresh gadget from [5] is actually $(m-1)$ -SNI and $(m-1)$ -STRPE2 for any sharing order m . This gadget can then be used as a building block for the $[m : n]$ -compression gadget, giving it a complexity of $\mathcal{O}(m \log m)$ and satisfying the necessary properties. In addition, this further provides an improvement over the complexity of the proposed gadget in [8] which has a complexity of $\mathcal{O}(\lfloor \frac{m}{n} \rfloor n^2)$ (because it performs a n -share ISW-refreshing $\lfloor \frac{m}{n} \rfloor$ times, see [8] for more details on the algorithm).

5.3 Construction of G_{submult}

To complete the construction of the overall multiplication gadget, we now exhibit relevant constructions for G_{submult} . We first rely on the construction from [8] which happens to achieve the desired goal in some settings. While all the cases are not covered by the state-of-the-art proposal, we then slightly modify the construction to meet all our requirements. Both constructions rely on linear multiplications that are not included yet on the expanding compiler. We thus start with a construction for this additional linear gadget that we further denote G_{cmult} .

Construction for G_{cmult} . We give a natural construction for G_{cmult} in Algorithm 2 which simply multiplies each input share by the underlying constant value and then applies a (t, f) -RPE refresh gadget G_{refresh} . Basically, with a (T)RPE refresh gadget G_{refresh} , we obtain a (T)RPE linear multiplication gadget G_{cmult} as stated in Lemma 6. The proof is given in Appendix A.5.

Algorithm 2: n -share multiplication by a constant

Input : sharing $(a_1, \dots, a_n)$, constant value $\hat{c}$, n -share refresh gadget G_{refresh}
Output: sharing $(d_1, \dots, d_n)$ such that $d_1 + \dots + d_n = \hat{c} \cdot (a_1 + \dots + a_n)$
 $(b_1, \dots, b_n) \leftarrow (c \cdot a_1, \dots, c \cdot a_n);$
 $(d_1, \dots, d_n) \leftarrow G_{\text{refresh}}((b_1, \dots, b_n));$
return $(d_1, \dots, d_n);$

Lemma 6. *Let G_{refresh} be a (t, f) -(T)RPE n -share refresh gadget of amplification order d . Then G_{cmult} instantiated with G_{refresh} is (t, f') -(T)RPE of amplification order d .*

Relying on an additional gate for the linear multiplication does not impact the security analysis and the application of the compilation, but it modifies the complexity analysis of the expanding compiler. From the analysis given in Section 2.4, a complexity vector is associated to each base gadget $N_G = (N_a, N_c, N_{cm}, N_m, N_r)^\top$ where $N_a, N_c, N_{cm}, N_m, N_r$ stand for the number of addition gates, copy gates, constant multiplication gates, (bilinear) multiplication gates and random gates respectively in the corresponding gadget. The matrix M_{CC} is now a 5×5 square matrix defined as

$$M = (N_{G_{\text{add}}} \mid N_{G_{\text{copy}}} \mid N_{G_{\text{cmult}}} \mid N_{G_{\text{mult}}} \mid N_{G_{\text{random}}})$$

including, for each vector, the number of linear multiplications. Five eigenvalues $\lambda_1, \lambda_2, \lambda_3, \lambda_4, \lambda_5$ are to be computed, *i.e.*, one more compared to the expanding compiler in the original setting.

We can consider as before that bilinear multiplication gates are solely used in G_{mult} ($N_{G_{\text{add}},m} = N_{G_{\text{copy}},m} = N_{G_{\text{cmult}},m} = 0$) and that constant multiplication gates are eventually solely used in G_{cmult} and G_{mult} ($N_{G_{\text{add}},cm} = N_{G_{\text{copy}},cm} = 0$) which is the case in the constructions we consider in this paper. It can be checked that (up to some permutation) the eigenvalues satisfy

$$(\lambda_1, \lambda_2) = \text{eigenvalues}(M_{ac}) , \quad \lambda_3 = N_{G_{\text{cmult}},cm} , \quad \lambda_4 = N_{G_{\text{mult}},m} \quad \text{and} \quad \lambda_5 = n$$

where M_{ac} is the top left 2×2 block matrix of M_{CC}

$$M_{ac} = \begin{pmatrix} N_{G_{\text{add}},a} & N_{G_{\text{copy}},a} \\ N_{G_{\text{add}},c} & N_{G_{\text{copy}},c} \end{pmatrix} .$$

We get two complexity expressions for the expansion strategy

$$|\widehat{C}| = \mathcal{O}(|C| \cdot N_{\text{max}}^k) \tag{18}$$

with $N_{\text{max}} = \max(|\text{eigenvalues}(M_{ac})|, N_{G_{\text{cmult}},cm}, N_{G_{\text{mult}},m}, n)$ and with the security parameter κ

$$|\widehat{C}| = \mathcal{O}(|C| \cdot \kappa^e) \quad \text{with} \quad e = \frac{\log N_{\text{max}}}{\log d} .$$

Note that the exhibited construction for the linear multiplication gadget requires $N_{G_{\text{cmult}},cm} = n$ linear multiplications. Hence $\lambda_3 = N_{G_{\text{cmult}},cm} = \lambda_5 = N_{G_{\text{random}},r} = n$ and the global complexity (18) can be rewritten as

$$|\widehat{C}| = \mathcal{O}(|C| \cdot N_{\text{max}}^k) \quad \text{with} \quad N_{\text{max}} = \max(|\text{eigenvalues}(M_{ac})|, N_{G_{\text{mult}},m})$$

if the number of multiplications is greater than n . The asymptotic complexity of the RPE compiler is thus not affected by our new base gadget G_{cmult} . We now describe our constructions of G_{submult} .

G_{submult} from [8]. The authors of [8] provide a $(n-1)$ -NI construction for G_{submult} which outputs $2n-1$ shares while consuming only a linear number of bilinear multiplications in the masking order. We first recall their construction which relies on two square matrices of $(n-1)^2$ coefficients in the working field. As shown in [8], these matrices are expected to satisfy some condition for the compression gadget to be $(n-1)$ -NI. Since we additionally want the compression gadget to be $(d-1, 2d-1)$ -partial NI, we introduce a stronger condition and demonstrate the security of the gadget in our setting.

Let $\mathbb{F}_q$ be the finite field with q elements. Let $\gamma = (\gamma_{i,j})_{1 \leq i,j < n} \in \mathbb{F}_q^{(n-1) \times (n-1)}$ be a constant matrix, and let $\delta = (\delta_{i,j})_{1 \leq i,j < n} \in \mathbb{F}_q^{(n-1) \times (n-1)}$ be the matrix defined by $\delta_{i,j} = 1 - \gamma_{j,i}$ for all $1 \leq i, j < n-1$. G_{submult} takes as input two n -sharings $\mathbf{a}$ and $\mathbf{b}$ and outputs a $(2n-1)$ -sharing $\mathbf{c}$ such that:

- $c_1 = \left(a_1 + \sum_{i=2}^n (r_i + a_i) \right) \cdot \left(b_1 + \sum_{i=2}^n (s_i + b_i) \right)$
- $c_i = -r_i \cdot \left(b_1 + \sum_{j=2}^n (\delta_{i-1,j-1} s_j + b_j) \right)$ for $i = 2, \dots, n$

- $c_{i+n-1} = -s_i \cdot \left(a_1 + \sum_{j=2}^n (\gamma_{i-1,j-1} r_j + a_j) \right)$ for $i = 2, \dots, n$

where r_i and s_i are randomly generated values for all $2 \leq i \leq n$. It can be easily checked that G_{submult} performs $2n - 1$ bilinear multiplications, and that it is correct, i.e. $\sum_{i=1}^{2n-1} c_i = \sum_{i=1}^n a_i \cdot \sum_{i=1}^n b_i$.

In [8], the authors prove that a gadget is $(n-1)$ -NI if one cannot compute a linear combination of any set of $n-1$ probes which can reveal all of the n secret shares of the inputs and which does not include any random value in its algebraic expression. We refer to [8] for more details on this result.

Based on this result, the authors demonstrate in [8], that G_{submult} is $(n-1)$ -NI if the matrices γ and δ satisfy Condition 1 that we recall below.

Condition 1 (from [8]) Let $\ell = 2 \cdot (n+1) \cdot (n-1) + 1$. Let $\mathbf{I}_{n-1} \in \mathbb{F}_q^{(n-1) \times (n-1)}$ be the identity matrix, $\mathbf{0}_{x \times y} \in \mathbb{F}_q^{x \times y}$ be a matrix of zeros (when $y = 1$, $\mathbf{0}_{x \times y}$ is also written $\mathbf{0}_x$), $\mathbf{1}_{x \times y} \in \mathbb{F}_q^{x \times y}$ be a matrix of ones, $\mathbf{D}_{\gamma,j} \in \mathbb{F}_q^{(n-1) \times (n-1)}$ be the diagonal matrix such that $D_{\gamma,j,i,i} = \gamma_{j,i}$, $\mathbf{T}_{n-1} \in \mathbb{F}_q^{(n-1) \times (n-1)}$ be the upper-triangular matrix with just ones, and $\mathbf{T}_{\gamma,j} \in \mathbb{F}_q^{(n-1) \times (n-1)}$ be the upper-triangular matrix for which $T_{\gamma,j,i,k} = \gamma_{j,i}$ for $i \leq k$:

$$\mathbf{I}_{n-1} = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix} \quad \mathbf{D}_{\gamma,j} = \begin{pmatrix} \gamma_{j,1} & 0 & \dots & 0 \\ 0 & \gamma_{j,2} & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & \gamma_{j,n-1} \end{pmatrix}$$

$$\mathbf{T}_{n-1} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ 0 & 1 & & 1 \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & 1 \end{pmatrix} \quad \mathbf{T}_{\gamma,j} = \begin{pmatrix} \gamma_{j,1} & \gamma_{j,1} & \dots & \gamma_{j,1} \\ 0 & \gamma_{j,2} & & \gamma_{j,2} \\ \vdots & & \ddots & \vdots \\ 0 & \dots & 0 & \gamma_{j,n-1} \end{pmatrix}$$

We define the following matrices (with $n' = n-1$):

$$\mathbf{L} = \left(\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c} 1 & \mathbf{0}_{1 \times n'} & \mathbf{0}_{1 \times n'} & \mathbf{0}_{1 \times n'} & \mathbf{0}_{1 \times n'} & \dots & \mathbf{0}_{1 \times n'} & \mathbf{1}_{1 \times n'} & \mathbf{1}_{1 \times n'} & \dots & \mathbf{1}_{1 \times n'} \\ \hline \mathbf{0}_{n'} & \mathbf{I}_{n'} & \mathbf{0}_{n' \times n'} & \mathbf{I}_{n'} & \mathbf{I}_{n'} & \dots & \mathbf{I}_{n'} & \mathbf{T}_{n'} & \mathbf{T}_{n'} & \dots & \mathbf{T}_{n'} \end{array} \right)$$

$$\mathbf{M} = \left(\begin{array}{c|c|c|c|c|c|c|c|c|c|c|c} \mathbf{0}_{n'} & \mathbf{0}_{n' \times n'} & \mathbf{I}_{n'} & \mathbf{I}_{n'} & \mathbf{D}_{\gamma,1} & \dots & \mathbf{D}_{\gamma,n'} & \mathbf{T}_{n'} & \mathbf{T}_{\gamma,1} & \dots & \mathbf{T}_{\gamma,n'} \end{array} \right)$$

Condition 1 is satisfied for a matrix γ if for any vector $\mathbf{v} \in \mathbb{F}_q^\ell$ of Hamming weight $\text{hw}(\mathbf{v}) \leq n-1$ such that $\mathbf{L} \cdot \mathbf{v}$ contains no coefficient equal to 0 then $\mathbf{M} \cdot \mathbf{v} \neq \mathbf{0}_{n-1}$.

In the above condition, the matrices $\mathbf{L}$ and $\mathbf{M}$ represent the vectors of dependencies for each possible probe. All the probes involving shares of $\hat{a}$ for matrix γ (and symmetrically shares of $\hat{b}$ for matrix δ) are covered in the columns of $\mathbf{L}$ and $\mathbf{M}$. Namely, the first column represents the probe a_1 . As it does not involve any random, it results in a zero column in $\mathbf{M}$. The next columns represents the probes a_i , then the probes r_i . They are followed by columns for the probes $(a_i + r_i)$, then $(a_i + \gamma_{j-1,i-1} r_i)$ (for $2 \leq j \leq n$), then $a_1 + \sum_{i=2}^k (r_i + a_i)$ (for $2 \leq k \leq n$), and finally then $a_1 + \sum_{j=2}^k (\gamma_{i-1,j-1} r_j + a_j)$ (for $2 \leq i \leq n$ and $2 \leq k \leq n$). The above condition means that there is no linear combination of $(n-1)$ probes which can include the expression of all of the input shares, and no random variable.

From this result and by the equivalence between non-interference and tight non-interference developed in [8], we conclude that G_{submult} is $(d-1)$ -NI for $d = \min(t+1, n-t)$ for any $t \leq n-1$. Lemma 4 also requires G_{submult} to be $(d-1, 2d-1)$ -partial NI to get an overall RPE multiplication gadget. For G_{submult} to satisfy this second property, we need to rely on a stronger condition for matrices γ and δ that we present in Condition 2.

Condition 2 Let $z = 2 \cdot (n+1) \cdot (n-1) + 1$. Let $\mathbf{I}_{n-1} \in \mathbb{F}_q^{(n-1) \times (n-1)}$, $\mathbf{0}_{\ell \times n} \in \mathbb{F}_q^{\ell \times n}$, $\mathbf{1}_{\ell \times n} \in \mathbb{F}_q^{\ell \times n}$, $\mathbf{D}_{\gamma,j} \in \mathbb{F}_q^{(n-1) \times (n-1)}$, $\mathbf{T}_{n-1} \in \mathbb{F}_q^{(n-1) \times (n-1)}$, $\mathbf{T}_{\gamma,j} \in \mathbb{F}_q^{(n-1) \times (n-1)}$ and $\mathbf{L}$ and $\mathbf{M}$ the same matrices as defined in Condition 1.

Condition 2 is satisfied for a matrix γ if and only if for any vector $\mathbf{v} \in \mathbb{F}_q^z$ of Hamming weight $\text{hw}(\mathbf{v}) \leq n-1$, and for any $i_1, \dots, i_K \in [z]$ such that $v_{i_1} \neq 0, \dots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \dots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient (i.e there are K probes of the form $a_1 + \sum_{i=2}^n (r_i + a_i)$ or $a_1 + \sum_{j=2}^n (\gamma_{i-1,j-1} r_j + a_j)$ for any $i \in \{2, \dots, n\}$), if $\mathbf{M} \cdot \mathbf{v} = 0$, then we have $\text{hw}(\mathbf{L} \cdot \mathbf{v}) \leq \text{hw}(\mathbf{v}) - K$.

Based on this new condition, we can prove our second property G_{submult} , as stated in Lemma 7. The proof is given in Appendix A.11.

Lemma 7. Let $t \leq n-1$ such that either n is even or $t \neq \lfloor \frac{n-1}{2} \rfloor$ and let $d = \min(t+1, n-t)$. Let G_{submult} the multiplication subgadget introduced in [8]. If both matrices γ and δ satisfy Condition 2, then G_{submult} is $(d-1)$ -NI and $(d-1, 2d-1)$ -partial NI.

The condition on t and n on Lemma 7 implies that the maximum amplification order for the multiplication gadget cannot be achieved for an odd number of shares (since the maximum order is reached when $t = \lfloor \frac{n-1}{2} \rfloor$). This is not a proof artifact but a limitation of the gadget G_{submult} with respect to the new $(d-1, 2d-1)$ -partial NI property. We can easily show that under this extreme conditions on t and n , we have $2d-1 = n$. If we consider the instantiation of G_{submult} for $n = 3$ input shares, we obtain the following $2n-1 = 5$ output shares:

$$\begin{aligned} c_1 &= (a_1 + (r_2 + a_2) + (r_3 + a_3)) \cdot (b_1 + (s_2 + b_2) + (s_3 + b_3)) \\ c_2 &= -r_2 \cdot (b_1 + (\delta_{1,1} \cdot s_2 + b_2) + (\delta_{1,2} \cdot s_3 + b_3)) \\ c_3 &= -r_3 \cdot (b_1 + (\delta_{2,1} \cdot s_2 + b_2) + (\delta_{2,2} \cdot s_3 + b_3)) \\ c_4 &= -s_2 \cdot (a_1 + (\gamma_{1,1} \cdot r_2 + a_2) + (\gamma_{1,2} \cdot r_3 + a_3)) \\ c_5 &= -s_3 \cdot (a_1 + (\gamma_{2,1} \cdot r_2 + a_2) + (\gamma_{2,2} \cdot r_3 + a_3)) \end{aligned}$$

To prove the $(d-1, 2d-1)$ -partial NI property, we need to ensure that any set of at most $2d-1 = 3$ probes can be perfectly simulated from at most $d-1 = 1$ shares of one of the inputs and any number of shares from the other one. However, the three probes on c_1, c_3, c_4 reveal information on each of their sub-product. In particular, $(a_1 + (r_2 + a_2) + (r_3 + a_3))$ (from c_1), r_3 (from c_3) and $(a_1 + (\gamma_{1,1} \cdot r_2 + a_2) + (\gamma_{1,2} \cdot r_3 + a_3))$ (from c_4) would reveal $\hat{a}$. Similarly, $(b_1 + (s_2 + b_2) + (s_3 + b_3))$ (from c_1), $(b_1 + (\delta_{2,1} \cdot s_2 + b_2) + (\delta_{2,2} \cdot s_3 + b_3))$ (from c_3) and s_2 (from c_4) would reveal $\hat{b}$. Hence, the gadget is not $(d-1, 2d-1)$ -partial NI. This counterexample with 3 shares can be directly extended to any odd number of shares.

This counterexample motivates a new construction for G_{submult} which would cover all values for n and t . In the following, we slightly modify the construction from [8] to achieve the maximum amplification order in any setting.

Remark 2. The current construction of G_{submult} outputs $m = 2n - 1$ shares, which does not satisfy the requirement $m \geq 2n$ shares for the compression gadget. Nevertheless, it is enough to add an artificial extra share c_{2n-1} equal to zero between both building blocks. In particular, the compression gadget (and subsequently the refresh gadget) does not expect the input sharing to be uniform to achieve the stated security properties.

New Construction for G_{submult} . As stated earlier, Lemma 7 does not hold for G_{submult} in the case where n is odd and $t = (n - 1)/2$. In order to cover this case, we propose a slightly modified version of G_{submult} with two extra random values r_1 and s_1 . In this version, we let $\gamma = (\gamma_{i,j})_{1 \leq i,j \leq n} \in \mathbb{F}_q^{n \times n}$ be a constant matrix, and let $\delta \in \mathbb{F}_q^{n \times n}$ be the matrix defined by $\delta_{i,j} = 1 - \gamma_{i,j}$. The sub-gadget G_{submult} outputs $2n + 1$ shares:

- $c_1 = \left(\sum_{i=1}^n (r_i + a_i) \right) \cdot \left(\sum_{i=1}^n (s_i + b_i) \right)$
- $c_{i+1} = -r_i \cdot \left(\sum_{j=1}^n (\delta_{i,j} s_j + b_j) \right)$ for $i = 1, \dots, n$
- $c_{i+n+1} = -s_i \cdot \left(\sum_{j=1}^n (\gamma_{i,j} r_j + a_j) \right)$ for $i = 1, \dots, n$

where r_i and s_i are randomly generated values. It can be easily checked that G_{submult} now performs $2n + 1$ bilinear multiplications, and that it is correct, i.e. $\sum_{i=1}^{2n+1} c_i = \sum_{i=1}^n a_i \cdot \sum_{i=1}^n b_i$.

We now need the following slightly modified version of Condition 2 on γ and on δ , which instead of considering a linear combination of at most $n - 1$ probes as in Condition 2, considers up to n probes:

Condition 3 Let $z = (2n + 4) \cdot n$. Let $\mathbf{I}_n \in \mathbb{F}_q^{n \times n}$ be the identity matrix, $\mathbf{0}_{\ell \times n} \in \mathbb{F}_q^{\ell \times n}$ be the matrix of zeros, $\mathbf{1}_{\ell \times n} \in \mathbb{F}_q^{\ell \times n}$ be the matrix of ones, $\mathbf{D}_{\gamma,j} \in \mathbb{F}_q^{n \times n}$ be the diagonal matrix such that $\mathbf{D}_{\gamma,j,i,i} = \gamma_{j,i}$, $\mathbf{T}_n \in \mathbb{F}_q^{n \times n}$ be the upper triangular matrix with just ones, $\mathbf{T}_{\gamma,j} \in \mathbb{F}_q^{n \times n}$ be the upper triangular matrix such that $\mathbf{T}_{\gamma,j,i,k} = \gamma_{j,i}$ for $i \leq k$. We define the following matrices:

$$\begin{aligned} \mathbf{L} &= \left[\begin{array}{c|c|c|c|c|c|c|c|c|c} \mathbf{I}_n & \mathbf{0}_{n \times n} & \mathbf{I}_n & \mathbf{I}_n & \dots & \mathbf{I}_n & \mathbf{T}_n & \mathbf{T}_n & \dots & \mathbf{T}_n \end{array} \right] \\ \mathbf{M} &= \left[\begin{array}{c|c|c|c|c|c|c|c|c|c} \mathbf{0}_{n \times n} & \mathbf{I}_n & \mathbf{I}_n & \mathbf{D}_{\gamma,1} & \dots & \mathbf{D}_{\gamma,n} & \mathbf{T}_n & \mathbf{T}_{\gamma,1} & \dots & \mathbf{T}_{\gamma,n} \end{array} \right] \end{aligned}$$

Then we say that γ satisfies Condition 3 if and only if

- for any vector $\mathbf{v} \in \mathbb{F}_q^z$ of Hamming weight $\text{hw}(\mathbf{v}) \leq n$,
- for any $i_1, \dots, i_K \in [z]$ such that $v_{i_1} \neq 0, \dots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \dots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient (i.e there are K probes of the form $\sum_{i=1}^n (r_i + a_i)$ or $\sum_{j=1}^n (\gamma_{i,j} r_j + a_j)$ for any $i = 1, \dots, n$),

if $\mathbf{M} \cdot \mathbf{v} = 0$, then we have $\text{hw}(\mathbf{L} \cdot \mathbf{v}) \leq \text{hw}(\mathbf{v}) - K$.

Under this new condition, we obtain the following result.

Lemma 8. Let $t \leq n - 1$ and $d = \min(t + 1, n - t)$. Let G_{submult} as defined above with n -share inputs. If both matrices γ and δ satisfy Condition 3, then G_{submult} is $(d - 1)$ -NI and $(d - 1, 2d - 1)$ -partial NI.

Proof. The proof of the Lemma is in fact the same as the proof of Lemma 7. The only difference is that in this lemma, we also cover the special case of an odd value for the number of shares n and $t = \lfloor \frac{n-1}{2} \rfloor = \frac{n-1}{2}$. In the latter case, we consider in the proof up to n probes on the gadget G_{submult} , while in Lemma 7, we could only have up to $n-1$ probes on the gadget. Since Condition 3 covers the case of having up to n probes on G_{submult} , then we can follow the exact same procedure of the proof of Lemma 7 to prove the Lemma by considering the new condition. $\square$

Remark 3. The number of output shares $m = 2n + 1$ of G_{submult} satisfies the constraint required by G_{compress} in Algorithm 1 ($m \geq 2n$). We can thus use the compression gadget G_{compress} exactly as described in the algorithm on the input sharing $(c_1, \dots, c_{2n+1})$, instantiated with the $\mathcal{O}(n \log n)$ refresh gadget from Section 4. Since the multiplication sub-gadget G_{submult} requires $\mathcal{O}(n)$ random values and G_{compress} requires $\mathcal{O}(n \log n)$ random values from the refresh gadget, the overall multiplication gadget G_{mult} also requires a quasi-linear number of random values $\mathcal{O}(n \log n)$.

5.4 Instantiations

We first state the existence of a matrix γ which satisfies Condition 3 over any finite field $\mathbb{F}_q$ for q large enough (with $\log(q) = \Omega(n \log n)$)⁶. The proof technique follows closely the proof of [8, Theorem 4.5] and makes use of the non-constructive “probabilistic method”. Specifically, it states that if one chooses γ uniformly at random in $\mathbb{F}_q^{n \log n}$, the probability that the matrix γ satisfies Condition 3 is strictly positive, when q is large enough. It is important to note that the proof relies on probability but the existence of a matrix γ which satisfies Condition 3 (for q large enough) is guaranteed without any possible error.

Theorem 4. *For any $n \geq 1$, for any prime power q , if γ is chosen uniformly in $\mathbb{F}_q^{n \times n}$, then*

$$\Pr[\gamma \text{ satisfies Condition 3}] \geq 1 - 2 \cdot (12n)^n \cdot n \cdot q^{-1}.$$

In particular, for any $n \geq 1$, there exists an integer $Q = \mathcal{O}(n)^{n+1}$, such that for any prime power $q \geq Q$, there exists a matrix $\gamma \in \mathbb{F}_q^{n \times n}$ satisfying Condition 3.

As when γ is uniformly random, so is δ , Theorem 4 immediately follows from the following proposition and the union bound.

Proposition 1. *For any $n \geq 1$, for any prime power q , if γ is chosen uniformly in $\mathbb{F}_q^{n \times n}$, then*

$$\Pr[\gamma \text{ satisfies Condition 3}] \geq 1 - (12n)^n \cdot n \cdot q^{-1}.$$

In particular, for any $n \geq 1$, there exists an integer $Q = \mathcal{O}(n)^{n+1}$, such that for any prime power $q \geq Q$, there exists a matrix $\gamma \in \mathbb{F}_q^{n \times n}$ satisfying Condition 3.

The proof of this proposition is very technical but follows essentially the proof of the analogous [8, Proposition 4.6]. It is provided in Appendix A.12.

In [8], Belaïd *et al.* presented examples of matrices which satisfy their condition for 2 shares and 3 shares. Karpman and Roche [19] proposed afterwards new explicit instantiations up to order $n = 6$ over large finite fields and up to $n = 4$ over practically relevant fields such as $\mathbb{F}_{256}$. It is worth mentioning that the matrices proposed in [19] are actually incorrect (due to a sign error) but this can be easily fixed and we check that matrices obtained following [19] also achieve our Condition 3. These matrices for 3, 4 and 5 shares are provided in Appendix A.13.

⁶ Such large finite fields may actually be useful to build efficient symmetric primitives (see for instance MiMC [2]).

6 Improved Asymptotic Complexity

In the previous sections, we exhibit the construction of a multiplication gadget G_{mult} which performs a linear number of multiplications between variables, and a quadratic number of multiplications by a constant operations. Using the results of Lemmas 5, 8 and 4, the constructed multiplication gadget is RPE and achieves the maximum amplification order $\lfloor \frac{n+1}{2} \rfloor$ for any number of shares n .

Using the three linear gadgets proposed in Section 4 (G_{add} , G_{copy} , G_{cmult}) with the $\mathcal{O}(n \log n)$ refresh gadgets, and the proposed construction of the multiplication gadget G_{mult} , we get an expanding compiler with a complexity matrix M_{CC} of eigenvalues:

$$(\lambda_1, \lambda_2) = (n, 6n \log(n) - 2n), \quad \lambda_3 = n, \quad \lambda_4 = 2n + 1 \quad \text{and} \quad \lambda_5 = n.$$

Hence we have $N_{\text{max}} = 6n \log(n) - 2n = \mathcal{O}(n \log n)$.

Figure 3 illustrates the evolution of the complexity exponent with respect to the number of shares n , for the best construction provided in [10] with quadratic complexity for an expanding compiler (orange curve), and our new construction with quasi-linear complexity (pink curve). While the best construction from [10] yields a complexity in $\mathcal{O}(|C| \cdot \kappa^e)$ for e close to 3 for reasonable numbers of shares, the new expanding compiler quickly achieves a sub-quadratic complexity in the same settings.

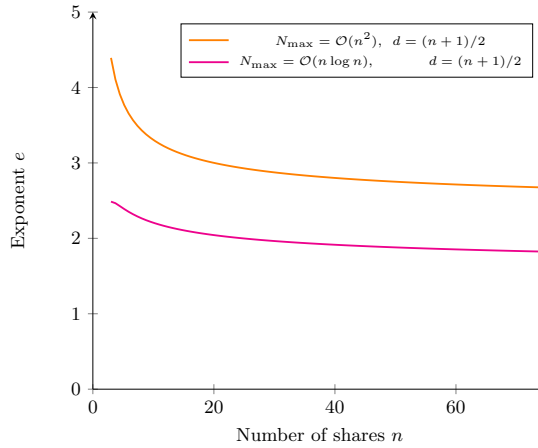


Fig. 3: Evolution of the complexity exponent $e = \log(N_{\text{max}})/\log(d)$ with respect to the number of shares n . The orange curve matches the instantiation from [10] with quadratic asymptotic complexity ($N_{\text{max}} = \mathcal{O}(n^2)$); the pink curve matches the new construction with quasi-linear asymptotic complexity ($N_{\text{max}} = \mathcal{O}(n \log n)$).

7 Conclusion

In this paper we have put forward a dynamic expansion strategy for random probing security which can make the most of different RPE gadgets in terms of tolerated leakage probability and asymptotic complexity. We further introduce new generic constructions of gadgets achieving RPE

for any number of shares n . When the base finite field of the circuit meets the requirement of our multiplication gadget, the asymptotic complexity of the obtained expanding compiler becomes arbitrary close to linear, which is optimal.

As for concrete instantiations, our small example on the AES demonstrates the benefits of our dynamic approach. Namely, it provides the best tolerated probability (from the best suited compiler) while optimizing the complexity using higher numbers of shares. Using two compilers with 3 and 5 shares instead of a single one already reduces the complexity by a factor 10.

To go further in the concrete use of our expanding compiler, future works could exhibit explicit constructions of matrices with (quasi)constant field size for our multiplication gadget. One could also investigate further designs of RPE multiplication gadgets with linear number of multiplications for arbitrary fields. Another interesting direction is to optimize the tolerated leakage probability for a set of (possibly inefficient) small gadgets to be used as starting point of the expansion in our dynamic approach before switching to more (asymptotically) efficient RPE gadgets.

References

1. Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 715–724. ACM Press, June 2011.
2. Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient encryption and cryptographic hashing with minimal multiplicative complexity. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *ASIACRYPT 2016, Part I*, volume 10031 of *LNCS*, pages 191–219. Springer, Heidelberg, December 2016.
3. Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 427–455. Springer, Heidelberg, August 2018.
4. Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 586–615. Springer, Heidelberg, May 2016.
5. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Heidelberg, August 2016.
6. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. Cryptology ePrint Archive, Report 2016/540, 2016. <https://eprint.iacr.org/2016/540>.
7. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 616–648. Springer, Heidelberg, May 2016.
8. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 397–426. Springer, Heidelberg, August 2017.
9. Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 339–368. Springer, Heidelberg, August 2020.
10. Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 313–343. Springer, Heidelberg, October 2021.
11. Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. Cryptology ePrint Archive, Report 2016/321, 2016. <https://eprint.iacr.org/2016/321>.
12. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 398–412. Springer, Heidelberg, August 1999.

13. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shihō Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 410–424. Springer, Heidelberg, March 2014.
14. Jean-Sébastien Coron, Franck Rondepierre, and Rina Zeitoun. High order masking of look-up tables with common shares. Cryptology ePrint Archive, Report 2017/271, 2017. <https://eprint.iacr.org/2017/271>.
15. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Heidelberg, May 2014.
16. Louis Goubin and Jacques Patarin. DES and differential power analysis (the “duplication” method). In Çetin Kaya Koç and Christof Paar, editors, *CHES’99*, volume 1717 of *LNCS*, pages 158–172. Springer, Heidelberg, August 1999.
17. Hannes Groß, Ko Stoffelen, Lauren De Meyer, Martin Krenn, and Stefan Mangard. First-order masking with only two random bits. In Begül Bilgin, Svetla Petkova-Nikova, and Vincent Rijmen, editors, *Proceedings of ACM Workshop on Theory of Implementation Security Workshop, TIS@CCS 2019, London, UK, November 11, 2019*, pages 10–23. ACM, 2019.
18. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Heidelberg, August 2003.
19. Pierre Karpman and Daniel S. Roche. New instantiations of the CRYPTO 2017 masking schemes. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 285–314. Springer, Heidelberg, December 2018.
20. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *CHES 2010*, volume 6225 of *LNCS*, pages 413–427. Springer, Heidelberg, August 2010.
21. Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *CT-RSA 2006*, volume 3860 of *LNCS*, pages 208–225. Springer, Heidelberg, February 2006.

A Proofs

A.1 Proof of Theorem 1

We consider that we have ℓ compilers $CC_1, \dots, CC_\ell$, and we want to prove the following result:

Lemma 9. *Let $CC_1, \dots, CC_\ell$ RPE compilers with expanding functions $f_1, \dots, f_\ell$. The dynamic expanding compiler for $CC_1, \dots, CC_\ell$, which on input circuit C outputs the compiled circuit $CC_\ell \circ \dots \circ CC_1(C)$, is an RPE compiler with expanding function f such that*

$$f = f_\ell \circ \dots \circ f_1.$$

It can be seen that proving Lemma 9 implies proving the result of Theorem 1. Indeed, we can replace ℓ in the lemma by $k_1 + \dots + k_\mu$ from Theorem 1 and consider the corresponding compilers with their expansion levels. Thus, we will prove in this appendix Lemma 9 and the proof of the Theorem will follow directly.

To prove the lemma, we first start by introducing some definitions from [9] for random probing expandability of level- ℓ with different sharing orders $n_1, \dots, n_\ell$ gadgets. First, we introduce a generalized definition of *adequate subsets* of $[n_1 \times \dots \times n_\ell]$ as in [9]. For this, we define recursively a family $S_k \in \mathcal{P}([n_1 \times \dots \times n_k])$ for $k \leq \ell$, where $\mathcal{P}([n_1 \times \dots \times n_k])$ denotes the set of all subsets of $[n_1 \times \dots \times n_k]$, as follows:

$$\begin{aligned} S_1(n, t) &= \{I \in [n], |I| \leq t\} \\ S_k(\{n_i\}_{i \in [k]}, \{t_i\}_{i \in [k]}) &= \{(I_1, \dots, I_{n_k}) \in (S_{k-1}(\{n_i\}_{i \in [k-1]}, \{t_i\}_{i \in [k-1]}) \cup [n_1 \times \dots \times n_{k-1}])^{n_k}, \\ &\quad I_j \in S_{k-1} \ \forall j \in [1, n_k] \text{ except at most } t_k\} \end{aligned}$$

In other words, a subset I belongs to S_k if among the n_k subset parts of I , at most t_k of them are full, while the other ones recursively belong to S_{k-1} . For simplicity, we will sometimes denote S_k without the parameters $(\{n_i\}_{i \in [k]}, \{t_i\}_{i \in [k]})$ which will be implicit in the notation. We will also denote for simplicity $N_i = n_1 \cdot \dots \cdot n_i$ for $i \in \mathbb{N}$.

Then we recall the generalized definition of RPE with S_k for level- k gadgets.

Definition 12 (Random Probing Expandability with $\{S_k\}_{k \in \mathbb{N}}$). *Let $f : \mathbb{R} \rightarrow \mathbb{R}$ and $k \in \mathbb{N}$. An N_k -share gadget $G : \mathbb{K}^{N_k} \times \mathbb{K}^{N_k} \rightarrow \mathbb{K}^{N_k}$ is (S_k, f) -random probing expandable (RPE) if there exists a deterministic algorithm Sim_1^G and a probabilistic algorithm Sim_2^G such that for every input $(\hat{x}, \hat{y}) \in \mathbb{K}^{N_k} \times \mathbb{K}^{N_k}$, for every set $J \in S_k \cup [N_k]$ and for every $p \in [0, 1]$, the random experiment*

$$\begin{aligned} W &\leftarrow \text{LeakingWires}(G, p) \\ (I_1, I_2, J') &\leftarrow \text{Sim}_1^G(W, J) \\ \text{out} &\leftarrow \text{Sim}_2^G(W, J', \hat{x}|_{I_1}, \hat{y}|_{I_2}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1 \equiv (I_1 \notin S_k)$ and $\mathcal{F}_2 \equiv (I_2 \notin S_k)$ verify

$$\Pr(\mathcal{F}_1) = \Pr(\mathcal{F}_2) = \varepsilon \quad \text{and} \quad \Pr(\mathcal{F}_1 \wedge \mathcal{F}_2) = \varepsilon^2 \tag{19}$$

with $\varepsilon = f(p)$ (in particular $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent),

2. the set J' is such that $J' = J$ if $J \in S_k$, and $J' = [N_k] \setminus \{j^*\}$ for some $j^* \in [N_k]$ otherwise,
3. the output distribution satisfies

$$out \stackrel{id}{=} (\text{AssignWires}(G, W, (\hat{x}, \hat{y})), \hat{z}|_{J'}) \quad (20)$$

where $\hat{z} = G(\hat{x}, \hat{y})$.

We are now ready to prove Lemma 9.

Proof (Lemma 9). We will prove the Lemma recursively. In other words, we will suppose that we have RPE compilers $CC_1, \dots, CC_k$ with expanding functions $f_1, \dots, f_k$ and (t_i, f_i) -RPE gadgets for each $i \leq k$, and we will prove that the gadgets of the expanding compiler $CC_k \circ \dots \circ CC_1$ are (S_k, f) -RPE with $f = f_k \circ \dots \circ f_1$. This will imply that the expanding compiler $CC_k \circ \dots \circ CC_1$ is RPE with expanding function f .

The base case is one of the theorem hypotheses, namely for $k = 1$, the level-1 gadgets are (t_1, f_1) -RPE, which is equivalent to (S_1, f_1) -RPE. We must then show the induction step: assuming that the level- k gadgets are $(S_k, f_k \circ \dots \circ f_1)$ -RPE, show that the level- $(k+1)$ gadgets are $(S_{k+1}, f_{k+1} \circ \dots \circ f_1)$ -RPE. For the sake of simplicity, we depict our proof by assuming that all the gadgets are 2-to-1 gadget (which is actually not the case for copy gadgets). The proof mechanism for the general case (with 2-to-1 and 1-to-2 gadgets) is strictly similar but heavier on the form. We also denote in the following

- $\varepsilon_k = f_k \circ \dots \circ f_1(p)$,
- G^{CC_k} to be a gadget of the expanding compiler CC_k ,
- $G^{(k)}$ to be the gadget resulting from applying $CC_{k-1} \circ \dots \circ CC_1(G^{CC_k})$, *i.e.* obtained by replacing each gate of the base gadget G^{CC_k} by the corresponding level- $(k-1)$ gadget $G^{(k-1)}$ and by replacing each wire of the base gadget by N_{k-1} wires carrying a N_{k-1} -linear sharing of the original wire.

In order to show that a gadget $G^{(k+1)}$ is $(S_{k+1}, \varepsilon_{k+1})$ -RPE we must construct two simulators $\text{Sim}_1^{G^{(k+1)}}$ and $\text{Sim}_2^{G^{(k+1)}}$ that satisfy the conditions of Definition 12 for the set of subsets S_{k+1} . More precisely, we must construct two simulators $\text{Sim}_1^{G^{(k+1)}}$ and $\text{Sim}_2^{G^{(k+1)}}$ such that for every $(\hat{x}^*, \hat{y}^*) \in \mathbb{K}^{N_{k+1}} \times \mathbb{K}^{N_{k+1}}$, and for every set $J^* \in S_{k+1} \cup [N_{k+1}]$, the random experiment

$$\begin{aligned} W^* &\leftarrow \text{LeakingWires}(G^{(k+1)}, p) \\ (I_1^*, I_2^*, J^{*'}) &\leftarrow \text{Sim}_1^{G^{(k+1)}}(W^*, J^*) \\ out &\leftarrow \text{Sim}_2^{G^{(k+1)}}(W^*, J^*, \hat{x}^*|_{I_1^*}, \hat{y}^*|_{I_2^*}) \end{aligned}$$

ensures that

1. the failure events $\mathcal{F}_1^* \equiv (I_1^* \notin S_{k+1})$ and $\mathcal{F}_2^* \equiv (I_2^* \notin S_{k+1})$ verify

$$\Pr(\mathcal{F}_1^*) = \Pr(\mathcal{F}_2^*) = \varepsilon_{k+1} \quad \text{and} \quad \Pr(\mathcal{F}_1^* \wedge \mathcal{F}_2^*) = \varepsilon_{k+1}^2 \quad (21)$$

2. the set $J^{*'}$ is such that $J^{*'} = J^*$ if $J^* \in S_{k+1}$ and $J^{*'} = [N_{k+1}] \setminus \{j^*\}$ otherwise,
3. the output distribution satisfies

$$out \stackrel{id}{=} (\text{AssignWires}(G^{(k+1)}, W, (\hat{x}, \hat{y})), \hat{z}|_{J^{*'}}) \quad (22)$$

where $\hat{z} = G^{(k+1)}(\hat{x}, \hat{y})$.

We distinguish two cases: either $J^* \in S_{k+1}$ (normal case), or $J^* = [N_{k+1}]$ (saturated case).

Normal case: $J^* \in S_{k+1}$. By definition of the expanding compiler, we have that a level- $(k+1)$ gadget $G^{(k+1)}$ is obtained by replacing each gate of the base gadget $G^{CC_{k+1}}$ of the compiler CC_{k+1} by the corresponding level- k gadget $G^{(k)}$ and by replacing each wire of the base gadget by N_k wires carrying a N_k -linear sharing of the original wire. In particular $G^{(k+1)}$ has N_{k+1} output wires which can be split in n_{k+1} groups of N_k wires, each group being the output of a different $G^{(k)}$ gadget. We split the set J^* accordingly so that $J^* = J_1^* \cup \dots \cup J_{n_{k+1}}^*$, where each set J_i^* pertains to the i th group of output wires. By definition of S_k , since $J^* \in S_{k+1}$, we must have $J_i^* \in S_k$ for all $1 \leq i \leq n_{k+1}$, except at most t_{k+1} of them for which $J_i^* = [N_k]$. We define J_{base} as the set of indexes i such that $J_i^* \notin S_k$. Therefore we must have $|J_{\text{base}}| \leq t_{k+1}$.

We first describe the simulator $\text{Sim}_1^{G^{(k+1)}}$ that takes the leaking wires W^* and the output wires $J^* \in S_{k+1}$ to be simulated and produce the sets $I_1^* \subseteq [N_{k+1}]$ and $I_2^* \subseteq [N_{k+1}]$ of required inputs. The simulator $\text{Sim}_1^{G^{(k+1)}}$ starts by defining a set W_{base} which is initialized to $\emptyset$; this will correspond to the set of leaking wires for the base gadget $G^{CC_{k+1}}$. Then the simulation goes through all the level- k gadgets composing $G^{(k+1)}$ from bottom to top *i.e.* starting with the level- k gadgets producing the output sharing up to the level- k gadgets processing the input sharings. Let us denote by $\{G_j^{(k)}\}_j$ these level- k gadgets. For each $G_j^{(k)}$, one runs the simulator Sim_1 from the $(S_k, f_k \circ \dots \circ f_1)$ -RPE property on input W_j and J_j defined as follows. The set of leaking wires W_j is defined as the subset of W^* corresponding to the wires of $G_j^{(k)}$. For the gadgets $G_j^{(k)}$ on the bottom layer, the set J_j is set to one of the J_i^* (with indices scaled to range in $[N_k]$). For all the other gadgets $G_j^{(k)}$ (which are not on the bottom layer), the set J is defined as the set I_1 or I_2 output from Sim_1 for the child gadget $G_{j'}^{(k)}$ (for which Sim_1 has already been run).

Whenever a failure event occurs for a $G_j^{(k)}$ gadget, namely when the set I (either I_1 or I_2) output from Sim_1 is such that $I \notin S_k$, we add the index of the wire corresponding to this input in the base gadget $G^{CC_{k+1}}$ to the set W_{base} . Once the Sim_1 simulations have been run for all the $G_j^{(k)}$ gadgets, ending with the top layers, we get the final sets I corresponding to the input shares. Each of these sets corresponds to an N_k -sharing as input of a $G_j^{(k)}$ gadget, which corresponds to a wire as input of the base gadget among the $2 \cdot n_{k+1}$ wires carrying the two input n_{k+1} -sharings of the base gadget. We denote by $I_{1,1}^*, \dots, I_{1,n_{k+1}}^*$ and $I_{2,1}^*, \dots, I_{2,n_{k+1}}^*$ the corresponding sets so that defining

$$I_1^* = I_{1,1}^* \cup \dots \cup I_{1,n_{k+1}}^* \quad \text{and} \quad I_2^* = I_{2,1}^* \cup \dots \cup I_{2,n_{k+1}}^* , \quad (23)$$

the tuple $\hat{x}^*|_{I_1^*}$ and $\hat{y}^*|_{I_2^*}$ contains the shares designated by the final I sets.

At the end of the $\text{Sim}_1^{G^{(k+1)}}$ simulation, the set W_{base} contains all the labels of wires in the base gadget $G^{CC_{k+1}}$ for which a failure event has occurred in the simulation of the corresponding $G_j^{(k)}$ gadget. Thanks to the (S_k, ε_k) -RPE property of these gadgets, the failure events happen (mutually independently) with probability ε_k which implies

$$W_{\text{base}} \stackrel{\text{id}}{=} \text{LeakingWires}(G^{CC_{k+1}}, \varepsilon_k) \quad (24)$$

Recall that $|J_{\text{base}}| \leq t_{k+1}$. We can then run $\text{Sim}_1^{G^{CC_{k+1}}}$ to obtain:

$$(I_{1,\text{base}}, I_{2,\text{base}}) = \text{Sim}_1^{G^{CC_{k+1}}}(W_{\text{base}}, J_{\text{base}}) . \quad (25)$$

For all $1 \leq i \leq n_{k+1}$, if $i \in I_{1,\text{base}}$, we force $I_{1,i}^* \leftarrow [N_k]$, so that the corresponding i -th input wire of the base gadget can be computed from the corresponding input wires in $I_{1,i}^*$. The simulator $\text{Sim}_1^{G^{(k+1)}}$ then returns (I_1^*, I_2^*) as output.

The (t_{k+1}, f_{k+1}) -RPE property of the base gadget $G^{CC_{k+1}}$ implies that the *base failure events* $|I_{1,\text{base}}| = n_{k+1}$ and $|I_{2,\text{base}}| = n_{k+1}$ are ε_{k+1} -mutually unlikely, where $\varepsilon_{k+1} = f_{k+1}(\varepsilon_k)$. We argue that for all $1 \leq i \leq n_{k+1}$, $I_{1,i}^* \notin S_k \iff i \in I_{1,\text{base}}$. Namely if a failure event has occurred for a set $I_{1,i}^*$ (i.e. $I_{1,i}^* \notin S_k$) then we must have $i \in I_{1,\text{base}}$. Indeed, if a failure event has occurred for a set $I_{1,i}^*$ then the label of the i th input wire (for the first sharing) of the base gadget $G^{CC_{k+1}}$ has been added to W_{base} and $\text{Sim}_1^{G^{CC_{k+1}}}$ has no choice but to include this index to the set $I_{1,\text{base}}$ so that $\text{Sim}_2^{G^{CC_{k+1}}}$ can achieve a perfect simulation of the wire assignment (as required by the RPE property of $G^{CC_{k+1}}$). Moreover if $i \in I_{1,\text{base}}$ then by construction we have set $I_{1,i}^* = [N_k]$ and therefore $I_{1,i}^* \notin S_k$. This implies that if $|I_{1,\text{base}}| \leq t_{k+1}$ then $I_1^* \in S_{k+1}$ (and the same happens for I_2^* w.r.t. $I_{2,\text{base}}$). We deduce that the failure events $\mathcal{F}_1^*$ and $\mathcal{F}_2^*$ are also ε_{k+1} -mutually unlikely, as required by the $(S_{k+1}, \varepsilon_{k+1})$ -RPE property of $G^{(k+1)}$.

We now describe the simulator $\text{Sim}_2^{G^{(k+1)}}$ that takes as input $\hat{x}^*|_{I_1^*}$ and $\hat{y}^*|_{I_2^*}$ and produces a perfect simulation of $(\text{AssignWires}(G^{(k+1)}, W^*, (\hat{x}^*, \hat{y}^*)), \hat{z}|_{J^*})$ where $\hat{z} = G^{(k+1)}(\hat{x}, \hat{y})$. Let $\hat{x}^b$ and $\hat{y}^b$ denote the n_{k+1} -linear sharings obtained by applying the linear decoding to each group of N_k shares in $\hat{x}^*$ and $\hat{y}^*$, so that the elements of $\hat{x}^b$ and $\hat{y}^b$ correspond to the input wires in the base gadget $G^{CC_{k+1}}$. The assignment expansion property implies that a perfect assignment of the wires of $G^{(k+1)}$ on input $\hat{x}^*$ and $\hat{y}^*$ can be derived from an assignment of the wires of the base gadget $G^{CC_{k+1}}$ on input $\hat{x}^b$ and $\hat{y}^b$. The simulator makes use of this property by first running

$$\text{out}_{\text{base}} \leftarrow \text{Sim}_2^{G^{CC_{k+1}}}(W_{\text{base}}, J_{\text{base}}, \hat{x}^b|_{I_{1,\text{base}}}, \hat{y}^b|_{I_{2,\text{base}}}), \quad (26)$$

Note that the input values $\hat{x}^b|_{I_{1,\text{base}}}$ and $\hat{y}^b|_{I_{2,\text{base}}}$ can be obtained from the corresponding shares in I_1^* and I_2^* . Thanks to the (t_{k+1}, f_{k+1}) -RPE property of $G^{CC_{k+1}}$ and by construction of $I_{1,\text{base}}$ and $I_{2,\text{base}}$, this outputs a distribution satisfying

$$\text{out}_{\text{base}} \stackrel{\text{id}}{=} \left(\text{AssignWires}(G^{CC_{k+1}}, W_{\text{base}}, (\hat{x}^b, \hat{y}^b)), \hat{z}^b|_{J_{\text{base}}} \right) \quad (27)$$

The simulator then goes through all the $G_j^{(k)}$ gadgets from input to output and for each of them runs the simulator Sim_2 of the RPE property on inputs W_j , J_j , $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$ where W_j and J_j are the sets from the first phase of the simulation for the gadget $G_j^{(k)}$, I_1 and I_2 are the corresponding sets produced by the Sim_1 simulator for $G_j^{(k)}$, and $\hat{x}$ and $\hat{y}$ are the inputs of $G_j^{(k)}$ in the evaluation of $G^{(k+1)}(\hat{x}^*, \hat{y}^*)$. Provided that the partial inputs $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$ are perfectly simulated, this call to Sim_2 produces a perfect simulation of $(\text{AssignWires}(G_j^{(k)}, W_j, (\hat{x}, \hat{y}), \hat{z}|_{J_j}))$ where $\hat{z} = G_j^{(k)}(\hat{x}, \hat{y})$. In order to get perfect simulations of the partial inputs $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$, the simulator proceeds as follows. For the top layer of $G^{(k)}$ gadgets (the ones processing the input shares) the shares $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$ can directly be taken from the inputs $\hat{x}^*|_{I_1^*}$ and $\hat{y}^*|_{I_2^*}$. For the next gadgets the shares $\hat{x}|_{I_1}$ and $\hat{y}|_{I_2}$ match the shares $\hat{z}|_J$ output from the call to Sim_2 for a parent gadget. The only exception occurs in case of a failure event.

In that case the simulation needs the full input $\hat{x} = (x_1, \dots, x_{N_k})$ (and/or $\hat{y} = (y_1, \dots, y_{N_k})$), while we have set $|I_1| = N_k - 1$ (and/or $|I_2| = (N_k - 1)$) to satisfy the RPE requirements of the

parent gadget in the first simulation phase. Nevertheless, for such cases a perfect simulation of the plain value $x = \text{LinDec}(\hat{x})$ (and/or $y = \text{LinDec}(\hat{y})$) is included to out_{base} by construction of W_{base} . We can therefore perfectly simulate the missing share from the $N_k - 1$ other shares and the plain value x (or y). We thus get a perfect simulation of $(\text{AssignWires}(G_j^{(k)}, W_j, (\hat{x}, \hat{y}), \hat{z}|_{J_j}))$ for all the level- k gadgets $G_j^{(k)}$ which gives us a perfect simulation of $(\text{AssignWires}(G^{(k+1)}, W^*, (\hat{x}^*, \hat{y}^*), \hat{z}|_{J^*}))$.

Saturated case: $J^* = [N_{k+1}]$. The saturated case proceeds similarly. The difference is that we must simulate all N_{k+1} output shares of the level- $(k+1)$ gadget, except for one share index j^* that can be chosen by the simulator.

The simulator $\text{Sim}_1^{G^{(k+1)}}$ is defined as previously. Since $J^* = [N_{k+1}]$, we must define $J_{\text{base}} = [1, n_{k+1}]$. Moreover we have $J_i^* = [N_k]$ for all $1 \leq i \leq n_{k+1}$. This implies that for the gadgets $G_j^{(k)}$ on the output layer, the sets J_j are all equal to $[N_k]$ as well. The set W_{base} is defined as previously, and the simulator $\text{Sim}_1^{G^{(k+1)}}$ returns (I_1^*, I_2^*) as previously. The failure events $\mathcal{F}_1^*$ and $\mathcal{F}_2^*$ are still ε_{k+1} -mutually unlikely, as required by the $(S_{k+1}, \varepsilon_{k+1})$ -RPE property of $G^{(k+1)}$.

The simulator $\text{Sim}_2^{G^{(k+1)}}$ is defined as previously. In particular, from the running of the base gadget simulator $\text{Sim}_2^{G^{C_{k+1}}}$, we obtain a perfect simulation of the output wires $\hat{z}^b|_{J'_{\text{base}}}$ for some J'_{base} with $|J'_{\text{base}}| = n_{k+1} - 1$. Combined with the perfect simulation of the output wires corresponding to the output sets J'_j from the gadgets $G_j^{(k)}$ on the output layer, with $|J'_j| = N_k - 1$, we obtain a subset J' of output wires for our level- $(k+1)$ gadget with $|J'| = N_{k+1} - 1$ as required. Eventually this gives us a perfect simulation of $(\text{AssignWires}(G^{(k+1)}, W^*, (\hat{x}^*, \hat{y}^*), \hat{z}|_{J'}))$. This terminates the proof of Lemma 9. As stated earlier, proving Lemma 9 implies proving Theorem 1. Thus, this also terminates the proof for the theorem. $\square$

A.2 Proof of Theorem 2

Proof. Let $\{\text{CC}_i\}_i$ be a family of circuit compilers with complexity matrices $\{M_{\text{CC}_i}\}_i$. Given a circuit C with its complexity vector N_C as described in Section 2.4, it can be verified that the complexity of the compiled circuit $\hat{C} = \text{CC}_\mu^{k_\mu} \circ \dots \circ \text{CC}_1^{k_1}(C)$ satisfies

$$N_{\hat{C}} = M_{\text{CC}_\mu}^{k_\mu} \cdot \dots \cdot M_{\text{CC}_1}^{k_1} \cdot N_C$$

If we denote $M_{\text{CC}_i} = Q_i \cdot A_i \cdot Q_i^{-1}$ to be the eigen decomposition of the matrix M_{CC_i} , then we get

$$N_{\hat{C}} = Q_\mu \cdot A_\mu^{k_\mu} \cdot Q_\mu^{-1} \cdot \dots \cdot Q_1 \cdot A_1^{k_1} \cdot Q_1^{-1} \cdot N_C \quad (28)$$

We consider in the theorem that the expansion levels $\{k_i\}_i$ are the main parameters. We can also see from (28) that the complexity of the compiled circuit is expressed in terms of the eigen matrices to the powers k_i as $A_i^{k_i}$. The parameters $\{k_i\}_i$ do not affect the matrices $\{Q_i, Q_i^{-1}\}_i$. Then, if we denote $\lambda_i := \max \text{eigenvalues}(M_{\text{CC}_i})$ i.e. the maximum of the eigenvalues in A_i , then we get that in terms of the parameters $\{k_i\}_i$, the complexity of the compiled circuit $\hat{C}$ can be expressed as

$$N_{\hat{C}} = \mathcal{O}\left(|\lambda_\mu|^{k_\mu} \cdot \dots \cdot |\lambda_1|^{k_1}\right) \cdot N_C$$

which gives

$$|\hat{C}| = |C| \cdot \mathcal{O}\left(\prod_{i=1}^{\mu} |\lambda_i|^{k_i}\right)$$

which concludes the proof of Theorem 2. $\square$

A.3 Proof of Theorem 3

To prove Theorem 3, we introduce the following lemma.

Lemma 10. *Let CC_i be an RPE circuit compiler of amplification order d_i and complexity s_i . For any probability*

$$p \leq \frac{1}{2} \cdot \frac{d_i + 1}{s_i - d_i} \quad (29)$$

the expanding function f_i of CC_i is upper bounded by

$$f_i(p) \leq 2 \binom{s_i}{d_i} p^{d_i} . \quad (30)$$

Proof (Lemma 10). Let us first recall the following general bound on f_i :

$$f_i(p) \leq \sum_{j=d_i}^{s_i} \binom{s_i}{j} p^j , \quad (31)$$

for any $p \in [0, 1)$. From (29), for any $j \in [s_i]$, we get:

$$\binom{s_i}{j+1} p^{j+1} \leq \frac{1}{2} \binom{s_i}{j} p^j$$

which gives

$$f_i(p) \leq \sum_{j=d_i}^{s_i} \binom{s_i}{d_i} \left(\frac{1}{2}\right)^{j-d_i} p^{d_i} = \binom{s_i}{d_i} p^{d_i} \sum_{j=0}^{s_i-d_i} \left(\frac{1}{2}\right)^j \leq 2 \binom{s_i}{d_i} p^{d_i} .$$

□

Proof (Theorem 3). We show that for every p satisfying

$$p < \frac{1}{e} \left(\frac{1}{2e}\right)^{\frac{1}{d_i-1}} \left(\frac{d_i}{s_i}\right)^{1+\frac{1}{d_i-1}} \quad (32)$$

we have $f_i(p) < p$. Let us define

$$\bar{f}_i : p \mapsto 2 \binom{s_i}{d_i} p^{d_i} .$$

(the upper bound on f_i from Lemma 10). The equation $\bar{f}_i(\gamma) = \gamma$ has the following solution

$$\gamma = \left(\frac{1}{2 \binom{s_i}{d_i}}\right)^{\frac{1}{d_i-1}}$$

which, from

$$\binom{s_i}{d_i} \leq \left(\frac{s_i \cdot \exp(1)}{d_i}\right)^{d_i} ,$$

further satisfies

$$\gamma \geq \frac{1}{e} \left(\frac{1}{2e}\right)^{\frac{1}{d_i-1}} \left(\frac{d_i}{s_i}\right)^{1+\frac{1}{d_i-1}}$$

We deduce that (32) implies $p < \gamma$ which further implies $\bar{f}_i(p) < p$. Moreover (32) implies

$$p < \frac{1}{2} \left(\frac{d_i}{s_i} \right)^{1 + \frac{1}{d_i - 1}} < \frac{1}{2} \cdot \frac{d_i}{s_i} < \frac{1}{2} \cdot \frac{d_i + 1}{s_i - d_i},$$

which, by Lemma 10, further implies $f_i(p) \leq \bar{f}_i(p)$. We hence deduce that (32) implies $f_i(p) < p$ which concludes the proof. $\square$

A.4 Proof of Corollary 1

Proof (Corollary 1). For any function $f(p) = c \cdot p^d$, we have

$$f^{(k)}(p) = c^{(d^{k-1} + d^{k-2} + \dots + 1)} \cdot p^{d^k} \leq c^{(1 + \frac{1}{d-1})d^{k-1}} p^{d^k}.$$

When $c_i = 2^{\binom{s_i}{d_i}}$, Equation (11) from Theorem 3 gives the first and the second inequalities. $\square$

A.5 Proof of Lemma 6

Proof. G_{cmult} has the exact same wires as the underlying G_{refresh} except for the extra input wires $\{a_1, \dots, a_n\}$ (the wires multiplied by the constant i.e. $\{c \cdot a_1, \dots, c \cdot a_n\}$ are the input wires to G_{refresh}). So to simulate probes on G_{cmult} , we use the simulator of G_{refresh} . Each probe which is in the set $\{a_1, \dots, a_n\}$ will be replaced by the corresponding input share multiplied by the constant c , in the set of probes on G_{refresh} , which would lead to a probe on an input share of G_{refresh} of the form $c \cdot a_i$. It is clear that if we can perfectly simulate $c \cdot a_i$ in G_{refresh} , then we can perfectly simulate the input share a_i in G_{cmult} . Thus any set of probes on G_{cmult} is simulated using the simulator of G_{refresh} with the exact same number of probes. Hence, if G_{refresh} is (t, f) -(T)RPE n -share refresh gadget of amplification order d , then the gadget G_{cmult} is also (t, f') -(T)RPE of amplification order d . This concludes the proof. $\square$

A.6 Proof of Lemma 2

Proof. Let G_{refresh} be a (t, f) -TRPE refresh gadget for any $t \leq n - 1$ with amplification order $d \geq \min(t + 1, n - t)$ and which satisfies Definition 9. We will prove that the construction of G_{add} using G_{refresh} described in Section 4 is (t, f) -TRPE for any $t \leq n - 1$ of amplification order $\min(t + 1, n - t)$. This amounts to proving that:

1. Any set of leaking wires W such that $|W| < \min(t + 1, n - t)$ can be simulated together with any set of outputs wires $J \subseteq [n]$ from sets of input wires I_1 on a and I_2 on b such that $|I_1| \leq \min(t, |W|)$ and $|I_2| \leq \min(t, |W|)$.
2. Any set of leaking wires such that $\min(t + 1, n - t) \leq |W| < 2 \min(t + 1, n - t)$ can be simulated together with any set of outputs wires $J \subseteq [n]$ from sets of input wires I_1, I_2 such that $|I_1| \leq \min(t, |W|)$ or $|I_2| \leq \min(t, |W|)$ (because of the double failure, i.e. failure on both inputs).

Indeed, this amplification order being the maximum one achievable by 2-input addition gadgets, it would conclude the proof.

We will denote $(e_1, \dots, e_n) = G_{\text{refresh}}(a_1, \dots, a_n)$ and $(f_1, \dots, f_n) = G_{\text{refresh}}(b_1, \dots, b_n)$. Then the gadget G_{add} consists in the sharewise addition $(e_1 + f_1, \dots, e_n + f_n)$ as described in Section 4.

We proceed by building the necessary simulators for G_{add} from the simulators that already exist for G_{refresh} . Concretely, we split each set W of leaking wires, into four subsets $W = W_1^r \cup W_1^a \cup W_2^r \cup W_2^a$ where W_1^r (resp. W_2^r) is the set of leaking wires during the computation of $G_{\text{refresh}}(a_1, \dots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \dots, b_n)$), and W_1^a (resp. W_2^a) is the set of leaking wires of $(e_1, \dots, e_n)$ (resp. $(f_1, \dots, f_n)$). We can see that $W_1^r \cup W_1^a$ (resp. $W_2^r \cup W_2^a$) contains only leaking wires during the computation of $G_{\text{refresh}}(a_1, \dots, a_n)$ (resp. $G_{\text{refresh}}(b_1, \dots, b_n)$). We now demonstrate how we can simulate W when the output set J is of size less than t ((T)RPE1) and when it is of size strictly more than t ((T)RPE2).

– if $|J| \leq t$ ((T)RPE1): we prove both properties 1 and 2:

1. we assume that $|W| < \min(t+1, n-t)$. We construct a new set of probes on $(e_1, \dots, e_n)$ that we denote J_e such that $J_e = W_1^a \cup \{e_i \mid i \in J\}$. Similarly, we construct the set of probes on $(f_1, \dots, f_n)$, $J_f = W_2^a \cup \{f_i \mid i \in J\}$. It is clear that if we can perfectly simulate W_1^r , W_2^r , J_e and J_f , then we can perfectly simulate W , and J (for each $i \in J$, we can perfectly simulate e_i in J_e and f_i in J_f so we can perfectly simulate $e_i + f_i$). We denote $|W_1^r| = m$ and $|W_2^a| = m'$. We have

$$|W_1^r| \leq \min(t+1, n-t) - 1 - m \quad , \quad |J_e| \leq t + m$$

and

$$|W_2^r| \leq \min(t+1, n-t) - 1 - m' \quad , \quad |J_f| \leq t + m'$$

From the (t, f) -TRPE property of G_{refresh} for any $t \leq n-1$ and specifically for $t' = t + m$ with amplification order at least $d' = \min(t+1+m, n-t-m)$, and since $|W_1^r| \leq \min(t+1, n-t) - 1 - m \leq d' - 1$, then there exists an input set of shares of a I_1 such that $|I_1| \leq \min(t+m, |W_1^r|) = |W_1^r| \leq |W|$ and I_1 perfectly simulates W_1^r and J_e .

Similarly, there exists an input set of shares of b I_2 such that $|I_2| \leq \min(t+m', |W_2^r|) = |W_2^r| \leq |W|$ and I_2 perfectly simulates W_2^r and J_f .

From these definitions, I_1 and I_2 together perfectly simulate W and J and are both of size less than $\min(t, |W|)$, which proves the first property in this scenario.

2. we now assume that $\min(t+1, n-t) \leq |W| < 2\min(t+1, n-t)$. Without loss of generality, let us consider that $|W_1^r \cup W_1^a| < \min(t+1, n-t) \leq t$ (the proof is similar in the opposite scenario). As in the first property, we construct a new set of probes on $(e_1, \dots, e_n)$ that we denote J_e such that $J_e = W_1^a \cup \{e_i \mid i \in J\}$. We fix the set of input shares I_2 on b as $I_2 = [n]$, so we can perfectly simulate all probes in W_2^r and W_2^a using the full input b . Next, we need to prove that we can perfectly simulate all probes in W_1^r and J_e similarly as before. We denote $|W_1^a| = m$. We have

$$|W_1^r| \leq \min(t+1, n-t) - 1 - m \quad , \quad |J_e| \leq t + m$$

From the (t, f) -TRPE property of G_{refresh} for any $t \leq n-1$ and specifically for $t' = t + m$ with amplification order at least $d' = \min(t+1+m, n-t-m)$, and since $|W_1^r| \leq \min(t+1, n-t) - 1 - m \leq d' - 1$, then there exists an input set of shares of a I_1 such that $|I_1| \leq \min(t+m, |W_1^r|) = |W_1^r| \leq |W|$ and I_1 perfectly simulates W_1^r and J_e .

From these definitions, I_1 and I_2 together perfectly simulate W and J (J is simulated by perfectly simulating each $i \in J$ by using e_i in J_e and simulating f_i using the full input b), and we only have a failure on at most one of the inputs (b in this case). This concludes the proof for the second property.

At this point, we proved that G_{add} achieves an amplification order greater than or equal to $\min(t+1, n-t)$ for TRPE1. Since this amplification order is the maximum achievable by 2-input addition gadgets, then G_{add} achieves an amplification order exactly equal to $\min(t+1, n-t)$.

- if $|J| > t$ ((T)RPE2): we prove both properties 1 and 2:

1. we assume that $|W| < \min(t+1, n-t)$. As before, we split W as $W = W_1^r \cup W_1^a \cup W_2^r \cup W_2^a$. We consider $J' = \{i \mid e_i \in W_1^a\} \cup \{i \mid f_i \in W_2^a\}$ so we have $|J'| \leq |W_1^a| + |W_2^a|$. We also construct the set W^r which contains the set of leaking wires on the first instance of G_{refresh} (on input a) in W_1^r , and all the wires that are leaking within the second instance of G_{refresh} in W_2^r . Hence, we have that $|W^r| \leq |W_1^r \cup W_2^r| < \min(t+1, n-t)$. Hence, we have $|W^r| + |J'| \leq \min(t+1, n-t) \leq n-1$, so by Definition 9 satisfied by G_{refresh} , there exists a set of output shares indices J such that $J' \subseteq J$ and $|J| = n-1$ such that W^r and J can be perfectly simulated from a set of input shares indices I such that $|I| \leq |W^r| + |J'|$. Thus, we can fix I_1 on a and I_2 on b such that $I_1 = I_2 = I$ and we fix the set of $n-1$ output shares indices on G_{add} as the same indices in J . Hence, we can perfectly simulate all wires in W^r and J , so we can perfectly simulate all wires in W_1^r and W_2^r and W_1^a and W_2^a as well as $n-1$ output shares of G_{add} using I_1 and I_2 such that $|I_1| = |I_2| \leq |W^r| + |J'| \leq |W| = \min(t, |W|)$. That concludes the proof for the first property.
2. we now assume that $\min(t+1, n-t) \leq |W| < 2\min(t+1, n-t)$. Without loss of generality, let us consider that $|W_1^r \cup W_1^a| < \min(t+1, n-t)$ (the proof is similar in the opposite scenario).

We fix $I_2 = [n]$ on input b , which allows us to perfectly simulate all wires and output shares on G_{refresh} instance with input sharing $(b_1, \dots, b_n)$, including W_2^a and W_2^r . Next, we set $J' = \{i \mid e_i \in W_1^a\}$. Since $|W_1^r| + |J'| \leq n-1$, by Definition 9 satisfied by G_{refresh} , there exists a set of output shares indices J such that $J' \subseteq J$ and $|J| = n-1$ such that W_1^r and J can be perfectly simulated from a set of input shares indices I_1 on a such that $|I_1| \leq |W_1^r| + |J'| \leq |W_1^r| + |W_1^a| \leq |W|$. Thus, we can fix the set of $n-1$ output shares indices on G_{add} as the same indices in J . We can perfectly simulate all output shares indexed in J since for each $i \in J$, we can perfectly simulate e_i using I_1 and f_i using the full input b in I_2 , so we can perfectly simulate $e_i + f_i$. Hence, we can perfectly simulate all wires in W as well as $n-1$ output shares of G_{add} using I_1 and I_2 such that $|I_1| \leq |W_1^r| + |W_1^a| \leq \min(t, |W|)$ and with a failure on input b with $I_2 = [n]$. That concludes the proof for the second property.

We thus proved that G_{add} achieves an amplification order greater than or equal to $\min(t+1, n-t)$ for TRPE2. Since $\min(t+1, n-t)$ is the maximum order achievable for TRPE2 for a 2-input gadget, then G_{add} achieves exactly the order $\min(t+1, n-t)$.

Since G_{add} has an amplification order equal to $\min(t+1, n-t)$ for TRPE1 and TRPE2, then G_{add} is a (t, f') -TRPE addition gadget for some function f' of amplification order $\min(t+1, n-t)$, which concludes the proof. $\square$

A.7 Algorithm for the $\mathcal{O}(n \log n)$ Refresh Gadget

A.8 Proof of Lemma 3

Proof. We will prove that the gadget from Algorithm 3 is (t, f) -TRPE for any $t \leq n-1$ of amplification order $d \geq \min(t+1, n-t)$. For this, we will prove both properties TRPE1 and TRPE2.

Algorithm 3: QuasiLinearRefresh

Input : $(a_1, \dots, a_n)$ input sharing
Output: $(d_1, \dots, d_n)$ such that $d_1 + \dots + d_n = a_1 + \dots + a_n$
if $n = 1$ **then** **return** a_1 ;
if $n = 2$ **then**
 $r \leftarrow \$$;
 return $(a_1 + r, a_2 - r)$;
end
for $i \leftarrow 1$ **to** $\lfloor n/2 \rfloor$ **do**
 $r \leftarrow \$$;
 $b_i \leftarrow a_i + r$;
 $b_{\lfloor n/2 \rfloor + i} \leftarrow a_{\lfloor n/2 \rfloor + i} - r$;
end
if $n \bmod 2 = 1$ **then** $b_n \leftarrow a_n$;
 $(c_1, \dots, c_{\lfloor n/2 \rfloor}) \leftarrow \text{QuasiLinearRefresh}(b_1, \dots, b_{\lfloor n/2 \rfloor})$;
 $(c_{\lfloor n/2 \rfloor + 1}, \dots, c_n) \leftarrow \text{QuasiLinearRefresh}(b_{\lfloor n/2 \rfloor + 1}, \dots, b_n)$;
for $i \leftarrow 1$ **to** $\lfloor n/2 \rfloor$ **do**
 $r \leftarrow \$$;
 $d_i \leftarrow c_i + r$;
 $d_{\lfloor n/2 \rfloor + i} \leftarrow c_{\lfloor n/2 \rfloor + i} - r$;
end
if $n \bmod 2 = 1$ **then** $d_n \leftarrow c_n$;
return $(d_1, \dots, d_n)$;

Proof for TRPE1

The gadget is proven to be $(n - 1)$ -SNI in [5], thus it is (t, f) -TRPE1 of amplification order $d \geq \min(t + 1, n - t)$ thanks to Lemma 6 from [10]. Note that we can find failure sets of wires of size $t + 1$ which require the knowledge of $t + 1$ input shares (simply consider the leaking wires $\{a_1, \dots, a_{t+1}\}$ on input a for instance), so $d \leq t + 1$.

Proof for $(n - 1)$ -STRPE2 (which implies TRPE2)

We will first start by recalling the result of Lemma 5 in [5] which will be useful for our proof.

Lemma 5 from [5]. Let $a_1, a_2 \in \mathbb{K}$ be inputs, and let $r \xleftarrow{\$} \mathbb{K}$. Let V be a subset of the variables $\{a_1, a_2, r\}$ and $O \in \{\emptyset, \{a_1 + r\}\}$. Then the variables in $V \cup O \cup \{a_2 - r\}$ can be perfectly simulated from $I \subset \{a_1, a_2\}$, with $|I| \leq |V| + 2 \cdot |O|$.

Proof of Lemma 5 from [5]. If $|O| = 1$ or $|V| \geq 2$, we can take $I = \{a_1, a_2\}$. If $|O| = 0$ and $|V| = 0$, we can simulate $a_2 - r$ with a random value. If $|O| = 0$ and $|V| = 1$, if $V = \{a_1\}$ we let $I = \{a_1\}$ and we can again simulate $a_2 - r$ with a random value; if $V = \{r\}$ or $V = \{a_2\}$ then we let $I = \{a_2\}$. $\square$

We are now ready to prove our main result. For TRPE2, we will prove the slightly stronger property $(n - 1)$ -STRPE2. We can clearly see that $(n - 1)$ -STRPE2 implies TRPE2 of amplification order $d = t + 1$ as shown in Remark 1. We will prove $(n - 1)$ -STRPE2 by recurrence on the number of shares $n \geq 2$.

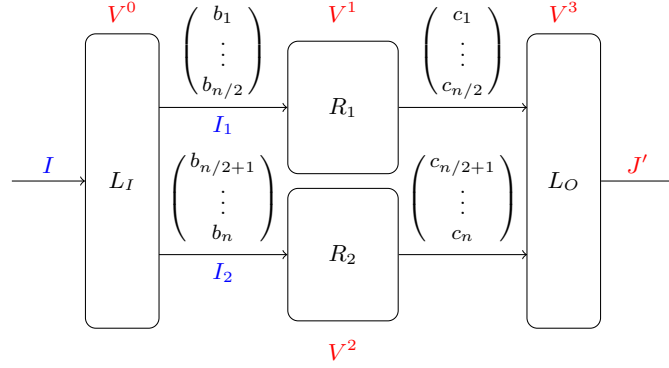


Fig. 4: $\mathcal{O}(n \log n)$ refresh gadget from [5]

The gadget in the base case ($n = 2$) gives the following output sharing:

$$\begin{aligned} d_1 &\leftarrow a_1 + r \\ d_2 &\leftarrow a_2 + r \end{aligned}$$

The proof in this case is easy. Mainly, if $J' = \emptyset$, it is easy to see that we can choose J of size 1 such that we can perfectly simulate W and J from a set of input shares I on a such that $|I| \leq |W| \leq 1$. Otherwise, if $|J'| = 1$, then $|W| = 0$, and we choose $J = J'$ and in this case we have $|I| = 0$, since we can perfectly simulate any of the output shares alone by simply generating a freshly random value. This concludes the proof for the base case.

Next we suppose that the gadget is $(n' - 1)$ -STRPE2 for any number of shares $n' < n$, and we prove the property for n shares.

To prove this, we split the gadget into four subgadgets as in Figure 4, where gadget L_I corresponds to the first loop in Algorithm 3 which adds $\lfloor n/2 \rfloor$ random values to the sharing, R_1 and R_2 gadgets correspond to the two recursive calls respectively, and L_O gadget corresponds to the second loop which also add $\lfloor n/2 \rfloor$ random values to the output sharing. We split any set of probes W on G_{refresh} into $W = V^0 \cup V^1 \cup V^2 \cup V^3$ on each of the subgadgets L_I , R_1 , R_2 and L_O respectively. The gadget R_1 is a $\lfloor \frac{n}{2} \rfloor$ -share gadget while R_2 is $\lceil \frac{n}{2} \rceil$ -share gadget. We consider that there are no probes on the output shares of R_1 and R_2 as they can be probed through V^3 . Similarly, we consider no output probes on L_I , since they can be probed through V^1 and V^2 .

Let W be the set of probes on G_{refresh} and J' be the set of output shares indices such that $|W| + |J'| \leq n - 1$. We will construct the sets J'_1 and J'_2 for output shares of the gadgets R_1 and R_2 as follows:

- for each $i \in J' \cap [\lfloor \frac{n}{2} \rfloor]$, add i to J'_1
- for each $i \in J' \cap [\lfloor \frac{n}{2} \rfloor + 1 : n]$, add i to J'_2
- for each $i \in [\lfloor \frac{n}{2} \rfloor]$ such that the input probe c_i to L_O is probed in V^3 , add i to J'_1
- for each $i \in [\lfloor \frac{n}{2} \rfloor + 1 : n]$ such that the input probe c_i to L_O is probed in V^3 , add i to J'_2

It can be seen that if we can perfectly simulate J'_1 and J'_2 , then we can perfectly simulate J' and all probes in V^3 (V^3 is composed of input probes c_i and random variables r_i , since probes of the form $c_i + r_j$ are probed in J'). Observe that we also have $|J'_1| + |J'_2| \leq |V^3| + |J'|$.

In order for the recurrence hypothesis to hold, we need the following condition to hold for the gadget R_1 :

$$|V^1| + |J'_1| \leq \left\lfloor \frac{n}{2} \right\rfloor - 1 \quad (33)$$

and the following for the gadget R_2 :

$$|V^2| + |J'_2| \leq \left\lceil \frac{n}{2} \right\rceil - 1 \quad (34)$$

We consider three cases based on the sizes of the sets of probes:

- $|V^2| + |J'_2| \geq \left\lceil \frac{n}{2} \right\rceil$. Then we must have $|V^1| + |J'_1| \leq \left\lfloor \frac{n}{2} \right\rfloor - 1$, because we have that $|W| + |J'| \leq n - 1$ and $|J'_1| + |J'_2| \leq |V^3| + |J'|$.

Since (33) holds, by the recurrence hypothesis on R_1 , we can choose a set J_1 of size $\left\lfloor \frac{n}{2} \right\rfloor - 1$ such that $J'_1 \subseteq J_1$ and we can perfectly simulate J_1 and V^1 from a set of input shares I_1 on $(b_1, \dots, b_{\lfloor n/2 \rfloor})$ such that $|I_1| \leq |V^1| + |J'_1|$. Since (34) does not hold for R_2 , we can set $J_2 = \left[\left\lceil \frac{n}{2} \right\rceil : n \right]$ and $I_2 = \left[\left\lceil \frac{n}{2} \right\rceil : n \right]$, and finally set $J = J_1 \cup J_2$ of $n - 1$ output shares on G_{refresh} . We can see that $J'_2 \subseteq J_2$ and J_2 and V^2 can be perfectly simulated from I_2 trivially (full input).

Next, we show how to perfectly simulate the sets I_1 , I_2 on intermediate variable b , and V^0 . In fact, thanks to the properties of the L_I gadget, we can apply Lemma 5 from [5] for all $1 \leq i \leq \lfloor n/2 \rfloor$ on each set of intermediate variables $\{a_i, a_{\lfloor n/2 \rfloor + i}, r_i\}$ and output variable $b_i = a_i + r_i$, where all output variables $b_{\lfloor n/2 \rfloor + i} = a_{\lfloor n/2 \rfloor + i} - r_i$ must be simulated (since we fixed $I_2 = \left[\left\lceil \frac{n}{2} \right\rceil : n \right]$), and by summing the inequalities, we construct $I \subset [n]$ on n -share input a to perfectly simulate I_1 , I_2 on intermediate variable b , and V^0 such that

$$|I| \leq |V^0| + 2|I_1| + (n \bmod 2) \leq |V^0| + 2(|V^1| + |J'_1|) + (n \bmod 2)$$

where $(n \bmod 2)$ comes from the fact that we need to perfectly simulate all shares of $(b_{\lfloor n/2 \rfloor}, \dots, b_n)$ and if $n \bmod 2 = 1$, then $b_n = a_n$ by construction of the gadget L_I .

From (33) which holds in this case, observe that we have

$$|V^1| + |J'_1| + (n \bmod 2) \leq \left\lfloor \frac{n}{2} \right\rfloor \leq \left\lceil \frac{n}{2} \right\rceil \leq |V^2| + |J'_2|,$$

then we have

$$|I| \leq |V^0| + 2(|V^1| + |J'_1|) + (n \bmod 2) \leq |V^0| + |V^1| + |J'_1| + |V^2| + |J'_2|$$

which gives

$$|I| \leq |W| + |J'|$$

and using the input shares in I , we can perfectly simulate probes in V^0 , I_1 and I_2 , and using I_1 and I_2 we proved that we can perfectly simulate probes in V^1 , V^2 , J_1 and J_2 , and so we can also perfectly simulate the chosen set of $n - 1$ output shares J and probes in V^3 . So we can perfectly simulate all internal probes plus the chosen set J of $n - 1$ output shares from I . This proves the recurrence step in this case.

– $|\mathbf{V}^1| + |\mathbf{J}'_1| \geq \left\lfloor \frac{n}{2} \right\rfloor$. Then we must have $|V^2| + |J'_2| \leq \left\lceil \frac{n}{2} \right\rceil - 1$, because we have that $|W| + |J'| \leq n - 1$ and $|J'_1| + |J'_2| \leq |V^3| + |J'|$.

Since (34) holds, by the recurrence hypothesis on R_2 , we can choose a set J_2 of size $\left\lceil \frac{n}{2} \right\rceil - 1$ such that $J'_2 \subseteq J_2$ and we can perfectly simulate J_2 and V^2 from a set of input shares I_2 on $(b_{\lceil n/2 \rceil}, \dots, b_n)$ such that $|I_2| \leq |V^2| + |J'_2|$. Since (33) does not hold for R_1 , we can set $J_1 = \left\lfloor \left\lceil \frac{n}{2} \right\rceil \right\rfloor$ and $I_1 = \left\lfloor \left\lceil \frac{n}{2} \right\rceil \right\rfloor$, and finally set $J = J_1 \cup J_2$ of $n - 1$ output shares on G_{refresh} . We can see that $J'_1 \subseteq J_1$ and J_1 and V^1 can be perfectly simulated from I_1 trivially (full input).

Next, we show how to perfectly simulate the sets I_1 , I_2 on intermediate variable b , and V^0 . In fact, thanks to the properties of the L_I gadget, we can apply Lemma 5 from [5] for all $1 \leq i \leq \lfloor n/2 \rfloor$ on each set of intermediate variables $\{a_i, a_{\lfloor n/2 \rfloor + i}, r_i\}$ and output variable $b_{\lfloor n/2 \rfloor + i} = a_{\lfloor n/2 \rfloor + i} - r_i$, where all output variables $b_i = a_i + r_i$ must be simulated (since we fixed $I_1 = \left\lfloor \left\lceil \frac{n}{2} \right\rceil \right\rfloor$), and by summing the inequalities, we construct $I \subset [n]$ on n -share input a to perfectly simulate I_1 , I_2 on intermediate variable b , and V^0 such that

$$|I| \leq |V^0| + 2|I_2| \leq |V^0| + 2(|V^2| + |J'_2|)$$

(in this case, we don't have the term $(n \bmod 2)$ anymore because we do not need the full input sharing $(b_{\lceil n/2 \rceil}, \dots, b_n)$ for the simulation as before). Since (34) holds and (33) does not hold, we observe that

$$|V^2| + |J'_2| \leq \left\lceil \frac{n}{2} \right\rceil - 1 \leq \left\lfloor \frac{n}{2} \right\rfloor \leq |V^1| + |J'_1|$$

so we get

$$|I| \leq |V^0| + 2(|V^2| + |J'_2|) \leq |V^0| + |V^2| + |J'_2| + |V^1| + |J'_1|$$

which gives

$$|I| \leq |W| + |J'|$$

and using the input shares in I , we can perfectly simulate probes in V^0 , I_1 and I_2 , and using I_1 and I_2 we proved that we can perfectly simulate probes in V^1 , V^2 , J_1 and J_2 , and so we can also perfectly simulate the chosen set of $n - 1$ output shares J and probes in V^3 . So we can perfectly simulate all internal probes plus the chosen set J of $n - 1$ output shares from I . This proves the recurrence step in this case.

– $|\mathbf{V}^1| + |\mathbf{J}'_1| \leq \left\lfloor \frac{n}{2} \right\rfloor - 1$ and $|\mathbf{V}^2| + |\mathbf{J}'_2| \leq \left\lceil \frac{n}{2} \right\rceil - 1$. This case can be treated in the exact same way as the above cases. Namely, if we have $|V^1| + |J'_1| + (n \bmod 2) \leq |V^2| + |J'_2|$, then we can consider the first case and treat it in the same way (by applying the recursion hypothesis on gadget R_1 and setting $J_2 = \left\lceil \frac{n}{2} \right\rceil : n$ and $I_2 = \left\lceil \frac{n}{2} \right\rceil : n$).

Otherwise, if we have $|V^2| + |J'_2| \leq |V^1| + |J'_1| + (n \bmod 2)$, then we can consider the second case and treat it in the same way (by applying the recursion hypothesis on gadget R_2 and setting $J_1 = \left\lfloor \frac{n}{2} \right\rfloor$ and $I_1 = \left\lfloor \frac{n}{2} \right\rfloor$).

This also concludes the proof in this case.

By treating all possible cases on the probed wires, we conclude the recursive proof. This proves that for any n shares such that $|W| + |J'| \leq n - 1$, we can choose a set J of $n - 1$ output shares such that $J' \subseteq J$ and we can perfectly simulate J and W from a set of input shares I such that

$|I| \leq |W| + |J'|$. Thus, we conclude that the gadget G_{refresh} is $(n-1)$ -STRPE2. Thus, it is also (t, f) -TRPE2 of amplification order $d = t + 1$. This concludes the proof. $\square$

A.9 Proof of Lemma 4

Proof. We will prove in this appendix Lemma 4, i.e that the constructed multiplication gadget from the composition of G_{submult} satisfying $(d-1)$ -NI and $(d-1, 2d-1)$ -partial NI, and G_{compress} satisfying (t, f') -comp-TRPE results in a (t, f) -RPE gadget G_{mult} with amplification order $d = \min(t+1, n-t)$. First let us fix $t \leq n-1$. We will be splitting a set of probe W on the multiplication gadget into two sets of probes $W = W_m \cup W_c$ where W_m are probes on G_{submult} (internal and output wires) and W_c are probes G_{compress} (on internal wires only).

We start by proving RPE1. Let J be a set of output shares such that $|J| \leq t$.

- Let W be a set of probes on the multiplication gadget such that $|W| = |W_m \cup W_c| \leq d-1$. We know in particular from the comp-TRPE property on G_{compress} that all wires in J and W_c can be simulated from a set of input shares I_c on the intermediate result c such that $|I_c| \leq |W_c|$ (since $|W_c| \leq d-1 < 2d$). Then, we have a set of probes $W'_m = W_m \cup I_c$ on G_{submult} which is of size $|W'_m| \leq |W_m| + |I_c| \leq |W_m| + |W_c| \leq d-1$, then from $(d-1)$ -NI property of G_{submult} we know that all the probes in W'_m can be simulated from sets of input shares I_a and I_b such that $|I_a| \leq d-1 \leq t$ and $|I_b| \leq d-1 \leq t$. This proves that we can simulate all probes in the overall set of probes W and in J from at most t shares of a and t shares of b . this proves the first property for RPE1.
- Next let W be a set of probes on the multiplication gadget such that $d \leq |W| = |W_m \cup W_c| \leq 2d-1$. We need to show that we can simulate W and J with at most a failure on one of the inputs a or b . We know in particular from the comp-TRPE property on G_{compress} that all wires in J and W_c can be simulated from a set of input shares I_c on the intermediate result c such that $|I_c| \leq |W_c|$ (since $|W_c| \leq 2d-1 < 2d$). Then, we have a set of probes $W'_m = W_m \cup I_c$ on G_{submult} which is of size $|W'_m| \leq |W_m| + |I_c| \leq |W_m| + |W_c| \leq 2d-1$. Hence from $(d-1, 2d-1)$ -partial NI property of G_{submult} , all the probes in W'_m can be simulated from sets of input shares I_a and I_b such that $|I_a| \leq d-1$ or $|I_b| \leq d-1 \leq t$. Since $d = \min(t+1, n-t)$, then this implies that we have a failure on at most one of the inputs.

This proves that we can simulate all probes in the overall set of probes W and in J from at most t shares of at least one of the inputs a or b (in other words, if we need more than t shares of a , then we need at most t shares of b). This proves the second property for RPE1.

From the above two cases, we conclude that the multiplication gadget is (t, f_1) -RPE1 with amplification order $d = \min(t+1, n-t)$.

Next we prove the property RPE2.

- Let W be a set of probes on the multiplication gadget such that $|W| = |W_m \cup W_c| \leq d-1$. We know in particular from the comp-TRPE property on G_{compress} that there exists a set J of $n-1$ output shares such that all wires in W_c and J can be simulated from a set of input shares I_c on the intermediate result c such that $|I_c| \leq |W_c|$ (since $|W_c| \leq d-1 < 2d$). Then, we have a set of probes $W'_m = W_m \cup I_c$ on G_{submult} which is of size $|W'_m| \leq |W_m| + |I_c| \leq |W_m| + |W_c| \leq d-1$, then from $(d-1)$ -NI property of G_{submult} we know that all the probes in W'_m can be simulated

from sets of input shares I_a and I_b such that $|I_a| \leq d - 1 \leq t$ and $|I_b| \leq d - 1 \leq t$. This proves that there exists a set J of $n - 1$ output shares such that we can simulate all probes in the overall set of probes W and in J from at most t shares of a and t shares of b . This proves the first property for RPE2.

- Next let W be a set of probes on the multiplication gadget such that $d \leq |W| = |W_m \cup W_c| \leq 2d - 1$. We know in particular from the comp-TRPE property on G_{compress} that there exists a set J of $n - 1$ output shares such that all wires in W_c and J can be simulated from a set of input shares I_c on the intermediate result c such that $|I_c| \leq |W_c|$ (since $|W_c| \leq 2d - 1 < 2d$). Then, we have a set of probes $W'_m = W_m \cup I_c$ on G_{submult} which is of size $|W'_m| \leq |W_m| + |I_c| \leq |W_m| + |W_c| \leq 2d - 1$. Hence as for RPE1, from $(d - 1, 2d - 1)$ -partial NI property of G_{submult} , we have that all the probes in W'_m can be simulated from sets of input shares I_a and I_b such that $|I_a| \leq d - 1$ or $|I_b| \leq d - 1 \leq t$. Since $d = \min(t + 1, n - t)$, then this implies that we have a failure on at most one of the inputs.

This proves that there exists a set J of $n - 1$ output shares such that we can simulate all probes in the overall set of probes W and in J from at most t shares of at least one of the inputs a or b (in other words, if we need more than t shares of a , then we need at most t shares of b). This proves the second property for RPE2.

From the above two cases, we conclude that the multiplication gadget is (t, f_2) -RPE2 with amplification order $d = \min(t + 1, n - t)$.

Combining both properties RPE1 and RPE2 with the same amplification order d , we conclude that the multiplication gadget is (t, f) -RPE with $f = \max(f_1, f_2)$ and of amplification order $d = \min(t + 1, n - t)$. This concludes the proof of lemma 4. $\square$

A.10 Proof of Lemma 5

Proof. Let G_{compress} be the $[m : n]$ -compression gadget from Algorithm 1 such that $m \geq 2n$ and let G_{refresh} be the m -share refresh gadget such that G_{refresh} is $(m - 1)$ -SNI and $(m - 1)$ -STRPE2. We will prove that G_{compress} $[m : n]$ -compression gadget constructed with such G_{refresh} is (t, f) -**comp-TRPE**. Let us denote $(c_1, \dots, c_m)$ the input shares of G_{compress} , $(d_1, \dots, d_n)$ its output shares, and $(c'_1, \dots, c'_m)$ the refreshed shares of $(c_1, \dots, c_m)$ using G_{refresh} . We write m as $m = K \cdot n + \ell$ for $K, \ell \in \mathbb{N}$ such that $K = \lfloor m/n \rfloor$. For each $1 \leq i \leq \ell$, we have $d_i = c'_i + \dots + c'_{i+K \cdot n}$, and for $\ell + 1 \leq i \leq n$, we have $d_i = c'_i + \dots + c'_{i+(K-1) \cdot n}$. We will prove that G_{compress} is (t, f) -comp-TRPE. This amounts to proving that $\forall W, |W| \leq 2d - 1$ a set of probes on the internal wires of G_{compress} where $d = \min(t + 1, n - t)$:

1. $\forall J, |J| \leq t$ a set of output shares of G_{compress} , J and W can be simulated from a set of input shares I of the input c of G_{compress} , such that $|I| \leq |W|$.
2. $\exists J', |J'| = n - 1$ a set of output shares of G_{compress} , such that J' and W can be simulated from a set of input shares I of the input c of G_{compress} , such that $|I| \leq |W|$.

We will prove both points separately

1. Let J be a set of output shares indices on G_{compress} such that $|J| \leq t$ for a $t \leq n - 1$ and let $d = \min(t + 1, n - t)$. Let W be a set of probes on G_{compress} such that $|W| \leq 2d - 1$. We need to prove that we can perfectly simulate W and J from input shares indices in I such that

$|I| \leq |W|$. For this, We will simulate W and J using probes on G_{refresh} . First let us consider J^* the set of probes such that $J^* = \{i \mid c'_i \in W \cap \{c'_1, \dots, c'_m\}\}$.

We construct the set W' of probes on G_{refresh} as follows:

$$W' = \{p \mid p \in W \setminus \{c'_1, \dots, c'_m\}\} \quad (35)$$

In addition, we construct the set J' of output shares on G_{refresh} as follows:

$$J' = J^* \cup \bigcup_{\substack{i \in J \\ i \leq \ell}} \{i, \dots, i + K.n\} \cup \bigcup_{\substack{i \in J \\ i > \ell}} \{i, \dots, i + (K-1).n\} \quad (36)$$

It is easy to see that if we can perfectly simulate W' and J' , then we can perfectly simulate W and J since $W = W' \cup \{c'_i \mid i \in J^*\}$ and by perfectly simulating $(c'_i, \dots, c'_{i+K.n})$ for $i \in J$ such that $i \leq \ell$, then we can perfectly simulate $d_i = c'_i + \dots + c'_{i+K.n}$ and by perfectly simulating $(c'_i, \dots, c'_{i+(K-1).n})$ for $i \in J$ such that $i > \ell$, then we can perfectly simulate $d_i = c'_i + \dots + c'_{i+(K-1).n}$; thus all output shares in J are perfectly simulate using shares in J' . Hence, we need to prove that we can perfectly simulate W' and J' using the G_{refresh} m -share gadget.

Observe that since $|J^*| \leq |W \setminus W'|$, then

$$|J'| \leq |W \setminus W'| + K.|J| + \min(t, \ell) \leq K.t + \min(t, \ell) \quad (37)$$

where the term $\min(t, \ell)$ comes from the worst case where all output shares $i \in J$ are such that $i \leq \ell$, because in this case we add to J' all the indices $(i, \dots, i + K.n)$ instead of $(i, \dots, i + (K-1).n)$ according to (36). Also, according to (35), we have $|W'| \leq |W|$. Hence, we have

$$|W'| + |J'| \leq |W'| + |W \setminus W'| + K.|J| + \min(t, \ell) \leq |W| + K.|J| + \min(t, \ell) \leq 2d - 1 + K.t + \min(t, \ell)$$

, so

$$|W'| + |J'| \leq 2 \min(t + 1, n - t) - 1 + K.t + \min(t, \ell)$$

, then

$$|W'| + |J'| \leq 2(n - t) + K.t + \ell - 1 \leq 2n + (K - 2).t + \ell - 1$$

, and from $t \leq n - 1$ we get

$$|W'| + |J'| \leq K.n + \ell - 1 - (K - 2).$$

Since by hypothesis we have $m \geq 2n$, so $K \geq 2$ and $(K - 2) \geq 0$, hence

$$|W'| + |J'| \leq K.n + \ell - 1 \leq m - 1$$

Then by the $(m - 1)$ -SNI property of m -share G_{refresh} , we can perfectly simulate the set of probes W' and output shares in J' from a set of input shares I such that $|I| \leq |W'|$, hence we have

$$|I| \leq |W'| \leq |W|$$

which completes the proof for the first point of comp-TRPE on gadget G_{compress} .

2. Let $t \leq n - 1$ and let $d = \min(t + 1, n - t)$. Let W be a set of probes on G_{compress} such that $|W| \leq 2d - 1$. We need to prove that we can perfectly simulate W and a chosen set J of $n - 1$ output shares from input shares indices in I such that $|I| \leq |W|$. For this, we will simulate W and choose the set J using probes on G_{refresh} . First let us consider J^* the set of probes such that $J^* = \{i \mid c'_i \in W \cap \{c'_1, \dots, c'_m\}\}$.

We construct the set W' of probes on G_{refresh} as follows:

$$W' = \{p \mid p \in W \setminus \{c'_1, \dots, c'_m\}\} \quad (38)$$

In addition, we construct the set J' of output shares on G_{refresh} as follows:

$$J' = J^* \quad (39)$$

Observe that

$$|W'| + |J'| \leq |W| \leq 2d - 1 \leq 2\min(t + 1, n - t) - 1$$

, so

$$|W'| + |J'| \leq 2n - 1 \leq m - 1$$

Then by the $(m - 1)$ -STRPE2 property of m -share G_{refresh} , there exists a set J'' such that $J' \subseteq J''$ and $|J''| = m - 1$ and W' and J'' can be perfectly simulated from input shares indexed in I such that $|I| \leq |W'| + |J'|$. Since $W = W' \cup \{c'_i \mid i \in J'\}$ then $|I| \leq |W|$.

By perfectly simulating W' and J'' , we can perfectly simulate W since $W = W' \cup \{c'_i \mid i \in J'\}$. In addition, we choose the set J of $n - 1$ output shares on G_{compress} as follows:

$$J = \{i \mid i \leq \ell \text{ and } \{i, \dots, i + K.n\} \subseteq J''\} \cup \{i \mid i > \ell \text{ and } \{i, \dots, i + (K - 1).n\} \subseteq J''\}$$

Since $|J''| = m - 1$, then we are sure that $|J| = n - 1$ since there is only 1 share of $(c'_1, \dots, c'_m)$ missing from J'' . And since we can perfectly simulate J'' then we can also perfectly simulate J like before.

This proves that we can choose a set J of $n - 1$ output shares on G_{compress} using probes on the internal gadget G_{refresh} such that W and J can be perfectly simulated from input shares in I such that $|I| \leq |W'| + |J'| \leq |W|$ for any $|W| \leq 2d - 1$. This concludes the proof for the second point of comp-TRPE on gadget G_{compress} .

Thus, we proved that G_{compress} from Algorithm 1 is (t, f) -STRPE2. This concludes the proof for Lemma 5. $\square$

A.11 Proof of Lemma 7

Proof. Let $t \leq n - 1$ where n is the number of shares such that $(n, t) \neq (2k + 1, \lfloor \frac{n-1}{2} \rfloor)$ for $k \in \mathbb{N}$ (i.e n is even or $t \neq \lfloor \frac{n-1}{2} \rfloor$), and let $d = \min(t + 1, n - t)$. We will prove that if both matrices γ and δ satisfy Condition 2, then G_{submult} from Lemma 7 is $(d - 1)$ -NI and $(d - 1, 2d - 1)$ -partial NI.

Proof for $(d - 1)$ -NI:

If the matrices γ and δ satisfy Condition 2, then they also satisfy Condition 1, since Condition 2

is stronger. Then, in [8], the authors prove that we have that if γ and δ satisfy Condition 1, then the gadget G_{submult} is $(n-1)$ -NI. In addition, if G_{submult} is $(n-1)$ -NI, then in particular it is also $(d-1)$ -NI for any $t \leq n-1$ and $d = \min(t+1, n-t)$. This implies that if the matrices satisfy Condition 2, then the gadget G_{submult} is $(d-1)$ -NI thanks to the proof from [8]. This concludes the proof for the first point of Lemma 7.

Proof for $(d-1, 2d-1)$ -partial NI:

We need to prove that G_{submult} is $(d-1, 2d-1)$ -partial NI where $d = \min(t+1, n-t)$. In other words, we need to consider a set of probes W of size $|W| \leq 2d-1 \leq n-1$ and show that W can be simulated from inputs shares I_a and I_b such that $|I_a| \leq d-1$ or $|I_b| \leq d-1$. For this, we will split the set W into 3 distinct subsets $W = W_1 \cup W_2 \cup W_3$ with respect to the form of the probes in W . In fact, The authors from [8] show that G_{submult} is $(n-1)$ -NI if the matrices γ and δ satisfy certain conditions. In fact, all of the probes on the sub-gadget G_{submult} are of a form in one of the following sets:

- Set 1:** $a_1, a_i, r_i, r_i + a_i, \gamma_{j-1, i-1} r_i, \gamma_{j-1, i-1} r_i + a_i$ (for $2 \leq i \leq n$ and $2 \leq j \leq n$)
- Set 2:** $a_1 + \sum_{i=2}^k (r_i + a_i)$ (for $2 \leq k \leq n$)
- Set 3:** $a_1 + \sum_{i=2}^k (\gamma_{j-1, i-1} r_i + a_i)$ (for $2 \leq j \leq n$ and $2 \leq k \leq n$)
- Set 4:** $b_1, b_i, s_i, s_i + b_i, \delta_{j-1, i-1} s_i, \delta_{j-1, i-1} s_i + b_i$ (for $2 \leq i \leq n$ and $2 \leq j \leq n$)
- Set 5:** $b_1 + \sum_{i=2}^k (s_i + b_i)$ (for $2 \leq k \leq n$)
- Set 6:** $b_1 + \sum_{i=2}^k (\delta_{j-1, i-1} s_i + b_i)$ (for $2 \leq j \leq n$ and $2 \leq k \leq n$)
- Set 7:** $-r_i \times (b_1 + \sum_{j=2}^n (\delta_{i-1, j-1} s_j + b_j))$ (for $2 \leq i \leq n$)
- Set 8:** $-s_i \times (a_1 + \sum_{j=2}^n (\gamma_{i-1, j-1} r_j + a_j))$ (for $2 \leq i \leq n$)
- Set 9:** $(a_1 + \sum_{i=2}^n (r_i + a_i)) \times (b_1 + \sum_{i=2}^n (s_i + b_i))$

The matrix γ would be related to probes of the form 1, 2 and 3, while the matrix δ is directly related to probes of the form 4, 5 and 6.

So we split the set W into $W = W_1 \cup W_2 \cup W_3$ with respect to the form of each probe as follows:

- W_1 contains probes of the forms in the sets 1, 2 and 3.
- W_2 contains probes of the forms in the sets 4, 5 and 6.
- W_3 contains probes of the forms in the sets 7, 8 and 9.

This split means that the set W_1 only contains probes involving the input shares of a and the randoms r_i , while W_2 only contains probes involving the input shares of b and the randoms s_i . W_3 contains products of both of the probes of W_1 and W_2 .

Next, we will construct two subsets of probes W_a and W_b from the set W and prove that we can simulate all probes in W from W_a and W_b . In other terms, we start with $W_a = W_1$ and $W_b = W_2$.

Suppose first that $W_3 = \emptyset$. Then we consider the sets $W_a = W_1$ and $W_b = W_2$ as before. Suppose that to simulate W_a , we need sets of input shares I_a such that $|I_a| \geq d$, and let M be the number of probes of the form in sets 2 and 3 in the set of probes W_a . Then from condition 2 on matrix γ we know that $|I_a| \leq |W_a| - M \leq |W_a|$ (because $|W_a| \leq 2d-1 \leq n-1$ since $t \leq n-1$ such that $((n=2k) \vee (t \neq \frac{n-1}{2}))$), then in order to have $|I_a| \geq d$, we must have:

$$d \leq |I_a| \leq |W_a|$$

Hence, since $|W| \leq 2d-1$, then we must have $|W_b| \leq d-1$ (because $|W_a| + |W_b| \leq 2d-1$), then from condition 2 on matrix δ , we can perfectly simulate W_b from I_b such that $|I_b| \leq |W_b| - M' \leq$

$|W_b| \leq d - 1$ where M' is the number of probes of the form in sets 5 and 6 in the set of probes W_b . Thus we showed that we can perfectly simulate W with $|W| \leq 2d - 1 \leq n - 1$ from W_a and W_b using I_a and I_b such that if $|I_a| \geq d$, then $|I_b| \leq d - 1$, so we have $|I_a| \leq d - 1$ or $|I_b| \leq d - 1$. This concludes the proof in the case where $W_3 = \emptyset$.

Next, we suppose that $W_3 \neq \emptyset$ so there is at least one probe of one of the sets 7, 8 or 9 in W_3 . We construct sets W_a and W_b as before starting with $W_a = W_1$ and $W_b = W_2$, and for each probe in W_3 :

- If the probe is of the form $-r_i \times (b_1 + \sum_{j=2}^n (\delta_{i-1,j-1} s_j + b_j))$, then we do $W_a = W_a \cup \{-r_i\}$, $W_b = W_b \cup \{(b_1 + \sum_{j=2}^n (\delta_{i-1,j-1} s_j + b_j))\}$. We denote the set of these probes in W_3 as W_3^7 .
- If the probe is of the form $-s_i \times (a_1 + \sum_{j=2}^n (\gamma_{i-1,j-1} r_j + a_j))$, then we do $W_a = W_a \cup \{(a_1 + \sum_{j=2}^n (\gamma_{i-1,j-1} r_j + a_j))\}$, $W_b = W_b \cup \{-s_i\}$. We denote the set of these probes in W_3 as W_3^8 .
- if the probe is of the form $(a_1 + \sum_{i=2}^n (r_i + a_i)) \times (b_1 + \sum_{i=2}^n (s_i + b_i))$, then we do $W_a = W_a \cup \{(a_1 + \sum_{i=2}^n (r_i + a_i))\}$, $W_b = W_b \cup \{(b_1 + \sum_{i=2}^n (s_i + b_i))\}$. We denote the set of these probes in W_3 as W_3^9 .

Suppose that in order to simulate W_a , we need the set I_a such that $|I_a| \geq d$. In addition, since $|W_a| \leq |W| \leq 2d - 1 \leq n - 1$ (because $t \leq n - 1$ such that $((n = 2k) \vee (t \neq \frac{n-1}{2}))$), then we know from condition 2 on γ that W_a can be perfectly simulated from I_a such that $|I_a| \leq |W_a| - M$ where M is the number of probes in W_a of the form $(a_1 + \sum_{j=2}^n (\gamma_{i-1,j-1} r_j + a_j))$ or $(a_1 + \sum_{i=2}^n (r_i + a_i))$. Then, since probes in the sets W_3^8 and W_3^9 add to W_a probes of these forms, then we have $|I_a| \leq |W_a| - |W_3^8| - |W_3^9|$. Hence, in order to have $|I_a| \geq d$, we must have

$$d \leq |I_a| \leq |W_a| - |W_3^8| - |W_3^9| \leq |W_1| + |W_3^7|$$

Similarly, suppose that to simulate W_b we need $|I_b| \geq d$, then we also must have

$$d \leq |I_b| \leq |W_b| - |W_3^7| - |W_3^9| \leq |W_2| + |W_3^8|$$

Hence, in order to have $|I_a| \geq d$ and $|I_b| \geq d$ at the same time, we must have

$$2d \leq |I_a| + |I_b| \leq |W_1| + |W_3^7| + |W_2| + |W_3^8| \leq |W|$$

which holds a contradiction with the fact that $|W| \leq 2d - 1$. Hence, we cannot have at the same time $|I_a| \geq d$ and $|I_b| \geq d$. So G_{submult} is $(d - 1, 2d - 1)$ -partial NI in the case where $W_3 \neq \emptyset$.

Hence, we conclude that G_{submult} is $(d - 1, 2d - 1)$ -partial NI after proving the property in both cases $W_3 = \emptyset$ and $W_3 \neq \emptyset$.

We conclude that G_{submult} satisfies both $(d - 1)$ -NI and $(d - 1, 2d - 1)$ -partial NI, which concludes the proof for Lemma 7. $\square$

A.12 Proof of Proposition 1

We consider the following matrices

$$\begin{aligned} \mathbf{L} &= [\mathbf{I}_n | \mathbf{0}_{n \times n} | \mathbf{I}_n | \mathbf{I}_n | \dots | \mathbf{I}_n | \mathbf{T}_n | \mathbf{T}_n | \dots | \mathbf{T}_n] \\ \mathbf{M} &= [\mathbf{0}_{n \times n} | \mathbf{I}_n | \mathbf{I}_n | \mathbf{D}_{\gamma,1} | \dots | \mathbf{D}_{\gamma,n} | \mathbf{T}_n | \mathbf{T}_{\gamma,1} | \dots | \mathbf{T}_{\gamma,n}] \end{aligned}$$

The matrices $\mathbf{L}$ and $\mathbf{M}$ have $z = (2n+4) \cdot n$ columns. We want to lower-bound the probability, for γ picked uniformly at random in $\mathbb{F}_q^{n \times n}$, that for any vector $v \in \mathbb{F}_q^z$ of Hamming weight $hw(v) \leq n$, and for any $i_1, \dots, i_K \in [z]$ such that $v_{i_1} \neq 0, \dots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \dots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient (*i.e.* there are K probes of the form $\sum_{i=1}^n (r_i + a_i)$ or $\sum_{j=1}^n (\gamma_{i,j} r_j + a_j)$ for any $i = 1, \dots, n$), if $\mathbf{M} \cdot v = 0$, then we have $hw(\mathbf{L} \cdot v) \leq hw(v) - K$.

For any set $I \subseteq \{1, \dots, z\}$, we denote by $\mathbf{L}_I$ the $n \times |I|$ submatrix of $\mathbf{L}$ obtained by only keeping the columns in $\mathbf{L}$ whose indices are in I and $\mathbf{M}_I$ is the $n \times |I|$ submatrix of $\mathbf{M}$ obtained by only keeping the columns in $\mathbf{M}$ whose indices are in I . We will lower-bound the probability that for any set $I \subseteq \{1, \dots, z\}$ of cardinal n and any vector $v \in \mathbb{F}_q^n$, if $hw(\mathbf{L}_I \cdot v) \geq hw(v) - K + 1$ then $\mathbf{M}_I \cdot v \neq \mathbf{0}_n$.

We consider different cases (in order of increasing generality) which depend on the columns selected with the set I :

1. $I \subseteq \{(n+4) \cdot n + 1, \dots, z\}$, *i.e.*, all columns in $\mathbf{M}_I$ are taken from the matrices $\mathbf{T}_{\gamma,i}$ for $i \in \{1, \dots, n\}$;
2. $I \subseteq \{(n+3) \cdot n + 1, \dots, z\}$, *i.e.*, all columns in $\mathbf{M}_I$ are taken from the matrix $\mathbf{T}_n$ or the matrices $\mathbf{T}_{\gamma,i}$ for $i \in \{1, \dots, n\}$;
3. $I \subseteq \{1, \dots, n+1\} \cup \{(n+3) \cdot n + 1, \dots, z\}$, *i.e.*, all columns in $\mathbf{M}_I$ are taken from the null vectors, from the matrix $\mathbf{T}_n$ or the matrices $\mathbf{T}_{\gamma,i}$ for $i \in \{1, \dots, n\}$;
4. $I \subseteq \{1, \dots, z\}$, *i.e.*, the columns in $\mathbf{M}_I$ can be taken arbitrarily.

Case 1. In order to analyze the probability in the first case, we recall the definition of a probability distribution on structured matrices introduced in [8]. In this distribution of structured matrices, a number of elements with known location are identically zero, and remaining elements are chosen uniformly at random independently of each other.

Definition 13. Let n and m be two positive integers. Let $\alpha = (\alpha_1, \dots, \alpha_m)$ be a non-decreasing finite sequence with $1 \leq \alpha_1 \leq \alpha_2 \leq \dots \leq \alpha_m \leq n$.

- A matrix $\Theta = (\theta_{i,j}) \in \mathbb{F}_q^{n \times m}$ is called a *progressive patterned matrix* with pattern α if $\theta_{i,j} = 0$ for all $j \in \{1, \dots, m\}$ and all $i \notin \{\alpha_{j-1} + 1, \dots, \alpha_j\}$ (where $\alpha_0 = 0$).
- The unitary progressive patterned matrix $\Upsilon_\alpha = (u_{i,j}) \in \mathbb{F}_q^{n \times m}$ with pattern α is defined by $u_{i,j} = 0$ for all $j \in \{1, \dots, m\}$ and all $i \notin \{\alpha_{j-1} + 1, \dots, \alpha_j\}$ and $u_{i,j} = 1$ for all $j \in \{1, \dots, m\}$ and all $i \in \{\alpha_{j-1} + 1, \dots, \alpha_j\}$.
- The distribution $\mathcal{D}_\alpha$ is the probability distribution on random progressive patterned matrix $\mathbf{S}_\alpha = (s_{i,j}) \in \mathbb{F}_q^{n \times m}$ whose elements $s_{i,j}$ for $(i, j) \in \{1, \dots, n\} \times \{1, \dots, m\}$ are sampled uniformly at random and independently according to:

$$\Pr[s_{i,j} = s] = \begin{cases} 1 & \text{if } s = 0 \text{ and } u_{i,j} = 0 \\ 0 & \text{if } s \neq 0 \text{ and } u_{i,j} = 0 \\ q^{-1} & \text{for all } s \in \mathbb{F}_q \text{ if } u_{i,j} = 1 \end{cases}$$

where $\Upsilon_\alpha = (u_{i,j}) \in \mathbb{F}_q^{n \times m}$ is the unitary progressive patterned matrix with pattern α .

A matrix Θ is thus a *progressive patterned matrix* with pattern $\alpha = (\alpha_1, \dots, \alpha_m)$ if it is of the form described in Figure 5 where the symbol $\star$ denotes an arbitrary value in $\mathbb{F}_q$. For the unitary progressive patterned matrix Υ_α , this symbol $\star$ is replaced by a 1 and for a random progressive

patterned matrix $\mathbf{S}_\alpha$ each symbol $\star$ is replaced by a value picked uniformly and independently at random in $\mathbb{F}_q$. Note that such a matrix can contain a null column (when $\alpha_i = \alpha_{i+1}$ for some $i \in \{1, \dots, m-1\}$).

$$\begin{array}{c}
 \overbrace{\hspace{10em}}^{m \text{ columns}} \\
 \left(\begin{array}{cccc|ccc}
 \star & 0 & 0 & 0 & \cdots & & 0 & 0 \\
 \star & 0 & 0 & 0 & \cdots & & 0 & 0 \\
 \star & 0 & 0 & 0 & \cdots & & 0 & 0 \\
 \hline
 0 & \star & 0 & 0 & \cdots & & 0 & 0 \\
 \alpha_2 & 0 & \star & 0 & \cdots & & 0 & 0 \\
 \alpha_3 & 0 & 0 & \star & \cdots & & 0 & 0 \\
 \hline
 0 & 0 & 0 & \star & \cdots & & 0 & 0 \\
 \alpha_4 & 0 & 0 & 0 & \cdots & & 0 & 0 \\
 \hline
 0 & 0 & 0 & 0 & \cdots & & 0 & 0 \\
 \vdots & \vdots & \vdots & \vdots & & & \vdots & \vdots \\
 \vdots & \vdots & \vdots & \vdots & & & \vdots & \vdots \\
 \alpha_{m-1} & 0 & 0 & 0 & \cdots & & \star & 0 \\
 \hline
 0 & 0 & 0 & 0 & \cdots & & 0 & \star \\
 \alpha_m & 0 & 0 & 0 & \cdots & & 0 & \star \\
 \hline
 0 & 0 & 0 & 0 & \cdots & & 0 & 0 \\
 \vdots & \vdots & \vdots & \vdots & & & \vdots & \vdots \\
 \vdots & \vdots & \vdots & \vdots & & & \vdots & \vdots \\
 0 & 0 & 0 & 0 & \cdots & & 0 & 0
 \end{array} \right)
 \end{array}
 \left. \vphantom{\begin{array}{c} \star \\ \vdots \\ 0 \end{array}} \right\} n \text{ rows}$$

Fig. 5: Form of a progressive patterned matrix with pattern $\alpha = (\alpha_1, \dots, \alpha_m)$

Belaïd *et al.* [8] also defined more generally block column matrices formed of progressive patterned matrices.

Definition 14. Let n, m, t be three positive integers. Let $m_1, \dots, m_t$ be positive integers such that $m_1 + \dots + m_t = m$ and let $\alpha^{(i)} = (\alpha_1^{(i)}, \dots, \alpha_{m_i}^{(i)})$ be a non-decreasing finite sequence with $1 \leq \alpha_1^{(i)} \leq \alpha_2^{(i)} \leq \dots \leq \alpha_{m_i}^{(i)} \leq n$ for all $i \in \{1, \dots, t\}$. We suppose that there exists at least one $j \in \{1, \dots, t\}$ such that $\alpha_{m_j}^{(j)} = n$.

- A matrix $\Theta \in \mathbb{F}_q^{n \times m}$ is called a block progressive patterned matrix with pattern $(\alpha^{(1)}, \dots, \alpha^{(t)})$ if there exist progressive patterned matrices $\Theta^{(i)} \in \mathbb{F}_q^{n \times m_i}$ with pattern $\alpha^{(i)}$ for all $i \in \{1, \dots, t\}$ such that $\Theta = (\Theta^{(1)} | \dots | \Theta^{(t)})$.
- The block unitary progressive patterned matrix $\Upsilon_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ with pattern $(\alpha^{(1)}, \dots, \alpha^{(t)})$ is $\Upsilon_{\alpha^{(1)}, \dots, \alpha^{(t)}} = (\Upsilon_{\alpha^{(1)}} | \dots | \Upsilon_{\alpha^{(t)}})$.
- The distribution $\mathcal{D}_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ is the probability distribution on block random progressive patterned matrix in $\mathbb{F}_q^{n \times m}$ defined by

$$\mathcal{D}_{\alpha^{(1)}, \dots, \alpha^{(t)}} = (\mathcal{D}_{\alpha^{(1)}} | \dots | \mathcal{D}_{\alpha^{(t)}}).$$

The main ingredient of the proof of Proposition 1 is the following technical lemma:

Lemma 11. Let n, m, t be three positive integers with $m \geq n$ and let $\alpha^{(i)}$ for $i \in \{1, \dots, t\}$ be patterns for block progressive patterned matrix as in Definition 14. For a block random progressive patterned matrix $\mathbf{S}$ drawn following the distribution $\mathcal{D}_{\alpha^{(1)}, \dots, \alpha^{(t)}}$, there exists a linear subspace of

$\mathbb{F}_q^m$ of dimension $m - n$ that contains $\{\mathbf{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = \mathbf{0}\}$, with probability at least $1 - mq^{-1}$.

Proof (Lemma 11). We will prove this lemma by induction on m .

For $m = 1$, since $m \geq n \geq 1$, Definition 14 implies that the matrix $\mathbf{S}$ consists simply in a single entry $s_{1,1}$ which is picked uniformly at random in $\mathbb{F}_q$ and this entry is null with probability q^{-1} . The set $\{\mathbf{v} \in \mathbb{F}_q \text{ s.t. } hw(\mathbf{v}) = 1 \text{ and } \mathbf{S} \cdot \mathbf{v} = \mathbf{0}\}$ is therefore the empty set with probability at least $1 - q^{-1}$ and it is thus included in the subspace of dimension 0 with probability at least $1 - q^{-1}$.

We now consider $m \geq 2$ and we suppose Lemma 11 proven for all block random progressive patterned matrix with strictly less than m columns.

We first assume that the matrix $\Upsilon_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ is the matrix of ones $\mathbf{U}_{n \times m}$ (i.e., does not contain any zero). Then $\mathbf{S}$ is simply a matrix drawn from $\mathbb{F}_q^{n \times m}$ with the uniform distribution.

It is well known that the number of full-rank $n \times m$ matrices over $\mathbb{F}_q$ (with $m \geq n$) is:

$$(q^m - 1)(q^m - q) \cdots (q^m - q^{n-1})$$

and the probability that $\mathbf{S}$ is of full rank is thus equal to:

$$(1 - q^{-m})(1 - q^{-m+1}) \cdots (1 - q^{-m+n-1})$$

which is greater than

$$1 - \sum_{i=m-n+1}^m q^{-i} \geq 1 - \sum_{i=m-n+1}^{\infty} q^{-i} = 1 - \frac{1}{q^{-m+n-1}(1 - 1/q)} \geq 1 - 2q^{n-m-1}.$$

The subspace $\{\mathbf{v} \in \mathbb{F}_q^m \text{ s.t. } \mathbf{S} \cdot \mathbf{v} = \mathbf{0}\}$ is therefore included in a linear subspace of dimension $m - n$ with probability at least $1 - 2q^{n-m-1}$ and the result follows (since $m \geq 2$).

We now assume that the matrix $\Upsilon_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ contains some 0. By assumption, there exists some $j \in \{1, \dots, t\}$ such that $\alpha_{m_j}^{(j)} = n$.

1. We first assume that $m_j > 1$ (i.e. that the column of index $m_1 + \dots + m_j$ consists in $\alpha_{m_j-1}^{(j)} \geq 1$ zeroes followed by $\alpha_{m_j}^{(j)} - \alpha_{m_j-1}^{(j)} = n - \alpha_{m_j-1}^{(j)} \geq 1$ ones, see Figure 6). We consider the submatrix of $\Upsilon_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ obtained by deleting the column of index $m_1 + \dots + m_j$ and the rows of indices in the set $\{\alpha_{m_j-1}^{(j)} + 1, \dots, \alpha_{m_j}^{(j)}\}$.

It is easy to see that this submatrix is a block unitary progressive patterned matrix with $n' \leq n - 1$ rows and $m - 1$ columns, where some columns may possibly contain only zeroes (see Figure 6). We can thus apply the induction hypothesis to the submatrix $\mathbf{S}'$ of $\mathbf{S}$ obtained by deleting the same column and the same rows.

By induction hypothesis, we know that with probability at least $1 - (m - 1)q^{-1}$, there exists a linear subspace $V' \subseteq \mathbb{F}_q^{m-1}$ of dimension $m - 1 - n'$ that contains the set $\{\mathbf{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\mathbf{v}) = m - 1 \text{ and } \mathbf{S}' \cdot \mathbf{v} = \mathbf{0}\}$.

If V' is of dimension 0, then $\{\mathbf{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\mathbf{v}) = m - 1 \text{ and } \mathbf{S}' \cdot \mathbf{v} = \mathbf{0}\} \subseteq \{\mathbf{0}_{m-1}\}$ and this set is thus the empty set. We then have $\{\mathbf{v} \in \mathbb{F}_q^m, hw(\mathbf{v}) = m \text{ and } \mathbf{S} \cdot \mathbf{v} = \mathbf{0}\} = \emptyset$ with probability at least $1 - (m - 1)q^{-1} \geq 1 - mq^{-1}$, and so there exists a linear subspace V of dimension $m - n$ that contains this set.

$$\begin{array}{cccc}
\overbrace{}^{m_1} & \overbrace{}^{m_2} & \overbrace{}^{m_j} & \overbrace{}^{m_t} \\
\left(\begin{array}{cccccccccccc}
1 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 1 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 1 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 1 & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 1 & 0 & \cdots & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & \cdots & 0 & 0 & 0 & 0
\end{array} \right)
\end{array}$$

Fig. 6: Example of a matrix $\Upsilon_{\alpha(1), \dots, \alpha(t)} \in \mathbb{F}_q^{n \times m}$. The column and the rows highlighted in red are deleted in order to apply the induction hypothesis.

If V' is of dimension $m - 1 - n' > 0$, we can assume without loss of generality that the column of $\mathbf{S}$ deleted to obtain $\mathbf{S}'$ was the last one (by permuting the blocks of the matrix). We have the following block-decomposition of $\mathbf{S}$

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}' & \mathbf{0}_{n' \times 1} \\ \mathbf{S}'' & \mathbf{u} \end{pmatrix}$$

where $\mathbf{S}''$ is a $(n - n') \times (m - 1)$ matrix and $\mathbf{u}$ a column vector of dimension $(n - n')$. Note that $\mathbf{u}$ is a random vector in $\mathbb{F}_q^{n - n'}$ independent from $\mathbf{S}'$ and $\mathbf{S}''$. Let $\mathbf{v} \in \mathbb{F}_q^m$ such that $hw(\mathbf{v}) = m$ and $\mathbf{S}\mathbf{v} = 0$.

We write $\mathbf{v} = \begin{pmatrix} \mathbf{w} \\ \tau \end{pmatrix}$ where $\mathbf{w} \in \mathbb{F}_q^{m-1}$ and $\tau \in \mathbb{F}_q$ is a scalar. We have $hw(\mathbf{w}) = m - 1$ and $\mathbf{S}'\mathbf{w} = 0$, and therefore $\mathbf{w} \in V'$. Since $\tau \neq 0$ by assumption, the vector $\mathbf{u}$ thus belongs to the image W of V' by $\mathbf{S}''$ (with probability at least $1 - (m - 1)q^{-1}$). Moreover, W has dimension at most $\max(m - 1 - n', n - n')$.

- If W is of dimension at most $n - n' - 1$, since $\mathbf{u}$ is independent of $\mathbf{S}'$ and $\mathbf{S}''$ (and thus of W), $\mathbf{u}$ belongs to W with probability at most q^{-1} . Therefore, with probability at least $(1 - q^{-1}) \cdot (1 - (m - 1)q^{-1}) \geq 1 - mq^{-1}$, $\{\mathbf{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\} = \emptyset$.
- If W is of dimension $n - n'$, with probability $1 - q^{-(n - n')} \geq 1 - q^{-1}$, we have $\mathbf{u} \neq \mathbf{0}_{(n - n') \times 1}$ and we can construct a basis $\mathbf{u}_1 = \mathbf{u}, \dots, \mathbf{u}_{n - n'}$ of W .

All subspaces $V' \cap \mathbf{S}''^{-1}(\langle \mathbf{u}_i \rangle)$ are of dimension at least one and we have

$$V' = \bigoplus_{i=1}^{n - n'} V' \cap \mathbf{S}''^{-1}(\langle \mathbf{u}_i \rangle).$$

Therefore the linear subspace V defined as $V = V' \cap \mathbf{S}''^{-1}(\langle \mathbf{u}_1 \rangle)$ satisfies

$$\begin{aligned} \dim(V) &= \dim(V') - \sum_{i=2}^{n-n'} \dim(V' \cap \mathbf{S}''^{-1}(\langle \mathbf{u}_i \rangle)) \\ &\leq m - 1 - n' - (n - n' - 1) \\ &= m - n. \end{aligned}$$

Moreover, we have $\{\mathbf{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\} \subseteq V$ and since this occurs with probability at least $(1 - q^{-1})(1 - (m - 1)q^{-1}) \geq 1 - mq^{-1}$, the result follows.

2. We now assume that $m_i = 1$ for all i such that $\alpha_{m_i}^{(i)} = n$ (i.e. that all the columns with a one in the last row consists only of ones, see Figure 7). Since the matrix $\Upsilon_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ contains some 0, there exists some $j \in \{1, \dots, t\}$ such that $m_j > 1$ and we consider such a $j \in \{1, \dots, t\}$ for which $\alpha_1^{(j)}$ is minimal (see Figure 7).

We consider the submatrix of $\Upsilon_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ obtained by deleting the column of index $m_1 + \dots + m_{j-1} + 1$ and the rows of indices in the set $\{1, \dots, \alpha_1^{(j)} - 1\}$. It is easy to see that this submatrix is a block unitary progressive patterned matrix with $n' \leq n - 1$ rows and $m - 1$ columns (see Figure 7). We can thus apply the induction hypothesis to the submatrix $\mathbf{S}'$ of $\mathbf{S}$ obtained by deleting the same column and the same rows.

$$\begin{array}{c} \begin{array}{ccccccccc} m_1 & m_2 & m_3 & & m_j & & m_t \end{array} \\ \left(\begin{array}{cccccccccccccccc} 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 1 & 0 & 0 & 0 & \cdots & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & \cdots & 0 & 1 & 0 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 1 & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & \cdots & 0 & 0 \end{array} \right) \end{array}$$

Fig. 7: Example of a matrix $\Upsilon_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$. The column and the rows highlighted in red are deleted in order to apply the induction hypothesis.

We know that with probability at least $1 - (m - 1)q^{-1}$, there exists a linear subspace $V' \subseteq \mathbb{F}_q^{m-1}$ of dimension $m - 1 - n'$ that contains the set $\{\mathbf{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\mathbf{v}) = m - 1 \text{ and } \mathbf{S}'\mathbf{v} = 0\}$.

If V' is of dimension 0, then $\{\mathbf{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\mathbf{v}) = m - 1 \text{ and } \mathbf{S}'\mathbf{v} = 0\} \subseteq \{0\}$ and this set is thus the empty set. We then have $\{\mathbf{v} \in \mathbb{F}_q^m, hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\} = \emptyset$ with probability at least $1 - (m - 1)q^{-1} \geq 1 - mq^{-1}$, and so there exists a linear subspace V of dimension $m - n$ that contains this set.

If V' is of dimension $m - 1 - n' > 0$, we can assume without loss of generality that the column of $\mathbf{S}$ deleted to obtain $\mathbf{S}'$ was the last one (by permuting the blocks of the matrix). We have the following block-decomposition of $\mathbf{S}$

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}'' & \mathbf{u} \\ \mathbf{S}' & \mathbf{0}_{n' \times 1} \end{pmatrix}$$

where $\mathbf{S}''$ is a $(n - n') \times (m - 1)$ matrix and $\mathbf{u}$ a column vector of dimension $(n - n')$. Note that $\mathbf{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$ and $\mathbf{S}''$. Let $\mathbf{v} \in \mathbb{F}_q^m$ such that $hw(\mathbf{v}) = m$ and $\mathbf{S}\mathbf{v} = 0$.

We write $\mathbf{v} = \begin{pmatrix} \tau \\ \mathbf{w} \end{pmatrix}$ where $\mathbf{w} \in \mathbb{F}_q^{m-1}$ and $\tau \in \mathbb{F}_q$ is a scalar. We have $hw(\mathbf{w}) = m - 1$ and $\mathbf{S}'\mathbf{w} = 0$, and therefore $\mathbf{w} \in V'$. Since $\tau \neq 0$ by assumption, the vector $\mathbf{u}$ thus belongs to the image W of V' by $\mathbf{S}''$ (with probability at least $1 - (m - 1)q^{-1}$). Moreover, W has dimension at most $\max(m - 1 - n', n - n')$.

- If W is of dimension at most $n - n' - 1$, since $\mathbf{u}$ is independent of $\mathbf{S}'$ and $\mathbf{S}''$ (and thus of W), $\mathbf{u}$ belongs to W with probability at most q^{-1} . Therefore, with probability at least $(1 - q^{-1}) \cdot (1 - (m - 1)q^{-1}) \geq 1 - mq^{-1}$, $\{\mathbf{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\} = \emptyset$.
- If W is of dimension $n - n'$ then $\mathbf{S}''$ is invertible. With probability $1 - q^{-(n-n')} \geq 1 - q^{-1}$, we have $\mathbf{u} \neq \mathbf{0}_{(n-n') \times 1}$ and we can construct a basis $\mathbf{u}_1 = \mathbf{u}, \dots, \mathbf{u}_{n-n'}$ of W .

All subspaces $V' \cap \mathbf{S}''^{-1}(\langle \mathbf{u}_i \rangle)$ are of dimension at least one and we have

$$V' = \bigoplus_{i=1}^{n-n'} V' \cap \mathbf{S}''^{-1}(\langle \mathbf{u}_i \rangle).$$

Therefore the linear subspace V defined as $V = V' \cap \mathbf{S}''^{-1}(\langle \mathbf{u}_1 \rangle)$ satisfies

$$\begin{aligned} \dim(V) &= \dim(V') - \sum_{i=2}^{n-n'} \dim(V' \cap \mathbf{S}''^{-1}(\langle \mathbf{u}_i \rangle)) \\ &\leq m - 1 - n' - (n - n' - 1) \\ &= m - n. \end{aligned}$$

Moreover, we have $\{\mathbf{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\} \subseteq V$ and since this occurs with probability at least $(1 - q^{-1})(1 - (m - 1)q^{-1}) \geq 1 - mq^{-1}$, the result follows.

This concludes the proof of Lemma 11.

Recall that we want to lower-bound the probability over the $\gamma \in \mathbb{F}_q^{n \times n}$, that for a given set $I \subseteq \{(n + 4) \cdot n + 1, \dots, z\}$ of cardinal n , if $hw(\mathbf{L}_I \cdot \mathbf{v}) \geq n - K$ then $\mathbf{M}_I \cdot \mathbf{v} \neq \mathbf{0}_n$ for any vector $\mathbf{v} \in \mathbb{F}_q^n$. where K denotes the number of coordinates $i_1, \dots, i_K \in [z]$ such that $v_{i_1} \neq 0, \dots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \dots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient.

Remark that the non-zero coefficients in the lower block of $\mathbf{L}_I$ and in $\mathbf{M}_I$ are at the same positions. If $K = 0$, then the matrices $\mathbf{M}_I$ and $\mathbf{L}_I$ have a null row. In this case, we have readily $hw(\mathbf{L}_I \cdot \mathbf{v}) \leq n - 1 = n - K - 1 < n - K$.

If $K \geq 1$, then the matrices $\mathbf{M}_I$ and $\mathbf{L}_I$ does not have a null row. The matrix $\mathbf{M}_I$ (up to some permutation of its columns) can be written as a block matrix where each block is of the form described in Figure 8 (on the left).

$$\begin{pmatrix}
\gamma_{i,1} & \gamma_{i,1} & \dots & \gamma_{i,1} & \dots & \gamma_{i,1} \\
\vdots & \vdots & & \vdots & & \vdots \\
\gamma_{i,\alpha_1} & \gamma_{i,\alpha_1} & \dots & \gamma_{i,\alpha_1} & \dots & \gamma_{i,\alpha_1} \\
0 & \gamma_{i,\alpha_1+1} & \dots & \gamma_{i,\alpha_1+1} & \dots & \gamma_{i,\alpha_1+1} \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \gamma_{i,\alpha_2} & \dots & \gamma_{i,\alpha_2} & \dots & \gamma_{i,\alpha_2} \\
0 & 0 & \dots & \gamma_{i,\alpha_2+1} & \dots & \gamma_{i,\alpha_2+1} \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \dots & 0 & \gamma_{i,\alpha_j} & \dots & \gamma_{i,\alpha_j} \\
0 & \dots & 0 & 0 & \dots & \gamma_{i,\alpha_j+1} \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \dots & \dots & 0 & \dots & \gamma_{i,\alpha_{m-1}} \\
0 & \dots & \dots & 0 & \dots & \gamma_{i,\alpha_{m-1}+1} \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \dots & \dots & 0 & \dots & \gamma_{i,\alpha_m} \\
0 & \dots & \dots & 0 & \dots & 0 \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \dots & \dots & \dots & \dots & 0
\end{pmatrix}
\quad
\begin{pmatrix}
\gamma_{i,1} & 0 & \dots & \dots & \dots & 0 \\
\vdots & \vdots & & \vdots & & \vdots \\
\gamma_{i,\alpha_1} & 0 & \dots & \dots & \dots & 0 \\
0 & \gamma_{i,\alpha_1+1} & \dots & \dots & \dots & 0 \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \gamma_{i,\alpha_2} & \dots & \dots & \dots & 0 \\
0 & 0 & \dots & \gamma_{i,\alpha_j-1+1} & \dots & 0 \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \dots & 0 & \gamma_{i,\alpha_j} & \dots & 0 \\
0 & \dots & 0 & 0 & \dots & 0 \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \dots & \dots & 0 & \dots & 0 \\
0 & \dots & \dots & 0 & \dots & \gamma_{i,\alpha_{m-1}+1} \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \dots & \dots & 0 & \dots & \gamma_{i,\alpha_m} \\
0 & \dots & \dots & 0 & \dots & 0 \\
\vdots & \vdots & & \vdots & & \vdots \\
0 & \dots & \dots & \dots & \dots & 0
\end{pmatrix}$$

Fig. 8: Blocks appearing in matrices $\mathbf{M}_I$ and $\tilde{\mathbf{M}}_I$

From this matrix, one can construct another matrix $\tilde{\mathbf{M}}_I$ such that in each block, one subtract each column to the following columns (i.e., one subtract iteratively the i -th column to the columns of index in $\{i+1, \dots, m\}$ for $i \in \{1, \dots, m\}$). The blocks appearing in the matrix $\tilde{\mathbf{M}}_I$ are given in Figure 8 (on the right). Since we apply only elementary operations on the columns, if there exists a vector $\mathbf{v} \in \mathbb{F}_q^n$ such that $\mathbf{M}_I \mathbf{v} = 0$ then, there exists a vector $\mathbf{v}' \in \mathbb{F}_q^n$ such that $\tilde{\mathbf{M}}_I \mathbf{v}' = 0$.

Since $\mathbf{M}_I$ has no null row, we have $\alpha_m = n$ in one of this block (with the notation from Figure 8) and the matrix $\tilde{\mathbf{M}}_I$ is thus a block random progressive patterned matrix as defined in Definition 14. By Lemma 11, for each non-empty subset J of the n columns of $\tilde{\mathbf{M}}_I$, the probability over γ that there exists a vector $\mathbf{v}' \in \mathbb{F}_q^n$ with support J (i.e., set of non-zero coordinates) such that $\tilde{\mathbf{M}}_I \mathbf{v}' = 0$ is upper bounded by $n \cdot q^{-1}$. By the union bound over all supports, the probability over γ that there exists a vector $\mathbf{v}' \in \mathbb{F}_q^n$ such that $\tilde{\mathbf{M}}_I \mathbf{v}' = 0$ is thus upper-bounded by $2^n \cdot n \cdot q^{-1}$.

For the sets $I \subseteq \{(n+4) \cdot n + 1, \dots, z\}$ of cardinal n , we have proved that with probability at least $1 - 2^n \cdot n \cdot q^{-1}$ (over the choice of $\gamma \in \mathbb{F}_q^{n \times n}$), we have $hw(\mathbf{L}_I \cdot \mathbf{v}) < n - K$ or $\mathbf{M}_I \cdot \mathbf{v} \neq \mathbf{0}_n$ for any vector $\mathbf{v} \in \mathbb{F}_q^n$.

Case 2. We now consider matrices $\mathbf{M}_I$ where all columns are taken from the matrix $\mathbf{T}_n$ or the matrices $\mathbf{T}_{\gamma,i}$ for $i \in \{1, \dots, n\}$ (i.e., $I \subseteq \{(n+3) \cdot n + 1, \dots, z\}$). With the notation from Definition 14, we consider the modified distribution $\tilde{\mathcal{D}}_{\alpha^{(1)}, \dots, \alpha^{(t)}}$ defined as the following probability distribution in $\mathbb{F}_q^{n \times m}$:

$$\tilde{\mathcal{D}}_{\alpha^{(1)}, \dots, \alpha^{(t)}} = (\Upsilon_{\alpha^{(1)}} | \mathcal{D}_{\alpha^{(2)}, \dots, \alpha^{(t)}}) = (\Upsilon_{\alpha^{(1)}} | \mathcal{D}_{\alpha^{(2)}} | \dots | \mathcal{D}_{\alpha^{(t)}})$$

(i.e., in which the first block is a fixed unitary progressive patterned matrix instead of being a random progressive patterned matrix). We can easily extend Lemma 11 to this distribution:

Lemma 12. *Let n, m, t be three positive integers with $m \geq n$ and let $\alpha^{(i)}$ for $i \in \{1, \dots, t\}$ be patterns for block progressive patterned matrix as in Definition 14. For a block random progressive patterned matrix $\mathbf{S}$ drawn following the distribution $\tilde{\mathcal{D}}_{\alpha^{(1)}, \dots, \alpha^{(t)}}$, there exists a linear subspace of $\mathbb{F}_q^m$ of dimension $m - n$ that contains $\{\mathbf{v} \in \mathbb{F}_q^m \text{ s.t. } hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\}$, with probability at least $1 - mq^{-1}$.*

Proof (Lemma 12). We will prove Lemma 12 by induction on m .

For $m = 1$, since $m \geq n \geq 1$, Definition 14 implies that the matrix $\mathbf{S}$ either (1) consists simply in a single entry $s_{1,1}$ which is picked uniformly at random in $\mathbb{F}_q$ or (2) a constant non-null vector. In the first case, this vector is null with probability q^{-1} and in all cases the set $\{\mathbf{v} \in \mathbb{F}_q \text{ s.t. } hw(\mathbf{v}) = 1 \text{ and } \mathbf{S}\mathbf{v} = 0\}$ is therefore the empty set with probability at least $1 - q^{-1}$. It is thus included in the subspace of dimension 0 with probability at least $1 - q^{-1}$.

We now consider $m \geq 2$ and we assume Lemma 12 proven for all block random progressive patterned matrix drawn from a distribution $\tilde{\mathcal{D}}_{\alpha^{(1)}, \dots, \alpha^{(t)}}$ with strictly less than m columns.

We first assume that the matrix $\mathbf{Y}_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ is the unitary matrix $\mathbf{U}_{n \times m}$ (i.e., does not contain any zero). Then, by assumption, we have $m_i = 1$ and $\alpha^{(i)} = n$ for $i \in \{1, \dots, t\}$. The matrix $\mathbf{S}$ is thus the concatenation of the vector $\mathbf{1}_{n \times 1}$ and a matrix picked from $\mathbb{F}_q^{n \times m-1}$ with the uniform distribution. Using elementary operations on the columns of $\mathbf{S}$, one can obtain a matrix of the form

$$\begin{pmatrix} 1 & \mathbf{0}_{1 \times m-1} \\ \mathbf{u}_{n-1} & \mathbf{S}' \end{pmatrix}$$

where $\mathbf{u}_{n-1} \in \mathbb{F}_q^{n-1}$ is the all-one vector and $\mathbf{S}'$ is a matrix drawn from $\mathbb{F}_q^{n-1 \times m-1}$ with the uniform distribution. As in the proof of Lemma 11, the matrix $\mathbf{S}'$ is of full rank $n - 1$ with probability at least $1 - 2q^{n-m-2}$. The matrix $\mathbf{S}$ is thus of full rank n with probability at least $1 - 2q^{n-m-2}$ and thus with probability at least $1 - mq^{-1}$.

We now assume that the matrix $\mathbf{Y}_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ contains some 0. By assumption, there exists $j \in \{1, \dots, t\}$ such that $\alpha_{m_j}^{(j)} = n$ and in the following, it there exist two indices $j \in \{1, \dots, t\}$ such that $\alpha_{m_j}^{(j)} = n$, we select one such index different from 1.

If $j = 1$, by assumption we have $\alpha_{m_i}^{(i)} < n$ for all $i \in \{2, \dots, t\}$ and the last row of the matrix $\mathbf{S}$ has one coordinate equal to 1 and all other coordinates equal to 0. If $\mathbf{v} \in \mathbb{F}_q$ is of full Hamming weight $hw(\mathbf{v}) = m$, the last coordinate of the vector $\mathbf{S}\mathbf{v}$ is always non-null and the set $\{\mathbf{v} \in \mathbb{F}_q \text{ s.t. } hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\}$ is therefore the empty set. It is thus included in the subspace of dimension 0 with probability at least $1 \geq 1 - mq^{-1}$. We therefore now assume that $j > 1$.

1. We first assume that $m_j > 1$ (i.e. that the column of index $m_1 + \dots + m_j$ consists in $\alpha_{m_j-1}^{(j)} \geq 1$ zeroes followed by $\alpha_{m_j}^{(j)} - \alpha_{m_j-1}^{(j)} = n - \alpha_{m_j-1}^{(j)} \geq 1$ ones).

We consider the submatrix of $\mathbf{Y}_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ obtained by deleting the column of index $m_1 + \dots + m_j$ and the rows of indices i in $\{\alpha_{m_j-1+1}^{(j)}, \dots, \alpha_{m_j}^{(j)}\}$. This submatrix is a block unitary progressive patterned matrix with $n' \leq n$ rows and $m - 1$ columns. We can thus apply the induction hypothesis to the submatrix $\mathbf{S}'$ of $\mathbf{S}$ obtained by deleting the same column and the same rows. We know that with probability $1 - (m - 1)q^{-1}$, there exist a linear subspace V' of dimension $m - 1 - n'$ that contains the set $\{\mathbf{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\mathbf{v}) = m - 1 \text{ and } \mathbf{S}'\mathbf{v} = 0\}$. If V' is of dimension 0, then $\{\mathbf{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\mathbf{v}) = m - 1 \text{ and } \mathbf{S}'\mathbf{v} = 0\} \subseteq \{0\}$ and the set is the empty set. We thus have $\{\mathbf{v} \in \mathbb{F}_q^m, hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\} = \emptyset$ and with probability $1 - (m - 1)q^{-1} \geq 1 - mq^{-1}$, there exist a linear subspace V of dimension $m - n$ that contains this set.

If V' is of dimension $m - 1 - n' > 0$, we can assume without loss of generality that the deleted column of $\mathbf{S}$ to obtain $\mathbf{S}'$ was the last one in the last block (i.e., in a block where $\mathbf{S}$ is a random progressive patterned matrix since $j > 1$).

By permuting some rows and columns, we can write

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}' & \mathbf{0}_{n' \times 1} \\ \mathbf{S}'' & \mathbf{u} \end{pmatrix}$$

where $\mathbf{S}'$ is a $(n - n') \times m - 1$ matrix on which we can apply the induction hypothesis (since $m_j > 1$). Let $\mathbf{v} \in \mathbb{F}_q^m$ such that $hw(\mathbf{v}) = m$ and $\mathbf{S}\mathbf{v} = 0$.

We write $\mathbf{v} = \begin{pmatrix} \mathbf{w} \\ \tau \end{pmatrix}$ where $\mathbf{w} \in \mathbb{F}_q^{m-1}$ and $\tau \in \mathbb{F}_q$ is a scalar. We have $hw(\mathbf{w}) = m - 1$ and $\mathbf{S}'\mathbf{w} = 0$, and therefore $\mathbf{w} \in V'$. Since $\tau \neq 0$ by assumption, the vector $\mathbf{u}$ thus belongs to the image W of V' by $\mathbf{S}''$ (with probability at least $1 - (m - 1)q^{-1}$). Since $j > 1$, note that $\mathbf{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$. We can then conclude as in the proof of Lemma 11.

2. We now assume that $m_i = 1$ for all i such that $\alpha_{m_i}^{(i)} = n$ for $i \in \{1, \dots, t\}$ (i.e. that all the columns with a one in the last row consists only of ones).

Since the matrix $\mathbf{\Upsilon}_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ contains some 0, there exists some $j \in \{2, \dots, t\}$ such that $m_j > 1$ and we consider such a $j \in \{2, \dots, t\}$ for which $\alpha_1^{(j)}$ is minimal.

We consider the submatrix of $\mathbf{\Upsilon}_{\alpha^{(1)}, \dots, \alpha^{(t)}} \in \mathbb{F}_q^{n \times m}$ obtained by deleting the column of index $m_1 + \dots + m_{j-1} + 1$ and the rows of indices in the set $\{1, \dots, \alpha_1^{(j)} - 1\}$. It is easy to see that this submatrix is a block unitary progressive patterned matrix with $n' \leq n - 1$ rows and $m - 1$ columns. We can thus apply the induction hypothesis to the submatrix $\mathbf{S}'$ of $\mathbf{S}$ obtained by deleting the same column and the same rows.

We write $\mathbf{v} = \begin{pmatrix} \mathbf{w} \\ \tau \end{pmatrix}$ where $\mathbf{w} \in \mathbb{F}_q^{m-1}$ and $\tau \in \mathbb{F}_q$ is a scalar. We have $hw(\mathbf{w}) = m - 1$ and $\mathbf{S}'\mathbf{w} = 0$, and therefore $\mathbf{w} \in V'$. Since $\tau \neq 0$ by assumption, the vector $\mathbf{u}$ thus belongs to the image W of V' by $\mathbf{S}''$ (with probability at least $1 - (m - 1)q^{-1}$). Since $j > 1$, note that $\mathbf{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$. We can then conclude as in the proof of Lemma 11.

We know that with probability at least $1 - (m - 1)q^{-1}$, there exists a linear subspace $V' \subseteq \mathbb{F}_q^{m-1}$ of dimension $m - 1 - n'$ that contains the set $\{\mathbf{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\mathbf{v}) = m - 1 \text{ and } \mathbf{S}'\mathbf{v} = 0\}$.

If V' is of dimension 0, then $\{\mathbf{v} \in \mathbb{F}_q^{m-1} \text{ s.t. } hw(\mathbf{v}) = m - 1 \text{ and } \mathbf{S}'\mathbf{v} = 0\} \subseteq \{0\}$ and this set is thus the empty set. We then have $\{\mathbf{v} \in \mathbb{F}_q^m, hw(\mathbf{v}) = m \text{ and } \mathbf{S}\mathbf{v} = 0\} = \emptyset$ with probability at least $1 - (m - 1)q^{-1} \geq 1 - mq^{-1}$, and so there exists a linear subspace V of dimension $m - n$ that contains this set.

If V' is of dimension $m - 1 - n' > 0$, we can assume without loss of generality that the column of $\mathbf{S}$ deleted to obtain $\mathbf{S}'$ was the last one (by permuting the blocks of the matrix). We have the following block-decomposition of $\mathbf{S}$

$$\mathbf{S} = \begin{pmatrix} \mathbf{S}'' & \mathbf{u} \\ \mathbf{S}' & \mathbf{0}_{n' \times 1} \end{pmatrix}$$

where $\mathbf{S}''$ is a $(n - n') \times (m - 1)$ matrix and $\mathbf{u}$ a column vector of dimension $(n - n')$. Note that $\mathbf{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$ and $\mathbf{S}''$. Let $\mathbf{v} \in \mathbb{F}_q^m$ such that $hw(\mathbf{v}) = m$ and $\mathbf{S}\mathbf{v} = 0$. Since $j > 1$, note that $\mathbf{u}$ is a random vector in $\mathbb{F}_q^{n-n'}$ independent from $\mathbf{S}'$. We can then conclude as in the proof of Lemma 11.

This concludes the proof of the lemma. □

Using the same arguments as above for Case 1 (but replacing Lemma 11 by Lemma 12), we obtain that for any set $I \subseteq \{1, \dots, z\}$ of cardinal n such that $\mathbf{M}_I$ has no identically zero column vectors, with probability at least $1 - 2^n \cdot n \cdot q^{-1}$ over the choice of γ , we have $hw(\mathbf{L}_I \cdot \mathbf{v}) < n - K$ or $\mathbf{M}_I \cdot \mathbf{v} \neq \mathbf{0}_n$ for any vector $\mathbf{v} \in \mathbb{F}_q^n$ (where K denotes the number of coordinates $i_1, \dots, i_K \in [z]$ such that $v_{i_1} \neq 0, \dots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \dots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient).

Case 3. We now consider the sets $I \subseteq \{1, \dots, n\} \cup \{(n+3) \cdot n + 1, \dots, z\}$ of cardinal n for which $\mathbf{M}_I$ has some identically zero column vectors (i.e., $I \cap \{1, \dots, n\} \neq \emptyset$). For each $i \in I \cap \{1, \dots, n\} \neq \emptyset$, the i -th column in $\mathbf{L}$ is the i -th vector in the canonical basis of $\mathbb{F}_q^n$ (i.e., it corresponds to a probe of a value a_i). We can consider the submatrix of $\mathbf{M}_I$ and $\mathbf{L}_I$ in which we delete for each $i \in I \cap \{1, \dots, n\} \neq \emptyset$, the i -th column and the i -th row. We denote $\rho = \#I \cap \{1, \dots, n\} \neq \emptyset$.

Let us denote $\mathbf{M}'_I$ and $\mathbf{L}'_I$ the corresponding matrices (with $m' = m - \rho$ columns). These matrices are of the form handled in the previous Case 2 (with $m' < m$). The previous argument shows therefore that with probability at least $1 - 2^n \cdot n \cdot q^{-1}$, we have $hw(\mathbf{L}'_I \cdot \mathbf{v}) < n - \rho - K$ or $\mathbf{M}'_I \cdot \mathbf{v} \neq \mathbf{0}_n$ for any vector $\mathbf{v} \in \mathbb{F}_q^{m'}$ (where K denotes the number of coordinates $i_1, \dots, i_K \in [z]$ such that $v_{i_1} \neq 0, \dots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \dots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient).

Going back to the original matrices $\mathbf{L}_I$ and $\mathbf{M}_I$ we have shown for any set $I \subseteq \{1, \dots, n\} \cup \{(n+3) \cdot n + 1, \dots, z\}$ of cardinal n , with probability at least $1 - 2^n \cdot n \cdot q^{-1}$ over the choice of γ , we have $hw(\mathbf{L}_I \cdot \mathbf{v}) < n - K$ or $\mathbf{M}_I \cdot \mathbf{v} \neq \mathbf{0}_n$ for any vector $\mathbf{v} \in \mathbb{F}_q^n$ (indeed a vector $\mathbf{v}$ satisfies $\mathbf{M}_I \cdot \mathbf{v} = \mathbf{0}_n$ if and only if $\mathbf{M}'_I \cdot \mathbf{v}' = \mathbf{0}_n$ where $\mathbf{v}'$ denotes the restriction of $\mathbf{v}$ to the support $I \cap \{1, \dots, n\}$ and the Hamming weight of $hw(\mathbf{L}_I \cdot \mathbf{v})$ is at smaller than $hw(\mathbf{L}_I \cdot \mathbf{v}') + \rho$ since at most ρ positions can be set arbitrarily).

Case 4. We now consider all sets $I \subseteq \{1, \dots, z\}$ (with no restrictions). Without loss of generality, we can assume that all not identically zero column vectors in $\mathbf{M}_I$ are pairwise distinct. Indeed, if two columns are equal, they come either from the two submatrices I_n of $\mathbf{M}$, or from the first column vectors of a submatrix I_n and the submatrix $\mathbf{T}_n$, or from the first column vectors of a submatrix $\mathbf{D}_{\gamma,i}$ for some $i \in \{1, \dots, n\}$ and the corresponding submatrix $\mathbf{T}_{\gamma,i}$. In all these cases, one can replace the index of the second vector in I by an index in $\{1, \dots, n-1\}$ (and modify the vector accordingly) in such a way that $\mathbf{M}_{I'}$ for the new set I' has a new null column vector for each duplicate in the original matrix $\mathbf{M}_I$.

We can now delete the columns corresponding to the null vectors as in Case 3 (i.e., for each $i \in I \cap \{1, \dots, n+1\} \neq \emptyset$, the i -th column and the i -th row in $\mathbf{M}_I$ and $\mathbf{L}_I$). The only difference occurs if a column in $\mathbf{M}_I$ is equal to the i -th vector in the canonical basis (for $i \geq 2$) or to the scalar multiplication of this vector by some element of the matrix $\gamma \in \mathbb{F}_q$ (corresponding to the cases $I \cap \{n+1, \dots, 2n\} \neq \emptyset$ and $I \cap \{2n+1, \dots, (n+3) \cdot n + 1\} \neq \emptyset$ respectively). As in Case 3, we can delete the corresponding column and row in $\mathbf{M}_I$ and $\mathbf{L}_I$ (i.e., it corresponds to a probe of a value r_i , a value $a_i + r_i$ or a value $a_i + \gamma_{j,i} r_i$).

As above, if we denote $\mathbf{M}'_I$ and $\mathbf{L}'_I$ the corresponding matrices (with m' columns and $n' < n$ and $n'+1$ rows, respectively), the previous argument shows that with probability at least $1 - 2^n \cdot n \cdot q^{-1}$, we have $hw(\mathbf{L}'_I \cdot \mathbf{v}) < n' - K$ or $\mathbf{M}'_I \cdot \mathbf{v} \neq \mathbf{0}_n$ for any vector $\mathbf{v} \in \mathbb{F}_q^{m'}$ (where K denotes the number of coordinates $i_1, \dots, i_K \in [z]$ such that $v_{i_1} \neq 0, \dots, v_{i_K} \neq 0$ and the corresponding columns $i_1, \dots, i_K$ in $\mathbf{L}$ and in $\mathbf{M}$ have no zero coefficient).

Going back to the original matrices $\mathbf{L}_I$ and $\mathbf{M}_I$ we have shown for any set $I \subseteq \{1, \dots, z\}$ of cardinal n , with probability at least $1 - 2^n \cdot n \cdot q^{-1}$ over the choice of γ , we have $hw(\mathbf{L}_I \cdot \mathbf{v}) < n - K$ or $\mathbf{M}_I \cdot \mathbf{v} \neq \mathbf{0}_n$ for any vector $\mathbf{v} \in \mathbb{F}_q^n$

Conclusion . By the union on all such sets, we obtain that the probability that, for γ picked uniformly at random in $\mathbb{F}_q^{n \times n}$, the matrix $\mathbf{M}$ satisfies Condition 3, i.e., for any vector $\mathbf{v} \in \mathbb{F}_q^z$ of Hamming weight $hw(\mathbf{v}) \leq n$ we have $hw(\mathbf{L} \cdot \mathbf{v}) < n - K$ or $\mathbf{M} \cdot \mathbf{v} \neq \mathbf{0}_n$ is at least

$$1 - \binom{z}{n} 2^n \cdot n \cdot q^{-1} = 1 - \binom{(2n+4) \cdot n + 1}{n} 2^n \cdot n \cdot q^{-1}.$$

The binomial coefficient in this lower-bound is always less than $(6n)^n$ (this can be checked by hand for small values of n and it follows for large values using the classical upper-bound $\binom{r}{s} \leq ((r \cdot \exp(1))/s)^s$). We thus obtain the claimed bounds and this concludes the proof. $\square$

A.13 Instantiations

In this paragraph, we present explicit matrices obtained following [19] that achieve our Condition 3 and can thus be used to instantiate our new multiplication gadget.

A first matrix for 3 shares can be used over the finite field $\mathbb{F}_{2^5}$ represented as $\mathbb{F}_2[X]/(X^5 + X^2 + 1)$:

$$\gamma = \begin{pmatrix} X+1 & X & X^2+1 \\ X & X^2+1 & X+1 \\ X^2+1 & X+1 & X \end{pmatrix}$$

Another matrix for 3 shares (denoted in hexadecimal by evaluating each polynomial at $X = 2$ and writing the result in base 16) can be used over the finite field $\mathbb{F}_{2^6}$ represented as $\mathbb{F}_2[X]/(X^6 + X + 1)$:

$$\gamma = \begin{pmatrix} 36 & 30 & 1d \\ 21 & 05 & 1a \\ 35 & 31 & 1b \end{pmatrix}$$

Another example for 4 shares can be instantiated using the following matrix (also denoted in hexadecimal) over the (AES) finite field $\mathbb{F}_{2^8}$ represented as $\mathbb{F}_2[X]/(X^8 + X^4 + X^3 + X + 1)$:

$$\gamma = \begin{pmatrix} 2d & f5 & 2e & 23 \\ e1 & c3 & ac & 30 \\ bd & f6 & fa & 8a \\ e6 & 4a & 4d & ab \end{pmatrix}$$

Eventually, we present a matrix for 5 shares over the finite field $\mathbb{F}_{2^{10}}$ represented as $\mathbb{F}_2[X]/(X^{10} + X^3 + 1)$:

$$\gamma = \begin{pmatrix} 225 & 2a9 & 0d0 & 224 & 2dd \\ 254 & 11b & 325 & 3a6 & 219 \\ 3d2 & 2bc & 2bf & 3a2 & 2a1 \\ 2af & 311 & 295 & 26b & 11d \\ 16c & 124 & 158 & 319 & 0b8 \end{pmatrix}$$

Appendix D

IronMask: Versatile Verification of Masking Security

Hereafter is appended the full version of our paper [BMRT22], joint work with Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb, published at **S&P 2022**.

IronMask: Versatile Verification of Masking Security

Sonia Belaïd¹, Darius Mercadier¹, Matthieu Rivain¹, and Abdul Rahman Taleb^{1,2}

¹ CryptoExperts, France

² Sorbonne Université, CNRS, LIP6, F-75005 Paris, France

{sonia.belaid,darius.mercadier,matthieu.rivain,abdul.taleb}@cryptoexperts.com

Abstract. This paper introduces **IronMask**, a new versatile verification tool for masking security. **IronMask** is the first to offer the verification of standard simulation-based security notions in the probing model as well as recent composition and expandability notions in the random probing model. It supports any masking gadgets with linear randomness (*e.g.* addition, copy and refresh gadgets) as well as quadratic gadgets (*e.g.* multiplication gadgets) that might include non-linear randomness (*e.g.* by refreshing their inputs), while providing complete verification results for both types of gadgets. We achieve this complete verifiability by introducing a new algebraic characterization for such quadratic gadgets and exhibiting a complete method to determine the sets of input shares which are necessary and sufficient to perform a perfect simulation of any set of probes. We report various benchmarks which show that **IronMask** is competitive with state-of-the-art verification tools in the probing model (**maskVerif**, **scVerif**, **SILVER**, **matverif**). **IronMask** is also several orders of magnitude faster than **VRAPS** –the only previous tool verifying random probing composability and expandability– as well as **SILVER** –the only previous tool providing complete verification for quadratic gadgets with non-linear randomness. Thanks to this completeness and increased performance, we obtain better bounds for the tolerated leakage probability of state-of-the-art random probing secure compilers.

Keywords: Side-channel security, masking, physical defaults, automatic verification, complete verification, composition, probing model, random probing model, **IronMask**

1 Introduction

Side-channel attacks exploit the physical leakage of a device executing cryptographic implementations to extract the manipulated secrets. They can be built from cheap equipment and are generally able to recover the keys in a limited time in the absence of specific protections.

Since the discovery of these attacks in the late nineties, the community investigated several approaches to counteract them. Among these approaches, *masking* [22,30] is one of the most deployed in practice. The main idea of masking is to split the information between several variables called *shares*. In a nutshell, for t^{th} -order Boolean masking, each sensitive value x is split into $(t+1)$ shares $x_1, \dots, x_{t+1}$. The first t shares $x_1, \dots, x_t$ are generated uniformly at random and the last one x_{t+1} is computed as $x_{t+1} \leftarrow x \oplus x_1 \oplus \dots \oplus x_t$. Doing so, the attacker must collect and then aggregate the information from all the shares to recover sensitive data. This becomes exponentially hard with the number of shares as each observation comes with noise.

While Boolean masking can be easily implemented for linear operations that are directly applied on each share separately, it becomes much more complex for non-linear computations for which shares must be interleaved. These operations require additional randomness to preserve the security order.

To properly reason on the security of these operations, and more generally on the security of masked implementations, the leakage is generally formalized in so-called leakage models. The most famous one is probably the t -probing model, introduced by Ishai, Sahai, and Wagner in 2003 [32].

In this model, the leakage is modeled as the exact values of t intermediate variables chosen by the attacker. Variants of this model include the *robust probing model* [28] in hardware scenarios, which considers wider leakage to model physical effects such as glitches. Instead of t intermediate variables, the attacker gets t sets of variables that belong to the same combinatorial set. While probing-like models are very convenient to build security proofs, they sometimes fail to closely reflect the reality of embedded devices. For instance, they do not capture *horizontal attacks* [9], which exploit in particular the repeated manipulation of variables within an execution.

Therefore, the community is starting to focus on more realistic leakage models, such as the *random probing model* [1,26]. The leakage is assumed to gather the exact value carried out by each wire of the circuit with probability p . The security tightly reduces to the security in the *noisy leakage model* [35,26] in which each variable leaks a noisy function of its value. The random probing model can also be extended to capture glitches or other side effects.

In any model, the security of almost all masking approaches relies on the security of small gadgets (*i.e.*, atomic masked operations) and their composition. For instance, in the probing model and its variants, the type-system of Barthe *et al.* [6] makes it possible to securely compose small gadgets that are proven to be (strong) non-interferent. The t -non-interference is itself satisfied when any set of t intermediate variables can be perfectly simulated from t input shares while the t -strong non-interference adds a condition on the output observations. Both properties imply t -probing security. Similarly, in the random probing model, Ananth *et al.* [2] use an expansion strategy on top of secure multi-party computation protocols while Belaïd *et al.* use an expansion strategy on top of random probing expandable gadgets [13,16,17]. The latter works only require gadgets that are masked at small orders to build circuits achieving an arbitrary level of random probing security.

Even at reasonable orders, the manual verification of security properties on small gadgets has been shown to be very error-prone [24]. Therefore, automatic tools are regularly built to apply a formal verification. For instance, two existing tools currently check random probing properties (namely, VRAPS [13] and STRAPS [20]). However, none of these two tools provide *completeness* (*i.e.*, they can falsely deem a set of leaking variables as insecure with respect to the secret). In terms of efficiency, VRAPS makes it difficult to verify even small gadgets at reasonable orders and STRAPS only manages to do it using verification rules from the underlying tool maskVerif [5,6,3] (which only verifies probing security properties), which by construction, are not complete.

As for tools which achieve complete verification of some (probing like) security properties, we count only SILVER [33], which suffers from low performance, and matverif [19], which is restricted to specific gadgets only.

Our contributions. In this work, we introduce IronMask, a new automatic verification tool to check probing and random probing security properties using complete and efficient procedures. Our main contributions are the following:

- We formalize all of the probing and random probing properties of the state of the art from a single common building block, a function we call SIS, and show that all of the security properties can be verified using a unique instantiation of that function (Section 2). In a nutshell, SIS receives as input a set of probes on a gadget (and the description of the corresponding gadget) and performs a number of operations on the algebraic expressions of the probes in order to determine the exact sets of input shares which are necessary and sufficient to perform a perfect simulation of these probes. While SIS partially uses some properties from the state of the art,

it was not clearly exhibited before, and the unification of all the (random) probing-like security notions with respect to this function was not explicitly well-defined.

- We extend the algebraic characterization introduced in [11,12] of gadgets with linear randomness (*i.e.* all random values are additive on the wires of the gadget) to more general gadgets with non-linear randomness which perform quadratic operations on input shares mixed with randomness (Section 3). Our extended characterization notably captures recent gadget designs [9,16], achieving advanced security properties such as resistance to *horizontal attacks* or random probing security. From this characterization, we provide a complete verification method that applies to most (if not all) masking gadgets for standard operations (addition, multiplication, refreshing, etc.). In comparison, the only previously existing complete verification method for such general gadgets would rely on exhausting the truth table of tuples of intermediate variables with respect to the inputs and the randomness, which is highly inefficient.
- We introduce **IronMask**, a new versatile verification tool for all probing and random probing-like properties in the state of the art (Section 4). **IronMask** supports the verification of traditional gadgets with linear randomness, as well as newly formalized gadgets with non-linear randomness along with a complete verification method for both types of gadgets based on our extended algebraic characterization. **IronMask** implements several optimizations to make the verification faster. We benchmark the performance of our new tool (Section 5) and show that it is competitive with state-of-the-art verification tools in the probing model (**maskVerif**, **scVerif**, **SILVER**, **matverif**). **IronMask** is also several orders of magnitude faster than **VRAPS**, the only previous tool verifying random probing composability and expandability, and **SILVER**, the only previous tool providing complete verification for gadgets with non-linear randomness. **IronMask** uses complete methods for the verification, unlike **VRAPS** and **STRAPS** which are the only verification tools in the random probing model. Thanks to this completeness and increased performance, we obtain better bounds for the tolerated leakage probability³ of state-of-the-art random probing secure compilers. **IronMask** is open-source and publicly available at:

<https://github.com/CryptoExperts/IronMask>

2 Characterization of Security Notions for Masking Gadgets

2.1 Preliminaries

Throughout the paper, $\mathbb{K}$ shall denote a finite field. For any $n \in \mathbb{N}$, we shall denote $[n]$ the integer set $[n] = [1, n] \cap \mathbb{Z}$. For any tuple $\vec{x} = (x_1, \dots, x_n) \in \mathbb{K}^n$ and any set $I \subseteq [n]$, we shall denote $\vec{x}|_I = (x_i)_{i \in I}$.

An *arithmetic circuit* over a field $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations over $\mathbb{K}$ (*e.g.*, additions, multiplications, copies). A *randomized arithmetic circuit* is equipped with an additional random gate of fan-in 0 which outputs a fresh uniform random value of $\mathbb{K}$.

In the following, the *n-linear decoding* mapping, denoted **LinDec**, refers to the function $\bigcup_n \mathbb{K}^n \rightarrow \mathbb{K}$ defined as

$$\text{LinDec} : (x_1, \dots, x_n) \mapsto x_1 + \dots + x_n ,$$

³ Increased performance yields a tighter approximation of the *failure* probability in the random probing model, while completeness allows to compute more accurate and complete values for that probability. Both factors result in a tighter, more accurate approximation of the failure probability, from which tighter bounds on the tolerated leakage probability are derived. See Section 5.1 for more details.

for every $n \in \mathbb{N}$ and $(x_1, \dots, x_n) \in \mathbb{K}^n$. We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $(\vec{x}_1, \dots, \vec{x}_\ell) \in (\mathbb{K}^n)^\ell$ the n -linear decoding mapping acts as

$$\text{LinDec} : (\vec{x}_1, \dots, \vec{x}_\ell) \mapsto (\text{LinDec}(\vec{x}_1), \dots, \text{LinDec}(\vec{x}_\ell)) .$$

For any $x \in \mathbb{K}$, an n -linear sharing of x is a vector $\vec{x} \in \mathbb{K}^n$ such that $\text{LinDec}(\vec{x}) = x$. A random vector $\vec{x}$ distributed over $\mathbb{K}^n$ is said to be a *uniform sharing* of $x := \text{LinDec}(\vec{x})$ if for any set $I \subseteq [n]$ with $|I| < n$ the random vector $\vec{x}|_I$ is uniformly distributed over $\mathbb{K}^{|I|}$.

In the following, we shall call an $(n\text{-share}, \ell\text{-to-}m)$ gadget denoted G for some function $g : \mathbb{K}^\ell \rightarrow \mathbb{K}^m$, a randomized arithmetic circuit that maps an input $(\vec{x}_1, \dots, \vec{x}_\ell) \in (\mathbb{K}^n)^\ell$ to an output $(\vec{y}_1, \dots, \vec{y}_m) \in (\mathbb{K}^n)^m$ such that

$$(\text{LinDec}(\vec{y}_i))_{1 \leq i \leq m} = g \left[(\text{LinDec}(\vec{x}_i))_{1 \leq i \leq \ell} \right]$$

with probability 1 over the internal randomness of G .

For any wire on a randomized arithmetic circuit, we shall call a *probe* on this wire, the symbolic expression of the circuit inputs (shares) and the generated randoms (outputs of random gates) associated to this wire.

In the following, we shall say that a tuple $\vec{P}$ of symbolic expressions of input sharings $\vec{x}_1, \dots, \vec{x}_\ell$ (of size n) and randomness $\vec{r}$ (of size ρ) can be *perfectly simulated* from $\vec{x}_1|_{I_1}, \dots, \vec{x}_\ell|_{I_\ell}$ if and only if for any distributions $\mathcal{D}_{\vec{x}_1}, \dots, \mathcal{D}_{\vec{x}_\ell}$ over $\mathbb{K}^n$ there exists a probabilistic algorithm $\mathcal{S}$ (the simulator) such that given $\vec{x}_1 \leftarrow \mathcal{D}_{\vec{x}_1}, \dots, \vec{x}_\ell \leftarrow \mathcal{D}_{\vec{x}_\ell}, \vec{r} \leftarrow \mathbb{K}^\rho$, we have

$$\mathcal{S}(\vec{x}_1|_{I_1}, \dots, \vec{x}_\ell|_{I_\ell}) \stackrel{\text{id}}{=} \text{eval}_{\vec{P}}(\vec{x}_1, \dots, \vec{x}_\ell, \vec{r}) , \quad (1)$$

where $\leftarrow$ means randomly sampling from a set or a distribution, $\stackrel{\text{id}}{=}$ means identically distributed and $\text{eval}_{\vec{P}}$ is the function which outputs an evaluation of the symbolic expressions in $\vec{P}$ from its arguments.

2.2 Security Notions

To be composed in secure circuits, gadgets are generally expected to satisfy slightly stronger security notions than (random) probing security⁴. In a nutshell, security proofs generally demonstrate that the observations made on a gadget and/or on its output shares can be perfectly simulated from a subset of its input shares. If this subset is strictly smaller than the set of input shares and if the sharing is uniform (which is the case in tight masking circuits), then the observations can be simulated independently from the secrets and are therefore independent from the secret. Such properties also make it possible to compose secure gadgets by analyzing the propagation of the simulated observations.

Although many different security notions have been introduced to build proofs of gadgets in the (random) probing model, we show that they can almost all be defined on top of a single building block: the *set of input shares* (SIS) function. The latter takes as input a set of probes on internal wires of the gadget as well as a set of output shares, and returns a set of input shares necessary (and sufficient) to perfectly simulate these internal probes and output shares. We formalize the SIS

⁴ Although some composition rules were established directly between probing secure gadgets (e.g., [15]), their scope remains limited.

primitive in Definition 1 before showing how to use it to express state-of-the-art properties. For the sake of clarity, we restrain the following definitions to the case of single output gadgets (which is the most common case), but the extension to multi-output gadgets is straightforward. We denote SIS_G to be SIS with input gadget G .

Definition 1. Let G be an $(n\text{-share}, \ell\text{-to-1})$ gadget mapping ℓ input sharings $(\vec{x}_1, \dots, \vec{x}_\ell) \in (\mathbb{K}^n)^\ell$ to an output sharing $\vec{y} \in \mathbb{K}^n$. Let $\vec{P}$ be a set of probes on G and $O \subseteq [n]$ a set of output indices. The function SIS_G maps $\vec{P}$ and O to the unique smallest sets of input indices $I_1, \dots, I_\ell$ such that $(\vec{P}, \vec{y}|_O)$ can be perfectly simulated from $\vec{x}_1|_{I_1}, \dots, \vec{x}_\ell|_{I_\ell}$.

Note that for any gadget G , the *smallest* set of input shares returned by SIS_G is *uniquely* defined from the result [11, Lemma 7.5], which demonstrates that if a set of probes can be simulated from different sets of inputs shares, then it can also be simulated by the intersection of these sets.

Probing security notions. We now formalize the *probing-like* security notions (*i.e.*, to achieve security and secure composition in the probing model) for any n -share, ℓ -to-1 gadget G (all these notions can be generalized for the case of multiple outputs). Definition 2 recalls the non-interference (NI) property from [6].

Definition 2 (t-NI). A gadget G is t -NI if for any tuple $\vec{P}$ of t_1 internal probes and any set O of t_2 output share indices such that $t_1 + t_2 \leq t$, the sets $(I_1, \dots, I_\ell) := \text{SIS}_G(\vec{P}, O)$ satisfy $|I_i| \leq t, \forall i \in [\ell]$.

Other common probing-like properties can be defined in a similar way by changing the condition on the sets $(I_1, \dots, I_\ell)$ in the output of the SIS_G primitive. We list these conditions in Table 1 for some of the most common probing-like properties, with respect to t and O . While most of the properties are interesting in the context of composing secure gadgets to achieve global security, directly verifying the probing security of a complete implementation is useful in some cases, such as analyzing a complete 2-share AES implementation. To represent this case, we denote PS^* the SIS-based probing security definition. PS^* is actually equivalent to the case of $(n-1)$ -NI with $O = \emptyset$ and $t = n-1$. We list this property in Table 1 as well.

Table 1: Probing-like security notions from the basic primitive SIS_G . t_1 indicates the number of probes on input and intermediate variables, while t_2 indicates the number of probes on output shares, with $t = t_1 + t_2$.

Notion	Condition
t -NI [5]	$ I_i \leq t, \forall i \in [\ell]$
t -SNI [6]	$ I_i \leq t_1, \forall i \in [\ell]$
t -TNI [11]	$ I_i \leq t_1 + t_2, \forall i \in [\ell]$
(t, f) -NI [4]	$ I_i \leq f(t_1, t_2), \forall i \in [\ell]$
t -PINI [21]	$ (\cup_i I_i) \setminus O \leq t_1$
PS^*	$ I_i \leq n-1, \forall i \in [\ell]$

Because the probing model is often criticized as it fails to reflect the reality of embedded systems (see for instance *horizontal attacks* [9]), the community recently started to study the random probing model. Despite its complexity compared to the simple probing model, the random probing model was shown to be closer to the more realistic noisy leakage model, reducing the gap between theoretical proofs and concrete observations.

Random probing security notions. To formalize security notions in the random probing model, we rely on the **LeakingWires** procedure formalized in [13]. This (probabilistic) procedure outputs a tuple of probes $\vec{P}$ on the gadget G such that each wire of G is added to $\vec{P}$ independently with probability p , where p is the *leakage probability* (a.k.a. *leakage rate*). Definition 3 recalls the *random probing composability* (RPC) notion [13] based on the **SIS** primitive.

Definition 3. Let $p, \varepsilon \in [0, 1]$ and $n, t \in [0, n]$. Let G be a n -share ℓ -to-1 gadget and let $\vec{P}$ be the random vector defined as $\vec{P} = \text{LeakingWires}(G, p)$. Then G is (t, p, ε) -RPC if for every $O \subseteq [n]$ with $|O| = t$, the sets $(I_1, \dots, I_\ell) = \text{SIS}_G(\vec{P}, O)$ satisfy $\Pr[(|I_1| > t) \vee \dots \vee (|I_\ell| > t)] \leq \varepsilon$, where the probability is taken over all tuples of probes $\vec{P}$ obtained through **LeakingWires**(G, p).

We call the event $((|I_1| > t) \vee \dots \vee (|I_\ell| > t))$ a *failure event* (failure of a perfect simulation) and ε is the *failure probability* or the probability of a failure event to occur. The *random probing expandability* (RPE) notions introduced and analyzed in [13,16,17] can also be defined in a similar fashion. Like the authors of [13,16,17], we restrict its definition to 2-input circuits for the sake of clarity but recall that the extension is straightforward. We have that G is (t, p, ε) -RPE1 (resp. RPE2) if for every $O \subseteq [n]$ with $|O| = t$ (resp. if there exists $O \subseteq [n]$ with $|O| = n - 1$), the sets $(I_1, I_2) = \text{SIS}_G(\vec{P}, O)$ satisfy

$$(\Pr[|I_2| > t] \leq \varepsilon) \wedge (\Pr[|I_2| > t] \leq \varepsilon) \wedge (\Pr[(|I_1| > t) \wedge (|I_2| > t)] \leq \varepsilon^2) .$$

The three random probing notions are summarized in Table 2. As in the probing security case earlier, it can be useful to directly verify the random probing security of a complete implementation. To represent this case, we denote (p, ε) -RPS* the **SIS**-based definition of random probing security. This notion is actually similar to the RPC definition, except that we do not consider probes on the outputs, *i.e.* $O = \emptyset$, and a failure occurs when all the shares (of one input) are necessary to perfectly simulate the probes, *i.e.* the failure event is

$$\Pr[(|I_1| = n) \vee \dots \vee (|I_\ell| = n)] \leq \varepsilon .$$

In [13], the authors introduce a method to verify random probing properties by computing the

Table 2: Random probing-like security notions from the basic primitive SIS_G .

Notion	Output O	Condition(s)
(t, p, ε) -RPC [13]	$\forall O, O = t$	$\Pr[(I_1 > t) \vee \dots \vee (I_\ell > t)] \leq \varepsilon$
(t, p, ε) -RPE1 [13]	$\forall O, O = t$	$(\forall i, \Pr[(I_i > t)] \leq \varepsilon) \wedge (\Pr[(I_1 > t) \wedge (I_2 > t)] \leq \varepsilon^2)$
(t, p, ε) -RPE2 [13]	$\exists O, O = n - 1$	$(\forall i, \Pr[(I_i > t)] \leq \varepsilon) \wedge (\Pr[(I_1 > t) \wedge (I_2 > t)] \leq \varepsilon^2)$
(p, ε) -RPS*	$O = \emptyset$	$\Pr[(I_1 = n) \vee \dots \vee (I_\ell = n)] \leq \varepsilon$

failure probability ε as a function $f(p)$ of the leakage probability p . For (p, ε) -RPS* of an n -share gadget of s wires for example, $\varepsilon = f(p)$ is computed as

$$f(p) = \sum_{\substack{\vec{P} \text{ s.t. } (I_1, \dots, I_\ell) = \text{SIS}(\vec{P}, \emptyset) \\ |I_1| = n \vee \dots \vee |I_\ell| = n}} p^{|\vec{P}|} (1 - p)^{s - |\vec{P}|} . \quad (2)$$

In the above equation, we consider that each tuple of probes $\vec{P}$ on a gadget can exactly leak with probability $p^{|\vec{P}|} (1 - p)^{s - |\vec{P}|}$ since each of the wires in $\vec{P}$ is added independently with probability p ,

and each of the remaining wires does not leak with probability $1 - p$. Then, out of all such possible tuples of wires, $f(p)$ represents the sum over the probabilities of obtaining tuples of probes only for which we get a failure event using **SIS** (the failure event being $(|I_1| = n \vee \dots \vee |I_\ell| = n)$ in this context). For a gadget with a total of s wires, computing $f(p)$ then amounts to computing

$$f(p) = \sum_{i=1}^s c_i p^i (1-p)^{s-i} \quad (3)$$

where we simply group the probabilities with respect to the size of the tuples of probes. In other words, c_i is the number of tuples of i wires, for which we obtain a failure event using **SIS**. For instance, if there are exactly 2 tuples of probes $\vec{P}_1, \vec{P}_2$ for which we get a failure event and such that $|\vec{P}_1| = |\vec{P}_2| = 3$, then we get $c_3 = 2$ in equation (3). For other random probing properties, the computation is similar with the difference of considering the correct failure event with the correct t , and the condition on the output set of shares O which is not empty anymore. As shown in [13], a set of (t, p, ε) -RPE gadgets with $\varepsilon = f(p)$ can be expanded into a probing secure circuit (with arbitrary security level) for any leakage probability smaller than p_{max} , with $p_{max} \in [0, 1]$ being the solution of the equation $f(p) = p$. This threshold is called the *tolerated leakage probability* of the gadgets.

The recent Probe Distribution Table (PDT) of Cassiers *et al.* [20] can also be expressed in terms of our basic function SIS_G . The PDT is a two-dimensional table indexed by all possible sets of input indices $\vec{I} = (I_1, \dots, I_\ell)$ where $I_i \subseteq [n]$ and by all possible sets of output indices $O \subseteq [n]$, defined as

$$\text{PDT}[\vec{I}][O] := \sum_{\vec{P} \text{ s.t. } \vec{I} = \text{SIS}(\vec{P}, O)} p^{|\vec{P}|} (1-p)^{s-|\vec{P}|} \quad (4)$$

where s is the number of wires in G . In other words, each entry in the PDT is a different function $f(p)$ as in equations (2), (3). Computing the PDT amounts to considering each possible tuple of probes $\vec{P}$ on the gadget, and compute $\text{SIS}_G(\vec{P}, O) = \vec{I} = (I_1, \dots, I_\ell)$ for each possible set of output shares indices $O \subseteq [n]$. Then, update the corresponding function in the PDT indexed by $\vec{I}$ and O as $\text{PDT}[\vec{I}][O] = \text{PDT}[\vec{I}][O] + p^{|\vec{P}|} (1-p)^{s-|\vec{P}|}$. When exploring all the possible sets of internal probes $\vec{P}$ and all the sets of output indices O , the output of SIS_G shall serve as a basis to compute the expected distributions.

We showed how standard probing and random probing security notions can be expressed in terms of the SIS_G function. In the next section, we focus on the algebraic characterization of masking gadgets and the concrete evaluation of the SIS_G function.

3 Algebraic Characterization of Masking Gadgets

In this section, we recall and extend the algebraic characterization of masking gadgets, and the subsequent security results. Previous works [11,12] considered gadgets with linear randomness, *i.e.* all random values are additive on the wires of the gadget. We refer to these gadgets as LR-gadgets (see for instance multiplications and refresh gadgets from [32]). In this work, we extend the characterization to gadgets with non-linear randomness, *i.e.* on gadget performing non-linear operations on input shares mixed with randomness. We denote these gadgets as NLR-gadgets. Our extended characterization notably captures recent gadget designs, see *e.g.* [9,16], achieving

advanced security properties such as resistance to horizontal attacks or random probing security. We also show how to verify the security of masking gadgets using this algebraic characterization by a concrete evaluation of the SIS primitive, which will be the core primitive of **IronMask**.

3.1 Characterization of Gadgets with Linear Randomness

In this paper, we call an LR-gadget any ℓ -to- m gadget $G : (\vec{x}_1, \dots, \vec{x}_\ell) \mapsto (\vec{y}_1, \dots, \vec{y}_m)$ with the output of the form:

$$(\vec{y}_1, \dots, \vec{y}_m) := R(F(\vec{x}_1, \dots, \vec{x}_\ell), \vec{r}) ,$$

where F is any arithmetic circuit, R is a linear arithmetic circuit (*i.e.* computing a linear function) and $\vec{r}$ is a vector of internal randomness uniformly drawn from $\mathbb{K}^\rho$. Formally, each coordinate of $\vec{r}$ is the output of a randomness gate of G , and F and R are composed solely of operation gates. Note that this characterization is more general than the one from [11,12], which only considers quadratic circuits for F . We show hereafter that we can still obtain an efficient and complete evaluation of SIS for those gadgets, which in turn yields an efficient verification of the considered security notions.

By definition, any probe on an LR-gadget can be written as

$$p = f_p(\vec{x}_1, \dots, \vec{x}_\ell) + \vec{r}^T \cdot \vec{s}_p \quad (5)$$

for some arithmetic function $f_p : (\mathbb{K}^n)^\ell \rightarrow \mathbb{K}$ and some constant vector $\vec{s}_p \in \mathbb{K}^\rho$.

Given a tuple of probes $\vec{P} = (p_1, \dots, p_d)$ on the gadget G , we are interested in determining the set of input shares necessary for a perfect simulation of all probes in $\vec{P}$. In particular, if $\vec{P}$ can be simulated with at most $n - 1$ of each input sharing, then we know that $\vec{P}$ is independent of the secret inputs. Belaïd *et al.* [11] showed how to use a Gaussian elimination technique in order to determine the simulatability of a tuple of probes for gadgets with linear randomness over the binary field. This technique was later extended to any finite field in [12]. We base the verification procedure for LR-gadgets in our tool on this technique.

We start by stating the result with Gaussian elimination from [11,12] in a different formulation that is more convenient for our purposes. For this, we first define a simple function `shares(.)`, which takes as input a tuple of symbolic expressions $(e_1, \dots, e_d)$ of the input shares, *i.e.* $e_i = f_{e_i}(\vec{x}_1, \dots, \vec{x}_\ell)$ for some algebraic function f_{e_i} , and which outputs the (smallest) sets of indices $I_1, \dots, I_\ell$ such that $(e_1, \dots, e_d)$ functionally depends on $(\vec{x}_1|_{I_1}, \dots, \vec{x}_\ell|_{I_\ell})$. Notice that evaluating `shares(.)` simply consists in extracting the indices of the input shares that are contained in the symbolic expressions $(e_1, \dots, e_d)$. We stress that the input shares $(\vec{x}_1|_{I_1}, \dots, \vec{x}_\ell|_{I_\ell})$ where $(I_1, \dots, I_\ell) := \text{shares}(e_1, \dots, e_d)$ are necessary and sufficient for a perfect simulation of $(e_1, \dots, e_d)$. Note that `shares(.)` is executed on the tuple of expressions that is the output tuple of the Gaussian elimination technique. In fact, after executing the Gaussian elimination, we are guaranteed that the remaining expressions cannot be simplified any further in the given field $\mathbb{K}$ and they are solely formed of operations between input shares (they do not include any random variables). In this case, to perfectly simulate the resulting tuple (which is equivalent to perfectly simulating the tuple given before Gaussian elimination), there is no choice but to have access to all of the input shares that are involved in the remaining expressions, which is why `shares(.)` simply extracts the indices of these input shares.

Lemma 1. *Let G be an n -share gadget. Let $\vec{P} = (p_1, \dots, p_d)$ be a tuple of probes on G . Let $S \in \mathbb{K}^{d \times \rho}$ be the matrix such that*

$$S = (\vec{s}_{p_1} \mid \vec{s}_{p_2} \mid \dots \mid \vec{s}_{p_d})$$

(i.e. each $\vec{s}_{p_i}$ is a column vector of S) and let S' be the row reduced form of the matrix S such that S' is of the form

$$S' = \begin{pmatrix} 0_{m,d-m} & 0_{m,R-d+m} \\ I_{d-m} & S'' \end{pmatrix}$$

up to some permutations on the rows with $S' = N \cdot S$ where N is an invertible matrix in $\mathbb{K}^{d \times d}$. Let $\vec{P}'$ be defined as

$$\vec{P}' = N \cdot \vec{P} = (p'_1, \dots, p'_m, p'_{m+1}, \dots, p'_d) .$$

Then, the set of input shares necessary to simulate the probes in $\vec{P}$ is $\text{shares}(p'_1, \dots, p'_m)$.

Sketch of proof. The proof of the result follows the proof of Theorem 3.1 from [11] and Theorem 3.2 of [12]. It is shown in the latter that we can perfectly simulate the probes in $\vec{P}$ by perfectly simulating all probes in $\vec{P}'$, since the matrix N is invertible and we can obtain $\vec{P}$ from $N^{-1} \cdot \vec{P}'$. Then, to perfectly simulate probes in the tuple $\vec{P}'$, we observe from S' that each algebraic expression in the tuple $(p'_{m+1}, \dots, p'_d)$ contains a random value that does not appear in any other algebraic expression in $\vec{P}'$. We can thus perfectly simulate $(p'_{m+1}, \dots, p'_d)$ by generating $d - m$ uniform random values without the need of any input shares. The remaining algebraic expressions $(p'_1, \dots, p'_m)$ contain no random values and are all of the form $p'_i = f_{p'_i}(\vec{x}_1, \dots, \vec{x}_m)$ for $i \in [m]$. Hence, to perfectly simulate each of them we need (and only need) the input shares which are involved in each $f_{p'_i}(\vec{x}_1, \dots, \vec{x}_m)$, namely the inputs shares indexed by $(I_1, \dots, I_\ell) := \text{shares}(p'_1, \dots, p'_m)$. Using the input shares $(\vec{x}_1|_{I_1}, \dots, \vec{x}_\ell|_{I_\ell})$ we can perfectly simulate $(p'_1, \dots, p'_m)$ and thus perfectly simulate all algebraic expressions in $\vec{P}'$, from which we get a perfect simulation of the probes in $\vec{P}$.

Lemma 1 actually provides a way to evaluate the function SIS in the case of LR-gadgets. Note that the set of probes $\vec{P}$ in the lemma must be defined as the union of $\vec{P}$ and $\vec{y}|_O$ in an evaluation of $\text{SIS}_G(\vec{P}, O)$ (while used to define security notions, SIS is based on two arguments to differentiate probes on internal wires and probes on output shares whereas this distinction is not used in the evaluation process of Lemma 1). According to the above lemma, an evaluation of SIS simply consists of a row reduction on the matrix of the random dependencies (S), after which the function $\text{shares}(\cdot)$ is used on the obtained expressions without random values (i.e. $(p'_1, \dots, p'_\ell)$ in the lemma). The output of SIS is then exactly the output of $\text{shares}(\cdot)$, which is the set of input shares necessary for a perfect simulation of all the probes. We show in Section 4 how this technique is efficiently implemented in our verification tool.

3.2 Characterization of Gadgets with Non-Linear Randomness

In this section, we extend the algebraic characterization for LR-gadgets of Section 3.1 to NLR-gadgets, i.e. gadgets performing non-linear operations on input shares mixed with randomness. An NLR-gadget is an ℓ -to- m gadget $G : (\vec{x}_1, \dots, \vec{x}_\ell) \mapsto (\vec{y}_1, \dots, \vec{y}_m)$ with the output of the form:

$$(\vec{y}_1, \dots, \vec{y}_m) := R_{\ell+1}(F(R_1(\vec{x}_1, \vec{r}_1), \dots, R_\ell(\vec{x}_\ell, \vec{r}_\ell)), \vec{r}_{\ell+1})$$

where F is any arithmetic circuit, the R_i are linear arithmetic circuits and the $\vec{r}_i$ are vectors of random values uniformly drawn from $\mathbb{K}^{\rho_i}$. We further assume that F computes a homogeneous multi-linear form, namely $F(\vec{z}_1, \dots, \vec{z}_\ell)$ is a sum of degree- ℓ monomials, each of which being a product containing exactly one coordinate from each $\vec{z}_i$.

For the sake of clarity, we describe the verification method for the particular case of 2-input gadgets; the extension to ℓ inputs is straightforward. We thus present NLR-gadgets as 2-to- m gadgets $G : (\vec{x}_1, \vec{x}_2) \mapsto (\vec{y}_1, \dots, \vec{y}_m)$ with the output of the form:

$$(\vec{y}_1, \dots, \vec{y}_m) := R_3(F(R_1(\vec{x}_1, \vec{r}_1), R_2(\vec{x}_2, \vec{r}_2)), \vec{r}_3)$$

This characterization notably covers a wide majority (if not the totality) of multiplication gadgets. It covers in particular multiplication gadgets which first start by refreshing one of (resp. each of) their inputs before performing sharewise products that are finally recombined into the output sharing (with additional randomness). Such multiplication gadgets have been recently described in [31,13,16,21].

Any probe on such an NLR-gadget is either a probe on the inner circuits $R_i(\vec{x}_i, \vec{r}_i)$ and is of the form:

$$p = \vec{x}_i^T \cdot \vec{w}_p + \vec{r}_i^T \cdot \vec{s}_p \quad (6)$$

for $i \in \{1, 2\}$ (since the R_i are linear arithmetic circuits) with $\vec{w}_p \in \mathbb{K}^n$, $\vec{s}_p \in \mathbb{K}^{\rho_i}$, or is a probe on the outer circuits and is of the form:

$$p = f_p(\vec{z}_1, \vec{z}_2) + \vec{r}_3^T \cdot \vec{s}_p \quad (7)$$

where $\vec{z}_i := R_i(\vec{x}_i, \vec{r}_i)$ for $i \in \{1, 2\}$ with $\vec{s}_p \in \mathbb{K}^{\rho_3}$, and for some arithmetic function $f_p : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}$. We show hereafter that we can still obtain an efficient and complete evaluation of SIS for those gadgets, which in turn yields an efficient verification of the considered security notions.

The verification technique for NLR-gadgets essentially consists in several iterations of the verification process for LR-gadgets used in Lemma 1. The steps of the technique are as follows. Suppose that we have a tuple of probes $\vec{P} = (p_1, \dots, p_k, p_{k+1}, \dots, p_d)$ where $(p_1, \dots, p_k)$ are all of the form (7) while $(p_{k+1}, \dots, p_d)$ are all of the form (6).

1. First, we apply the Gaussian elimination technique of Section 3.1 on the probes $(p_1, \dots, p_k)$ with respect to the vector of randoms $\vec{r}_3$. This is possible since all of these probes respect the form (5) w.r.t. inputs $(\vec{z}_1, \vec{z}_2)$ and randomness $\vec{r}_3$. Specifically, let $S_3 := (\vec{s}_{p_1} \mid \vec{s}_{p_2} \mid \dots \mid \vec{s}_{p_k})$, with $\vec{s}_{p_i}$ defined from (7), and let N_3 the permutation matrix such that $S'_3 = N_3 \cdot S_3$ is the row reduced form of S_3 (see Lemma 1). From this, we get a new derived tuple $\vec{P}' := N_3 \cdot \vec{P} = (p'_1, \dots, p'_m, p'_{m+1}, \dots, p'_k)$ and we know from Lemma 1 that each of the expression in $(p'_{m+1}, \dots, p'_k)$ can be perfectly simulated by simply generating $k - m$ uniform random values. Thus, we end up with $(p'_1, \dots, p'_m)$, which we need to perfectly simulate, and where each of the p'_i is of the form $f_{p'_i}(\vec{z}_1, \vec{z}_2)$ with no random values from $\vec{r}_3$, along with the remaining probes $(p_{k+1}, \dots, p_d)$. We then construct the new tuple to simulate $\vec{P}'' = (p'_1, \dots, p'_m, p_{k+1}, \dots, p_d)$, which we rewrite as $\vec{P}'' = (p''_1, \dots, p''_{m+d-k})$. Thus, in order to perfectly simulate the tuple of probes $\vec{P}$, we need to perfectly simulate the probes in $\vec{P}''$.

We stress at this stage that each algebraic expression p''_i in $\vec{P}''$ is either of the form $p''_i = f_{p''_i}(\vec{z}_1, \vec{z}_2)$ with $f_{p''_i}$ a homogeneous bilinear form (this is of the first m coordinates resulting from Gaussian elimination) or of the form (6) (*i.e.* the probes on R_1 or R_2 that are not affected by the previous Gaussian elimination since they do not contain any randoms from $\vec{r}_3$).

2. For each p''_i in $\vec{P}''$ of the form $p''_i = f_{p''_i}(\vec{z}_1, \vec{z}_2)$, we factor its algebraic expression with respect to the vector of values $(\vec{x}_1 \parallel \vec{r}_1)$. In other terms, we rewrite each p''_i as

$$p''_i = (\vec{x}_1 \parallel \vec{r}_1)^T \cdot \vec{h}_{p''_i} \quad (8)$$

where $\vec{h}_{p_i''}$ is a tuple of $n + \rho_1$ algebraic expressions of the form (6) w.r.t. $(\vec{x}_2, \vec{r}_2)$. We then construct a new tuple $\vec{P}_2 := (\vec{h}_{p_1''} \parallel \dots \parallel \vec{h}_{p_m''})$ to which we append all the expressions p_i'' of the form (6) w.r.t. $(\vec{x}_2, \vec{r}_2)$ (i.e. probes from R_2).

3. We perform the same procedure as in the last step but this time factoring each p_i'' in $\vec{P}''$ of the form $p_i'' = f_{p_i''}(\vec{z}_1, \vec{z}_2)$ with respect to $(\vec{x}_2 \parallel \vec{r}_2)$, rewriting each p_i'' as

$$p_i'' = (\vec{x}_2 \parallel \vec{r}_2)^T \cdot \vec{g}_{p_i''} \quad (9)$$

From those expressions we define a new tuple $\vec{P}_1 := (\vec{g}_{p_1''} \parallel \dots \parallel \vec{g}_{p_m''})$ where the coordinates of the $\vec{g}_{p_i''}$'s are of the form (6) w.r.t. $(\vec{x}_1, \vec{r}_1)$, to which we append all the expressions p_i'' of the form (6) w.r.t. $(\vec{x}_1, \vec{r}_1)$ (i.e. probes from R_1).

4. Recall from the first step that perfectly simulating $\vec{P}$ amounts to perfectly simulating $\vec{P}''$. We will prove later in this section that the input shares from $\vec{x}_1$ and $\vec{x}_2$ that are necessary and sufficient to produce a perfect simulation of $\vec{P}''$ are the same as the ones for a perfect simulation of $\vec{P}_1, \vec{P}_2$ constructed in the last two steps. Observe that all probes in $\vec{P}_1, \vec{P}_2$ respect the form (6), which is a special case of (5). Hence, we separately apply the Gaussian elimination technique of Lemma 1 on $\vec{P}_1$ with respect to $(\vec{x}_1, \vec{r}_1)$, and on $\vec{P}_2$ with respect to $(\vec{x}_2, \vec{r}_2)$. This provides us with the sets of input shares I_1 on $\vec{x}_1$ and I_2 on $\vec{x}_2$ that are respectively necessary and sufficient to produce a perfect simulation of the expressions in $\vec{P}_1$ and $\vec{P}_2$. These sets are therefore output as the necessary and sufficient sets of input shares for a perfect simulation of $\vec{P}$.

We state in the following lemma that the above verification method is complete (the proof is in appendix).

Lemma 2. *Let G be a 2-input n -share NLR-gadget. Let $\vec{P} = (p_1, \dots, p_d)$ be a tuple of probes on G . Let $\vec{P}_1, \vec{P}_2$ be the tuples of linear expressions w.r.t. $(\vec{x}_1, \vec{r}_1)$ and $(\vec{x}_2, \vec{r}_2)$ obtained by applying the above method. The sets I_1, I_2 obtained by applying the method of Lemma 1 on $\vec{P}_1$ with respect to $(\vec{x}_1, \vec{r}_1)$ and separately on $\vec{P}_2$ with respect to $(\vec{x}_2, \vec{r}_2)$ are the sets of input shares necessary and sufficient to simulate $\vec{P}$.*

The verification method introduced above actually describes the procedure of the function SIS in the case of NLR-gadgets to determine the simulatability of a set of probes on such gadgets. We show in Section 4 how this technique is implemented in **IronMask**. We now present a concrete example of SIS execution on a set of probes on an NLR-gadget.

Example: Let us consider the following 2-share multiplication gadget (with inputs a and b , and output e) while taking $\mathbb{K} = \mathbb{F}_2$:

$$\begin{aligned} c_1 &= a_1 + r_a, & c_2 &= a_2 + r_a \\ d_1 &= b_1 + r_b, & d_2 &= b_2 + r_b \\ e_1 &= (c_1 * d_1 + r) + c_1 * d_2 \\ e_2 &= (c_2 * d_1 + r) + c_2 * d_2 \end{aligned}$$

The above gadget is an example of NLR-gadgets, and uses 3 random values: r_a is used to refresh the input sharing a , r_b is used to refresh the input sharing b , and r is used during the compression of the products into the output sharing e . The non-linear random values are r_a and r_b with respect

to e .

Suppose that we would like to verify one of the security properties defined in Section 2 using SIS. To do this, we need to be able to determine for each set of probes (formed of intermediate values and/or output shares) on the gadget, the exact set of input shares necessary and sufficient for a perfect simulation of all of the probes in the set. Let us consider for instance the following set of 2 probes on the gadget:

$$P = \{p_1 = c_1 * d_1 + r, \quad p_2 = c_2 * d_1 + r\}$$

We need to determine the set of input shares of a and b necessary to perfectly simulate probes in P . SIS will be executed in four steps as described earlier.

Step 1: get rid of the random values that are additive in the compression step (which are not additive to the shares of a and b), in this case it is the unique random value r . Using the Gaussian elimination technique, we construct a new set :

$$P' = \{p_1 + p_2 = c_1 * d_1 + c_2 * d_1, \quad p_2 = c_2 * d_1 + r\}$$

Since r only appears in p_2 , this probe can be perfectly simulated by a uniform random value. Next we need to consider the simulation of the new set

$$P'' = \{c_1 * d_1 + c_2 * d_1\} = \{(a_1 + r_a) * (b_1 + r_b) + (a_2 + r_a) * (b_1 + r_b)\}$$

Step 2: factor the expressions in P'' with respect to the elementary variables of shares of a and random values which are additive to the shares of a and the constant term 1, in this case the variables $(a_1, a_2, r_a, 1)$. Since there is a single expression in P'' , we can rewrite it as:

$$P'' = \{a_1 * (b_1 + r_b) + a_2 * (b_1 + r_b) + r_a * (b_1 + r_b + b_1 + r_b) + 1 * (0)\}$$

from which we construct the new set of the expressions multiplying $(a_1, a_2, r_a, 1)$

$$P_2 = \{b_1 + r_b, b_1 + r_b, 0, 0\}$$

Step 3: do the same thing with respect to $(b_1, b_2, r_b, 1)$:

$$P'' = \{b_1 * (a_1 + r_a + a_2 + r_a) + b_2 * (0) + r_b * (a_1 + r_a + a_2 + r_a) + 1 * (0)\}$$

from which we construct

$$P_1 = \{a_1 + a_2, 0, a_1 + a_2, 0\}$$

Step 4: determine the input shares of a necessary to simulate the expressions in P_1 and the shares of b necessary to simulate the expressions in P_2 .

- for input a , we trivially need both input shares (a_1, a_2) to perfectly simulate expressions in P_1 .
- for input b , we apply one step of Gaussian elimination with respect to r_b , to obtain the new set $P'_2 = \{b_1 + r_b, 0, 0, 0\}$. We can see that we can perfectly simulate the single non-zero expression with a uniform random value. Thus, in this case, no shares of b are necessary to perfectly simulate P'_2 and hence also P_2 .

Hence, to perfectly simulate P_1 and P_2 , we need both input shares of a and no shares of b . Thanks to Lemma 2, we can conclude that P can be perfectly simulated using both shares of a and no shares of b .

4 Efficient Verification

In this section, we introduce **IronMask**, a new tool that we developed to check probing and random probing security properties using the algorithms presented in Section 3. The implementation of **IronMask** considers currently a finite field $\mathbb{K}$ of characteristic 2, it can be easily extended in the future to any finite field since the verification methods introduced in the previous sections work in any finite field $\mathbb{K}$. **IronMask** is written in C, and the only external libraries it depends on are the GNU Multiple Precision Arithmetic Library (GMP) and the POSIX Threads (pthreads) library.

4.1 Data Representation

<pre> #shares 2 #in a b #randoms r0 #out c m0 = a0 * b1 t0 = r0 + m0 m1 = a1 * b0 t1 = t0 + m1 m2 = a0 * b0 c0 = m2 + r0 m3 = a1 * b1 c1 = m3 + t1 </pre>	<pre> #shares 3 #in a #randoms r0 r1 r2 #out d d0 = a0 + r0 d0 = d0 + r1 d1 = a1 + r0 d1 = d1 + r2 d2 = a2 + r1 d2 = d2 + r2 </pre>
(a) 2-share ISW multiplication	(b) 3-share refresh

Fig. 1: Masking gadgets written in **IronMask**'s syntax

IronMask takes as input gadgets written in a simple syntax to describe circuits, borrowed from VRAPS [13]: a gadget is a list of assignments of additions or multiplications into variables, alongside directives to specify the number of shares, the inputs, the outputs and the randoms. Figure 1 illustrates our input syntax on a 2-shares ISW multiplication (Figure 1a) and a 3-share refresh gadget (Figure 1b). In Figure 1a, the variables **a0/b0** (resp. **c0**) and **a1/b1** (resp. **c1**) are the 1st and 2nd shares of the input **a/b** (resp. output **d**). Similarly to **maskVerif**, the syntax **![expr]** can be used to stop the propagation of glitches in the robust probing model. For instance, **tmp = a0*b0** could be replaced by **tmp = ![a0*b0]**, in which case **tmp** would leak **a0*b0** instead of leaking **a0** and **b0** separately.

Internally, **IronMask** represents each wire of the gadget as an array of integers composed of three parts. The first ℓ parts correspond to linear dependencies on the inputs of the gadget: if the k^{th} bit of the n^{th} element is set to 1, then the wire depends linearly on the k^{th} share of the n^{th} input. The second part is a bitvector, where the k^{th} bit set to 1 indicates a linear dependency on the k^{th} random of the gadget. Finally, the third part is a bitvector as well, where the k^{th} bit set to 1 indicates a linear dependency on the k^{th} quadratic monomial appearing in the symbolic expressions

of the gadget wires. For instance, the internal representation of the wires `a0`, `a1`, `r0`, `m3`, `t1` and `c1` of Figure 1a are as follows:

	inputs	randoms	mults
<code>a0</code> :	[1, 0,	0,	0,0,0,0]
<code>a1</code> :	[2, 0,	0,	0,0,0,0]
<code>r0</code> :	[0, 0,	1,	0,0,0,0]
<code>m3</code> :	[0, 0,	0,	0,0,0,1]
<code>t1</code> :	[0, 0,	1,	1,1,0,0]
<code>c0</code> :	[0, 0,	1,	0,0,1,0]

With an additional data structure storing the operands of each multiplication:

0: <code>a0 * b1</code>	1: <code>a1 * b0</code>
2: <code>a0 * b0</code>	3: <code>a1 * b1</code>

Using this internal representation enables efficient operations down the line: the linear dependencies of a wire on the input shares are accessible with a single operation, the number of such input shares is efficiently obtained by counting the number of bits to one in its first element (or first two elements for 2-input gadgets), and `xoring` two wires, which is one of the basic operations of our Gaussian elimination, can be easily done by `xoring` pointwise the arrays representing them.

To model glitches in the robust probing model, we use the same glitch model as in [3]. Namely, we consider that an expression $a + b$ (resp $a * b$) leaks a and b separately, instead of leaking $a + b$ (resp $a * b$). Registers (usually called *flip-flops*) can be used to stop the propagation of these glitches. In `IronMask`, when taking glitches and transitions into consideration, each wire is represented by an array of arrays instead of a single array, since the leakage of an assignment is the union of the leakages of its right-hand side operands. For instance, the wire `c0` in Figure 1a in the presence of glitches is represented as:

	inputs	randoms	mults
<code>c0</code> :	[[0, 0,	1,	0,0,0,0],
	[1, 0,	0,	0,0,0,0],
	[0, 1,	0,	0,0,0,0]]

If a flip-flop was added to `m2` to stop the propagation of glitches by doing `m2 = ![a0*b0]`, then the robust leakage of `c0` would become:

	inputs	randoms	mults
<code>c0</code> :	[[0, 0,	1,	0,0,0,0],
	[0, 0,	0,	0,0,1,0]]

4.2 Basic Verification

In this section, we present the procedures implemented in `IronMask` for the verification of probing and random probing properties. Recall that in Section 2, we give definitions of all the security properties based on a single building block `SIS`: a primitive that, given a set of probes (internal probes and output probes), determines the input shares necessary for a perfect simulation of these probes. Thus, to verify any security property, `IronMask` uses a concrete implementation of the function `SIS` based on the algebraic characterization techniques discussed in Section 3.

Gadgets with linear randomness. For the verification of LR-gadgets introduced in Section 3.1 (*i.e.* gadgets in which all random values are additive), **IronMask** relies on the SIS_LR procedure (Algorithm 1). This procedure is a direct application of the result presented in Lemma 1. We recall that Algorithm 1 in **IronMask** considers currently a finite field of characteristic 2.

Algorithm 1 SIS_LR returns the input shares that are leaked by the tuple $\vec{P}$ with expressions of the form (5), assuming ℓ input sharings

```

1: procedure GAUSSELIMINATION( $\vec{P}$ )
2:   for each probe  $p_i$  of  $\vec{P}$  do
3:     if  $p_i$  contains at least one random variable then
4:        $r \leftarrow$  choose (any) one random variable in  $p_i$ 
5:       for each probe  $p_j$  of  $\vec{P}$  with  $i \neq j$  do
6:         if  $p_j$  contains  $r$  then
7:            $p_j \leftarrow p_i + p_j$ 
8:        $p_i \leftarrow r$ 
9: procedure SHARES( $\vec{P}$ )
10:   $I_1 \leftarrow \emptyset, \dots, I_\ell \leftarrow \emptyset$ 
11:  for each probe  $p_i$  of  $\vec{P}$  do
12:    Add all input shares in  $p_i$  of each input  $j$  to  $I_j$ 
13:  return  $I_1, \dots, I_\ell$ 
14: procedure SIS_LR( $\vec{P}$ )
15:   $\vec{P}' \leftarrow$  GAUSSELIMINATION( $\vec{P}$ )
16:  return SHARES( $\vec{P}'$ )

```

As in Lemma 1, a Gaussian elimination is first performed on the tuple by the procedure GAUSSELIMINATION, after which each probe of the input tuple is either “replaced” by a random r (as shown on line 8 of the procedure GAUSSELIMINATION), or contains one or more input shares and no random values. In fact, what we mean by replacing the probe p_i by a random value r on line 8 is that after eliminating r from the expressions of all other expressions p_j in the same tuple (loop from line 5 to 7 where the instruction $p_j \leftarrow p_i + p_j$ aims to remove r from p_j in a finite field of characteristic 2), we end up with r only appearing in the expression of p_i and so as explained in the proof of Lemma 1, simulating p_i amounts to generating r uniformly at random without the need for any other variables. We represent this by replacing the expression of p_i by the single random value r . Then, the shares leaked by the input tuple can be found on the probes that do not contain any randoms using the procedure SHARES. The latter actually corresponds to an implementation of the function `shares(.)` used in Lemma 1.

Gadgets with non-linear randomness. For NLR-gadgets (*i.e.* gadgets performing non-linear operations on input shares mixed with randomness), **IronMask** uses the SIS_NLR procedure (Algorithm 2), which implements the four steps described in Section 3.2. As mentioned in Section 3.2, SIS_NLR currently only supports gadgets with two input sharings, but can be extended in the future to ℓ input sharings.

First, SIS_NLR performs Gaussian elimination with respect to the vector of output randoms (*i.e.* $\vec{r}_3$), using a modified version of GAUSSELIMINATION that takes as inputs the randoms to use for the elimination. This corresponds to step 1 of Section 3.2. The modified probes in $\vec{P}'$ do not contain any more random values of $\vec{r}_3$. Next, two new tuple of probes $\vec{P}_1, \vec{P}_2$ are constructed

Algorithm 2 SIS_NLR returns the input shares that are leaked by the tuple P in an NLR-gadget refreshing its output with the randoms $\vec{r}_3$ (c.f. sec. 3.2), assuming 2 input sharings

```

procedure SIS_NLR( $\vec{P}, \vec{r}_3$ )
   $\vec{P}' \leftarrow \text{GAUSSELMINATION}(\vec{P}, \vec{r}_3)$ 
   $\vec{P}_1 \leftarrow ()$ ,  $\vec{P}_2 \leftarrow ()$ 
  for each probe  $p_i$  in  $\vec{P}'$  do
    if  $p_i$  contains no randoms of  $\vec{r}_3$  then
       $(\vec{P}'_1, \vec{P}'_2) \leftarrow \text{FACTANDEXTRACT}(p_i)$ 
       $(\vec{P}_1, \vec{P}_2) \leftarrow (\vec{P}_1 || \vec{P}'_1, \vec{P}_2 || \vec{P}'_2)$ 
   $I_1 \leftarrow \text{SIS\_LR}(\vec{P}_1)$ ,  $I_2 \leftarrow \text{SIS\_LR}(\vec{P}_2)$ 
  return  $I_1, I_2$ 

```

from the probes in $\vec{P}'$, using the FACTANDEXTRACT procedure, which corresponds to the factoring technique discussed in steps 2 – 3 of Section 3.2. The pseudo-code of this function is left out for conciseness. We thus get two tuples P_1 and P_2 containing input shares, randoms and refreshed input shares from each input. Since those variables are linear, we can use the initial SIS_LR procedure to extract the input shares that they leak.

Verification of security properties. Checking any probing or random probing property (e.g. NI, SNI, RPC, RPE, ...) consists in enumerating tuples of probes, using SIS_LR or SIS_NLR to get the input shares that they leak (we abbreviate with SIS and suppose that we make a call to the correct algorithm for LR-gadgets and NLR-gadgets), and take some action in consequence (see Section 2.2). In the following, we shall call a t -failure tuple (or simply a failure tuple when t is not made explicit) any tuple of probes that leaks more than t input shares of one or more input sharings (i.e. for which SIS outputs a set or more of cardinality strictly greater than t).

For instance, to verify if an n -share gadget is t -NI, we enumerate all tuples of size t , and make sure that none of them is a t -failure tuple (Algorithm 3). This corresponds to the first row of Table 1. Or, to verify the (p, ε) -RPS* of an n -share gadget G in the random probing model (first row of Table 2), we need to compute the coefficients c_i from equation (3) of the failure probability function $f(p) = \varepsilon$ as explained in Section 2.2. This corresponds to enumerating all the tuples of probes (excluding the output wires) of size 1 to a threshold $c_{\max}$ and count how many leak more than t -shares (after $c_{\max}$, upper and lower bounds on $f(p)$ are obtained, we refer the readers to [13] for more details on the process and the threshold). This corresponds to the procedure depicted in Algorithm 4.

Algorithm 3 IS_T_NI returns true if G is t -NI and false otherwise, assuming G has ℓ input sharings

```

procedure IS_T_NI( $G, t$ )
  for each tuple  $\vec{P}$  of size  $t$  in  $G$  do
     $I_1, \dots, I_\ell \leftarrow \text{SIS}(\vec{P})$ 
    if  $|I_1| > t$  or ... or  $|I_\ell| > t$  then
      return false
  return true

```

Enumerating all tuples becomes impractical as soon as gadgets start growing larger than a few hundred variables, since the number of tuples of size k in a gadget containing s variables is $\binom{s}{k}$.

Algorithm 4 GETCOEFFSRPS^{*} returns an array of $c_{\max}$ cells where the k^{th} index contains the number of failure tuples of k probes on n -share gadget G with ℓ input sharings

```

procedure GETCOEFFSRPS*( $G, c_{\max}$ )
   $coeffs \leftarrow [0, \dots, 0]$  of size  $c_{\max}$ 
  for  $k = 1$  to  $c_{\max}$  do
    for each tuple  $\vec{P}$  of  $k$  probes on  $G$  do
       $I_1, \dots, I_\ell \leftarrow \text{SIS}(\vec{P})$ 
      if  $|I_1| > n - 1$  or  $\dots$  or  $|I_\ell| > n - 1$  then
         $coeffs[k] += 1$ 
  return  $coeffs$ 

```

For instance, checking that a 9-share masked ISW multiplication containing 279 variables is 8-NI requires enumerating $\binom{279}{8} \approx 8 \times 10^{14}$ tuples, which is not far from being out of reach for modern computers.

The rest of the section is organized as follows. In Section 4.3, we address dimension reduction techniques proposed in [11,19] to reduce the search space of the enumerated tuples. In Section 4.4, we present some optimizations of our implementations that make verification faster by reducing the cost of SIS_LR (since the latter is also a building block for SIS_NLR) and parallelizing our procedures. Finally, in Section 4.5, we introduce a constructive algorithm to generate failures without having to enumerate all tuples in the case of linear gadgets.

4.3 Dimension Reduction

Checking any probing or random probing property requires enumerating many tuples. For instance, for a gadget G made of s variables, $\binom{s}{t}$ tuples need to be checked to assess whether G is t -NI or not. To reduce the number of tuples that have to be considered, we remove some variables from the search. First, as proposed in [11] and further explained in [19], elementary deterministic probes can be removed when checking any probing or random probing property. Then, when checking for probing properties only, we use the “reduced sets” optimization proposed in [19], which consists in eliminating some “less powerful” variables from the search. In appendix, we recall the principle of those two optimizations, and show how to make the first one work in the random probing model, and why the second one cannot be used in this model. Note that the dimension reduction technique is proved to be sound in [19], which means that our verification technique implementing the optimization remains sound.

4.4 Implementation Optimizations

On-the-fly Gaussian Elimination. In order to find all failures of a given size, we enumerate all the tuples of that size, and apply the SIS procedure on each of them. This means that a full Gaussian elimination has to be performed on each tuple. However, we generate the tuples in lexicographic order, which mean that two consecutive tuples only differ only by their last elements, and, in most cases, only by their very last element. For two consecutive tuples, it is thus very likely that most of the Gaussian elimination will be identical. We take advantage of this by implementing our Gaussian elimination on the fly: for each tuple, we only recompute the elimination on the elements that differ from the previous tuple.

Table 3: RPS* performance of our new constructive algorithm against our traditional enumerative one

Gadget	Shares	#wires	Enumerative		Constructive	
			$c_{\max}$	Verif time	$c_{\max}$	Verif time
ISW refresh	8	140	8	5min	8	3sec
					10	6min
ISW add	7	224	7	18min	8	2sec
					9	2min
$n \log n$ refresh	8	100	9	1min	9	2sec
					11	2min
ISW mult refreshed	6	297	6	2min	6	12min
			7	38min		

The cost of the Gaussian elimination for a single tuple of k elements of a gadget containing s inputs and randoms is $\mathcal{O}(sk^2)$. Performing the elimination on-the-fly brings the amortized complexity down to $\mathcal{O}(sk)$.

A similar, slightly more efficient technique has been used by [19] to speed up their implementation. They used a revolving-door algorithm to generate the tuples, so that each consecutive tuple differs by exactly one element, which allowed them to amortize the cost of their analysis. However, we cannot use this revolving-door algorithm because when changing the i^{th} element of a tuple, the Gaussian elimination needs to be recomputed from this i^{th} element up to the end of the tuple.

Parallelization. Recall that we generate the tuples in lexicographic order, which admits an efficient *unranking* algorithm. This means that we can easily compute what the j^{th} tuple of size k is, for any j and any k . Multi-threading the verification of n tuples is thus trivial: to run l threads in parallel, the j^{th} thread starts with the $\lfloor j \times n/l^{\text{th}} \rfloor$ tuple, and verifies the next $\lfloor n/l \rfloor$ tuples.

Our implementation is multi-threaded in this fashion using POSIX threads, provided by the `pthread` library. In order to be transparent from the properties’ point of view (*e.g.*, from Algorithms 3 and 4), the multi-threading is done inside SIS. To this end, we use a few mutexes, which incur an overhead in the random probing model: the more failures a gadget contains, the less of a speedup multi-threading offers. Although it would not be hard to implement multi-threading on the properties’ side rather than in SIS, we opted for readability and maintainability of the code, at the expense of a bit of performance.

4.5 Constructive Approach

The enumerative approach of Section 4.2 generates a lot of tuples that are trivial non-failures because they do not contain enough shares to be failures, or their shares are masked by random variables. To overcome this issue, we designed a constructive algorithm to only generate potential failures. We give a detailed description of this algorithm in Appendix D.

This constructive algorithm is faster than the traditional enumerative algorithm of Section 4.2 when checking $(n - 1)$ -NI and RPS* properties for linear gadgets. Table 3 shows the exact performance improvements when checking the RPS* property on some common linear gadgets (ISW refresh [32], $n \log n$ refresh [9], and an “ISW addition” made of a share-wise addition preceded by an ISW refresh of each input), and on an ISW multiplication with a circular refresh on one of the

inputs. On linear gadgets, the constructive algorithm can go about 2 coefficients further than the enumerative one within the same time, thus producing much more precise results.

Furthermore, the constructive algorithm enables the verification of larger, previously out of reach, gadgets. For instance, a 9-share ISW addition contains 243 variables, and thus contains $\binom{243}{9} \approx 7 * 10^{15}$ tuples of size 9 (for 8-RPS*), which is clearly beyond the capabilities of the enumerative algorithm. Yet, our constructive algorithm is able to generate all of its failures of size 9 in 7 minutes.

However, on multiplication gadgets, the constructive algorithm is slower than the enumerative one. Table 3 illustrates this on an ISW multiplication with an ISW refresh on one of the inputs. Additionally, the constructive algorithm does not perform well in terms of performance when checking t -NI or t -RPS* with $t < n - 1$, as well as SNI, RPC and RPE. We explain in Appendix D why this is the case.

5 Evaluation

To showcase **IronMask**, we start in Section 5.1 by providing new bounds for the maximum RPE leakage probability tolerated by some common gadgets (in the random probing model). Then, we compare the scope (Section 5.2) and performance (Section 5.2) of **IronMask** and existing verification tools: **VRAPS** and **STRAPS** in the random probing model, and **maskVerif**, **matverif** and **SILVER** in the probing model. The description files of the gadgets tested in the following sections are publicly available on **IronMask**'s GitHub repository.

5.1 New Random Probing Expandability Results

So far, **VRAPS** [13] was the only tool verifying the (t, p, ε) -RPE property. **IronMask** is several orders of magnitude faster than **VRAPS**, in addition to being complete (**IronMask** avoids failure false positives *i.e.* detected failure tuples which are not really failures, unlike **VRAPS**), allowing us to compute more precise bounds for the coefficient of the failure function $f(p) = \varepsilon$ (*c.f.* Section 2.2) and hence more precise bounds on the tolerated leakage probability. In particular, we consider the ISW multiplication and refresh [32], the $n \log n$ refresh [9], the circular refresh [7], as well as the addition, copy, and multiplication from [16, Section 6.2]. Additionally, we consider addition (resp. copy) gadgets obtained by doing an ISW or $n \log n$ refresh on one of the inputs followed by a simple addition (resp. copy). Finally, we also evaluate a double-SNI multiplication [31] made of an ISW multiplication where one of the inputs is refreshed using a n circular refreshes [25] (with n shares).

The result are shown in Table 4. For the t parameters, we used $t = \lfloor (n - 1)/2 \rfloor$ (with n shares), as recommended by [16]. For large gadgets, we cannot compute precisely the maximum leakage probability tolerated in reasonable time. Instead, like [13], we compute all failures up to a given size c_{max} , which allows us to obtain upper and lower bounds for the leakage probability.

For ISW multiplication, our results improve previous results from [13] (obtained with **VRAPS**) in two ways: by increasing the value c_{max} of the verification we obtain tighter bounds on the failure event function $f(p)$ and thus tighter intervals (and even exact values in some cases) for the tolerated leakage probability. Plus, thanks to the completeness of the verification of **IronMask** by avoiding failure false positives (unlike **VRAPS**), we obtain better values for the estimated tolerated leakage probability (by better we mean higher probability values). For example, our results show that the (exact) tolerated leakage probability of the 6-share ISW multiplication is 2^{-12} instead of the 2^{-13} lower bound of [13]. We obtain similar improvements for the 5-share gadgets of [16] which are today

Table 4: Maximum t -RPE leakage probabilities tolerated by some common masking gadgets

Gadget	Shares	t	Ampl. order	c _{max}	#wires	log ₂ maximum tolerated proba.	Verif. time	
Linear Randomness								
ISW [32]	mult	5	2	3/2	6	180	-10.54	24min
		6	2	3/2	5	267	-12.00	13min
		7	3	2	5	371	[-10.45, -8.73]	28min
	refresh	5	2	3	10	50	-4.28	2min
		6	2	3	8	75	[-4.81, -4.61]	5min
		7	3	4	7	105	[-5.50,-4.01]	21min
	add	5	2	3	7	110	[-6.48,-4.70]	11min
		6	2	3	6	162	[-7.81,-5.03]	17min
		7	3	4	6	224	[-8.47,-4.15]	3h
	copy	5	2	3	6	105	[-5.92,-5.54]	12min
		6	2	3	5	156	[-6.92,-5.93]	24min
		7	3	4	4	217	[-8.02,-3.87]	33min
n log n [9]	refresh	4	1	2	30	30	-5.27	1sec
		8	3	4	7	100	[-5.42,-4.36]	18min
	add	4	1	2	8	68	-5.40	4min
		8	3	4	6	216	[-8.40,-4.40]	4h
	copy	4	1	2	6	64	-6.96	27sec
		8	3	4	4	208	[-7.94,-4.25]	55min
circular refresh [7]	5	2	3	25	25	-4.84	1sec	
	10	4	3	8	50	-5.21	1min	
[16]	add	5	2	3	9	55	[4.67,-4.42]	10min
	copy	5	2	3	6	60	-6.17	41sec
Non-linear Randomness								
Double-SNI	4	1	2	5	190	-9.85	5min	
ISW	5	2	5/2	5	305	[-10.01,-8.09]	31min	
[16] mult	5	2	3	6	405	[-9.67,-7.66]	31h	

the gadgets giving the best asymptotic complexity of $\mathcal{O}(\kappa^{3.23})$ for the expansion strategy with a constant leakage probability and for a target random probing security of $2^{-\kappa}$. Our results improve the lower bound on the tolerated leakage probability for those gadgets from 2^{-12} to $2^{-9.67}$. For all the other gadgets in the table, we report the first verification results of the RPE property.

5.2 Comparison with State-Of-The-Art Tools

We compare **IronMask** to six carefully chosen state-of-the-art tools: **maskVerif** [5,6,3] (and its extension **scVerif** [8]), **matverif** [19], **SILVER** [33], **VRAPS** [13], and **STRAPS** [20], with which our new tool **IronMask** shares the following features:

- does not rely on any gadget’s structure (unlike *e.g.*, **maskComp** [6], **tightPROVE** [15], **Tornado** [14]),
- verifies probing or random probing-like security notions.

We discuss the properties that are concretely verified, and provide some benchmarks to highlight the main differences with **IronMask**.

Scope. Introduced in 2015 [5] and then extended multiple times ([6,3]), **maskVerif** is the very first tool able to verify reasonable higher-order masking schemes. Based on a symbolic representation of

leakage, it integrates the language-based verification of (robust) probing security and (S)NI notions with or without leakage on registers transitions. One step further, the latest extension of **maskVerif**, referred to as **scVerif** [20], captures even more hardware side effects, potentially configurable by the user [8]. Compared to our proposal, **maskVerif** includes tricks to verify bigger circuits (*e.g.*, s-boxes, block encryption scheme) but fails to provide a complete verification as soon as the randomness is not linear (*i.e.*, failure false positives may be produced).

In the same vein, **matverif** [19] targets the same properties as **maskVerif**. It features a new method to obtain a complete verification (*i.e.*, without any failure false positive) for specific circuits (*e.g.*, ISW multiplications) and significantly improve its performance thanks to dimension-reduction strategies. In terms of supported gadgets, **matverif** is more limited than **maskVerif** and **IronMask**, as it does not support gadgets with non-linear randomness at all. Unlike our proposal and similarly to **maskVerif**, **matverif** focuses only on the verification of probing-like properties.

Following a different strategy, **SILVER** [33] was built to verify the physical security of hardware designs. It takes as input either a Verilog implementation or an instruction list and checks the probing, (S/PI)NI notions in the standard and robust models, as well as the uniformity of some output sharing. On the one hand, it outperforms the capacities of **maskVerif** by offering a complete verification based on a symbolic and exhaustive analysis of probability distributions and statistical independence of joint distributions. On the other hand, its verification is significantly slower than that of **maskVerif**.

Introduced in 2020, **VRAPS** is the first tool to verify random-probing-like properties [13] (to the best of our knowledge). Written in Python and SageMath, it was built to evaluate the RPE security of some base gadgets, in order to assess the global security of the expanding compiler of [13]. Specifically, **VRAPS** detects all the leaking tuples within an implementation with respect to the RPS*, RPE1, RPE2 and RPC security properties introduced in Section 2. Nevertheless, it suffers from low performance and, unlike **IronMask**, can generate failure false positives for both gadgets with linear and non-linear randomness. **VRAPS** supports more gadgets than **IronMask** which is limited to LR-gadgets and NLR-gadgets. Nevertheless, to the best of our knowledge, all the masking gadgets in the literature fit the latter representations. While **VRAPS** can additionally (directly) verify bigger gadgets (*i.e.*, composition of atomic gadgets), in practice, the performance and the completeness would be very low. In addition, the verification of atomic gadgets using **IronMask** already makes it possible to obtain secure global circuits since once individually verified (for probing or random probing properties), they can be safely composed [6,13].

Finally, **STRAPS** is a very recent tool designed to verify random probing-like properties [20]. In particular, it was built to compute the distribution of a gadget’s input sets of shares with respect to the output observations and the leakage probability of each internal wire. In its deterministic mode, it relies on **maskVerif** as a basic primitive. One step further, it integrates a probabilistic mode, based on Monte-Carlo methods, which significantly improves the performance by avoiding a full exploration and limiting the analysis to selected tuples. While the probabilistic mode can allow increased performance and thus more accurate results for random probing properties, it uses a set of rules from **maskVerif** as a building block. These rules by construction do not provide complete verification, which implies that the verification method of **STRAPS** is not complete either.

Table 5 recalls the categories of properties (as in Section 2) that are verified by the aforementioned tools on higher-order masked implementations. It additionally specifies the consideration of hardware effects, *i.e.* glitches (captured in the robust probing model). A green check (✓) means that the row tool verifies the column property. On the contrary, a red cross (✗) means that the

Table 5: Verified security properties on higher-order masked implementations for carefully chosen state-of-art automatic tools.

Tools	probing-like		RP-like	
	soft	robust	soft	robust
maskVerif	✓	✓	✗	✗
scVerif	✓	✓	✗	✗
matverif	✓	✓	✗	✗
SILVER	✓	✓	✗	✗
VRAPS	✓	✗	✓	✗
STRAPS	✗	✗	✓	✗
IronMask	✓	✓	✓	✓

column property is not handled by the row tool. We can see that **IronMask** is the first tool to provide verification for probing-like properties and also for random probing-like properties in the standard model and in presence of glitches (robust model).

Additionally **IronMask** offers a complete verification method for gadgets with linear randomness as well as for most deployed gadgets with non-linear randomness (and in particular all known multiplication gadgets). The only other tool providing complete verification for such gadgets is **SILVER** but this is achieved by an exhaustive approach making its running time quickly prohibitive (see comparison hereafter).

Performance. We evaluate the performance of **IronMask** compared to other state-of-the-art verification tools in both the probing and random probing models.

Probing Model. We compared the time required by **IronMask**, **maskVerif** and **matverif** to check that some commonly-used gadgets are $(n - 1)$ -NI and $(n - 1)$ -SNI (abstracted NI and SNI hereafter for conciseness). In particular, the gadgets we considered are: the ISW multiplication [32], the double-SNI multiplication [31,21] using an ISW multiplication and a circular refresh [25] on one of the inputs, the new NI and SNI multiplications from [19], and the $n \log n$ refresh [9]. The results are presented in Table 6.

We used the multi-threaded version of each tool, setting the maximal number of cores to use to 4, to give a fair chance to **maskVerif**: while **IronMask** and **matverif** can use an arbitrary number of cores, **maskVerif** is limited to 4 cores. We evaluated several masking orders for each gadget in order to highlight the scaling of each tool. To save time, we did not run the verification of gadgets when it would have taken more than a few hours. Finally, to analyze a gadget, **matverif** needs probes description files to be generated using a SageMath script, and the main program to be recompiled. While the incurred additional time was ignored in [19], we take it into account so that time we report reflects the actual time that a user would need to check those gadgets.

Before discussing the results, we recall that the three tools are not functionally equivalent. **maskVerif** can handle any circuit, while **IronMask** is limited to LR-gadgets and NLR-gadgets as characterized in Section 3, and **matverif** only handles LR-gadgets. On the other hand, the verification of **IronMask** is complete for both types of gadgets, while **maskVerif**'s is not. In addition, while **IronMask** is limited to LR-gadgets and NLR-gadgets, to the best of our knowledge, all the masking gadgets in the literature fit the latter representations, and the verification of atomic gadgets already makes it possible to obtain secure global circuits through composition [6,13].

Table 6: Comparison of the performance of **IronMask**, **maskVerif** and **matverif** on higher-order masked gadgets. The multithreaded versions of each tools were used, with the maximum number of threads set to 4. N/A means that a tool cannot check a gadget, whereas – means that a tool was not evaluated on a gadget because we deemed it too slow.

Gadget	Type	Shares	Property	Verification time		
				IronMask	maskVerif	matverif
ISW mult	LR	7	NI SNI	7sec 8sec	1min30sec 3min56sec	24sec 25sec
		8	NI SNI	4min 6sec 5min 15sec	2h 10min 6h 30min	5min 19sec 5min 15sec
		9	NI SNI	2h 22min 3h 7min	– –	2h 3min 1h 58min
ISW mult refreshed	NLR	6	NI SNI	2sec 3sec	3sec 10sec	N/A N/A
		7	NI SNI	3min 41sec 6min	1min 50sec 8min 16sec	N/A N/A
		8	NI SNI	8h 52min 14h 46min	2h 2min 10h 4min	N/A N/A
NI mult [19]	LR	7	NI	1sec	1min50sec	5sec
		8		5sec	2h 10min	9sec
		9		2min50sec	–	40sec
		10		6h 28min	–	1h 40min
SNI mult [19]	LR	7	SNI	1sec	6min	8sec
		8		46sec	6h 26min	17sec
		9		24min	–	4min 37sec
		10		24h	–	1h 54min
refresh $n \log n$	LR	9	NI SNI	1sec 24sec	< 1sec 2sec	1sec 1sec
		10	NI SNI	1sec 16min	<1sec 9sec	10sec 10sec
		11	NI SNI	1sec 7h 50min	<1sec 1min	3min 3min
		12	NI SNI	1sec –	<1sec 5min	3h 35min 1h 52min

Overall, the three tools have similar performance, and allow to analyse gadgets up to similar masking orders, although each tool has its strengths and weaknesses. For instance, **maskVerif** performs better on LR-gadgets, while **matverif** shines on LR multiplications. In the following, we investigate in more details the relative speed of each tool.

To analyse the performance of **IronMask** compared to **matverif**, five main factors need to be taken into account. First, **matverif**’s method to verify a tuple of probes on an LR-gadget has a complexity linear in the tuple’s length, while our **SIS_LR** method has the complexity of a Gaussian elimination which in our implementation is quadratic in the tuple’s length. Second, the dimension reduction performed by **IronMask** is faster than that of **matverif** in most cases, probably because ours is written in C and **matverif**’s in SageMath. Third, our dimension reduction often removes more wires, resulting in fewer tuples to consider. Fourth, **matverif** needs to be recompiled for each gadget. Fifth and last, on linear gadgets (add and copy) and when checking $(n - 1)$ -NI, **IronMask** uses the constructive algorithm (see Section 4.5), which is much faster than any enumerative algorithm.

It can be observed through Table 6 that the scale of the speedups offered by **IronMask** compared to **matverif** and **maskVerif** depends mostly on the structure and nature of the gadgets. We will explain hereafter the reason for this scaling depending on the tested gadgets.

matverif tends to be slower than **IronMask** at smaller masking orders, and at any order when checking $(n - 1)$ -NI on linear gadgets. However, at the highest masking orders, the cost of the dimension reduction and the recompilation of **matverif** become negligible compared to the main enumerative algorithm. **matverif** thus becomes faster than **IronMask**, thanks to its complexity linear in the tuples' length (compared to the quadratic complexity of **IronMask**'s **SIS_LR** primitive).

On the standard ISW multiplication and the multiplication from [19], both **matverif** and **IronMask** outperform **maskVerif** thanks to the dimension reduction they use (see Section 4.3), which is not implemented in **maskVerif**. For instance, the 7-order ISW multiplication contains initially 220 variables, but only 77 remain after the dimension reduction.

On the double-SNI ISW multiplication with a circular refresh on the inputs, the dimension reduction is much less potent. For the 6th-order double-SNI multiplication, 175 probes must thus be taken into account, against 57 for the standard 6th-order ISW multiplication. As a result, **maskVerif** and **IronMask** have similar performance on this gadget.

The $n \log n$ refresh is obviously NI at any order, since at no point are multiple secret shares part of the same probe. **maskVerif** seems to have a special rule to detect that, resulting in a verification of NI that is almost instantaneous. We did not add this special rule in **IronMask**, but our constructive algorithm (presented in Section 4.5) is able to very quickly detect that the gadget is indeed NI. To check that this gadget is SNI, however, both **matverif** and **IronMask** enumerate all tuples, which becomes very expensive as the masking order grows. As a result, checking that the 11th-order gadget is SNI with **IronMask** would require at least a few days. On the other hand, **maskVerif** does not need to enumerate all tuples, and is able to much quickly determine that this gadget is SNI, taking just 5 minutes to do so at order 11.

Table 7: Verification time of NI and SNI verification of the ISW multiplication and $n \log n$ refresh by **IronMask** and **SILVER**

Gadget	Shares	Property	Verification time	
			IronMask	SILVER
ISW mult	4	NI	<1sec	1sec
		SNI	<1sec	2sec
	5	NI	<1sec	9min
		SNI	<1sec	14min
	6	NI	<1sec	>10h
		SNI	<1sec	>10h
refresh $n \log n$	6	NI	<1sec	8sec
		SNI	<1sec	20sec
	7	NI	<1sec	7min
		SNI	<1sec	14min
	8	NI	<1sec	>10h
		SNI	2sec	>10h

As mentioned earlier, **SILVER**, while being the only tool mentioned here that is complete on any gadget, suffers from severe performance limitations. This is illustrated in Table 7, which shows

that **SILVER** is several orders of magnitude slower than **IronMask** on the ISW multiplication and the $n \log n$ refresh.

Random Probing Model. Table 8 shows the time needed by **VRAPS** and **IronMask** to compute the maximum tolerated leakage probability of the ISW multiplications when setting c_{max} to 4 (which is the maximum that is computable by **VRAPS** in reasonable time). **IronMask** was not multi-threaded in this benchmark, and we recall that **VRAPS** does not support multi-threading. **IronMask** is several orders of magnitude faster than **VRAPS** in addition to being more precise (since **VRAPS** can incorrectly classify tuples as failures). The performance gains are explained by several factors. First, **IronMask** is written in C, whereas **VRAPS** is written in SageMath. Second, **IronMask** uses a complete technique based on Gaussian elimination to determine if tuples are failures, whereas **VRAPS** uses SageMath’s symbolic calculus to iteratively apply simplification rules inspired from **maskVerif**. Third, **IronMask** allocates less memory, and performs its Gaussian elimination on the fly (see Section 4.4), whereas **VRAPS** allocates chunks of memory to store batches of tuples, and restarts the simplifications from scratch for each tuple.

Table 8: Performance of t-RPE verification of **IronMask** and **VRAPS** on ISW multiplication gadgets at orders 4 to 6

Shares	t	c_{max}	#wires	$\log_2$ maximum tolerated proba	Verification time	
					IronMask	VRAPS
5	2	4	180	[-11.00,-10.67]	3sec	1h 15min
6	2	4	267	-13	17sec	24h
7	3	4	371	[-12.00,-7.83]	24sec	24h

The performance gains of **IronMask** over **VRAPS** have two main benefits. First, **IronMask** can be more useful for prototyping, since for small c_{max} it can provide approximate results within a few seconds. Second, **IronMask** can compute exact and more precise results by increasing c_{max} , as shown in Table 4.

6 Related Work

Many tools have been implemented in the past few years to verify software and hardware masked implementations. While we do not intend to provide an exhaustive list, we briefly recall the main lines of works for the verification of probing-like and random-probing like properties.

In 2012, Moss *et al.* [34] design the first automatic type-based masking compiler to provide first-order security against DPA. Following this seminal work, Bayrak *et al.* [10] investigate SMT-based method to evaluate the statistical independence between leakage and secrets. Eldib, Wang, and Schaumont [27] extend it to the verification of higher-order targets using a notion very similar to non-interference. Nevertheless, the complexity of their model counting approach restricts it to small masking orders.

Barthe *et al.* [5] then formalize the connection between the security of masked implementation and probabilistic non-interference. Their method makes it possible to overcome the combinatorial explosion of observation sets for high orders. The resulting tool, **maskVerif**, thus verifies reasonable circuits at reasonable masking orders. After several improvements in the past few years [6,3], it includes the verification of most probing-like security notions for different leakage models, including

the robust probing model in the presence of glitches. Its extension into `scVerif` [20] captures even more advanced hardware side effects [8]. In the same line of work, `checkMasks` [23] offers the same functionalities with a larger scope (*e.g.*, verification of Boolean to Arithmetic masking conversion) and also a polynomial time verification on selected gadgets. In the same vein, Zhang *et al.* use abstraction-refinement techniques to improve scalability and precision with their tool `SCInfer` [36] whose complexity remains significant. Bordes and Karpman [19] also improves accuracy with the elimination of false negatives.

In a parallel sequence of work, `Rebecca` [18] was designed to verify the probing security in presence of glitches directly on Verilog implementations. It preceded the similar improvement on `maskVerif` to also handle hardware implementations. One step further, `Coco` [29] was designed to check masked software implementations given any possible architectural side effects. Inspired from `Rebecca`, it analyzes a CPU design as a hardware circuit and investigates all the potential leaks of several shares. Finally, `SILVER` [33] offers the verification of the classical probing-like security properties for hardware implementations with a complete method based of the analysis of probability distributions.

Since the few past years, the community has made important effort to provide designs in the more realistic random probing model, see *e.g.* [1,26,2,13,16,20]. The random probing expandability (RPE) approach developed in [13] currently gives the best complexity to achieve arbitrary random probing security with a constant (and quantifiable) leakage probability. `VRAPS` [13] was the very first tool to verify random probing properties and it was followed by `STRAPS` [20], which additionally provides a probabilistic mode to boost the performances, but still does not provide a complete verification method since it uses a set of verification rules from `maskVerif` which by construction are not complete.

While all existing verification tools struggle to scale either to higher orders or to larger algorithms, a different and complementary approach consists in compiling programs that are secure by design using the composition security properties of some gadgets. This is notably the case of `maskComp` [6], which tackles the composition problem by introducing the (S)NI notions. This compiler is based on standard t -SNI gadgets (built from the ISW scheme [32]) and inserts t -SNI refresh gadgets at carefully chosen locations for the whole implementation to be t -NI. This was further improved in `tightPROVE` [15] and `tornado` [14], which rely on the same gadgets and inserts t -SNI refresh gadgets only when mandatory for the circuit to be t -probing secure.

7 Conclusion

In this paper, we introduce `IronMask`, a new tool for the formal verification of masking security. Our tool is versatile: it supports the verification of many probing and random probing security, composition and expandability notions. We further introduce a new algebraic characterization for quadratic gadgets with non-linear randomness which notably captures multiplication gadgets refreshing their inputs (which have recently been used in different state-of-the-art masking schemes). From this characterization, we design a complete verification method in the sense that it produces complete verification results (which are notably not affected by false positives). We provide a detailed description of the different algorithms, data structures and optimizations composing our tool. We also introduce a new constructive method for the exhaustive enumeration of so-called incompressible failure tuples. This approach provides speed-up in some cases (specifically for RPS and NI notions).

We developed (in C) and benchmarked the implementation of **IronMask**, which, enjoying various optimizations, will be made open source. For standard probing security notions (NI, SNI), **IronMask** has similar performance as existing high-performance tools (**maskVerif** & **matverif**), while having the advantage of providing complete results for quadratic gadgets with non-linear randomness. It is also several order of magnitude faster than **SILVER**, the only other tool providing complete results for those gadgets. For random probing security notions (RPC, RPE) our tool is several orders of magnitude faster than, and also complete compared to, the previous tool **VRAPS**. These completeness and increased performance allow us to report tighter and better bounds for RPE masking gadgets, which improve the tolerated leakage probability of state-of-the-art random probing secure compilers.

Acknowledgment. This work is partly supported by the French FUI-AAP25 VeriSiCC project.

References

1. Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd ACM STOC*, pages 715–724. ACM Press, June 2011.
2. Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part III*, volume 10993 of *LNCS*, pages 427–455. Springer, Cham, August 2018.
3. Gilles Barthe, Sonia Belaïd, Gaëtan Cassiers, Pierre-Alain Fouque, Benjamin Grégoire, and François-Xavier Standaert. **maskVerif**: Automated verification of higher-order masking in presence of physical defaults. In Kazuo Sako, Steve Schneider, and Peter Y. A. Ryan, editors, *ESORICS 2019, Part I*, volume 11735 of *LNCS*, pages 300–318. Springer, Cham, September 2019.
4. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Improved parallel mask refreshing algorithms: generic solutions with parametrized non-interference and automated optimizations. *J. Cryptogr. Eng.*, 10(1):17–26, 2020.
5. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, and Pierre-Yves Strub. Verified proofs of higher-order masking. In Elisabeth Oswald and Marc Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 457–485. Springer, Berlin, Heidelberg, April 2015.
6. Gilles Barthe, Sonia Belaïd, François Dupressoir, Pierre-Alain Fouque, Benjamin Grégoire, Pierre-Yves Strub, and Rébecca Zucchini. Strong non-interference and type-directed higher-order masking. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *ACM CCS 2016*, pages 116–129. ACM Press, October 2016.
7. Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 535–566. Springer, Cham, April / May 2017.
8. Gilles Barthe, Marc Gourjon, Benjamin Grégoire, Maximilian Ortl, Clara Paglialonga, and Lars Porth. Masking in fine-grained leakage models: Construction, implementation and verification. *IACR TCHES*, 2021(2):189–228, 2021.
9. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 23–39. Springer, Berlin, Heidelberg, August 2016.
10. Ali Galip Bayrak, Francesco Regazzoni, David Novo, and Paolo Ienne. Sleuth: Automated verification of software power analysis countermeasures. In Guido Bertoni and Jean-Sébastien Coron, editors, *CHES 2013*, volume 8086 of *LNCS*, pages 293–310. Springer, Berlin, Heidelberg, August 2013.
11. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 616–648. Springer, Berlin, Heidelberg, May 2016.
12. Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Private multiplication over finite fields. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part III*, volume 10403 of *LNCS*, pages 397–426. Springer, Cham, August 2017.

13. Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part I*, volume 12170 of *LNCS*, pages 339–368. Springer, Cham, August 2020.
14. Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In Anne Canteaut and Yuval Ishai, editors, *EUROCRYPT 2020, Part III*, volume 12107 of *LNCS*, pages 311–341. Springer, Cham, May 2020.
15. Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In Thomas Peyrin and Steven Galbraith, editors, *ASIACRYPT 2018, Part II*, volume 11273 of *LNCS*, pages 343–372. Springer, Cham, December 2018.
16. Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 313–343. Springer, Cham, October 2021.
17. Sonia Belaïd, Matthieu Rivain, Abdul Rahman Taleb, and Damien Vergnaud. Dynamic random probing expansion with quasi linear asymptotic complexity. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 157–188. Springer, Cham, December 2021.
18. Roderick Bloem, Hannes Groß, Rinat Iusupov, Bettina Könighofer, Stefan Mangard, and Johannes Winter. Formal verification of masked hardware implementations in the presence of glitches. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 321–353. Springer, Cham, April / May 2018.
19. Nicolas Bordes and Pierre Karpman. Fast verification of masking schemes in characteristic two. In Anne Canteaut and François-Xavier Standaert, editors, *EUROCRYPT 2021, Part II*, volume 12697 of *LNCS*, pages 283–312. Springer, Cham, October 2021.
20. Gaëtan Cassiers, Sebastian Faust, Maximilian Ortl, and François-Xavier Standaert. Towards tight random probing security. In Tal Malkin and Chris Peikert, editors, *CRYPTO 2021, Part III*, volume 12827 of *LNCS*, pages 185–214, Virtual Event, August 2021. Springer, Cham.
21. Gaëtan Cassiers and François-Xavier Standaert. Trivially and efficiently composing masked gadgets with probe isolating non-interference. *IEEE Trans. Inf. Forensics Secur.*, 15:2542–2555, 2020.
22. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *CRYPTO’99*, volume 1666 of *LNCS*, pages 398–412. Springer, Berlin, Heidelberg, August 1999.
23. Jean-Sébastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 2018*, volume 10892 of *LNCS*, pages 65–82. Springer, Cham, July 2018.
24. Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *FSE 2013*, volume 8424 of *LNCS*, pages 410–424. Springer, Berlin, Heidelberg, March 2014.
25. Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with $d+1$ shares in hardware. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *CHES 2016*, volume 9813 of *LNCS*, pages 194–212. Springer, Berlin, Heidelberg, August 2016.
26. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, Berlin, Heidelberg, May 2014.
27. Hassan Eldib, Chao Wang, and Patrick Schaumont. Formal verification of software countermeasures against side-channel attacks. *ACM Trans. Softw. Eng. Methodol.*, 24(2):11:1–11:24, 2014.
28. Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR TCHES*, 2018(3):89–120, 2018.
29. Barbara Gigerl, Vedad Hadzic, Robert Primas, Stefan Mangard, and Roderick Bloem. Coco: Co-design and co-verification of masked software implementations on CPUs. In Michael Bailey and Rachel Greenstadt, editors, *USENIX Security 2021*, pages 1469–1468. USENIX Association, August 2021.
30. Louis Goubin and Jacques Patarin. DES and differential power analysis (the “duplication” method). In Çetin Kaya Koç and Christof Paar, editors, *CHES’99*, volume 1717 of *LNCS*, pages 158–172. Springer, Berlin, Heidelberg, August 1999.
31. Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *EUROCRYPT 2017, Part I*, volume 10210 of *LNCS*, pages 567–597. Springer, Cham, April / May 2017.

32. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, Berlin, Heidelberg, August 2003.
33. David Knichel, Pascal Sasdrich, and Amir Moradi. SILVER - statistical independence and leakage verification. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part I*, volume 12491 of *LNCS*, pages 787–816. Springer, Cham, December 2020.
34. Andrew Moss, Elisabeth Oswald, Dan Page, and Michael Tunstall. Compiler assisted masking. In Emmanuel Prouff and Patrick Schaumont, editors, *CHES 2012*, volume 7428 of *LNCS*, pages 58–75. Springer, Berlin, Heidelberg, September 2012.
35. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT 2013*, volume 7881 of *LNCS*, pages 142–159. Springer, Berlin, Heidelberg, May 2013.
36. Jun Zhang, Pengfei Gao, Fu Song, and Chao Wang. Scinfer: Refinement-based verification of software countermeasures against side-channel attacks. In Hana Chockler and Georg Weissenbacher, editors, *Computer Aided Verification - 30th International Conference, CAV 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings, Part II*, volume 10982 of *Lecture Notes in Computer Science*, pages 157–177. Springer, 2018.

A Proof of Lemma 2

Proof. The proof follows the different steps of the method described above. All the statements hold from Lemma 1 except that the sets I_1, I_2 are necessary and sufficient for a perfect simulation of $\vec{P}''$. We prove this statement hereafter.

For any random distributions $\mathcal{D}_{\vec{x}_1}$ and $\mathcal{D}_{\vec{x}_2}$ over $\mathbb{K}^n$, we denote $\mathcal{D}_{\vec{P}}$ the distribution induced on $\vec{P}$ by picking $\vec{x}_1 \leftarrow \mathcal{D}_{\vec{x}_1}$, $\vec{x}_2 \leftarrow \mathcal{D}_{\vec{x}_2}$, $\vec{r}_1 \leftarrow \mathbb{K}^{\rho_1}$, $\vec{r}_2 \leftarrow \mathbb{K}^{\rho_2}$. Then I_1 and I_2 are the minimal sets such that for any distributions $\mathcal{D}_{\vec{x}_1}$ and $\mathcal{D}_{\vec{x}_2}$, there exists a probabilistic algorithm $\mathcal{S}$ (the simulator) which given $\vec{x}_1|_{I_1}$ and $\vec{x}_1|_{I_2}$ outputs a tuple $\vec{P}$ which is i.i.d. as $\mathcal{D}_{\vec{P}}$ w.r.t. the random draw $\vec{x}_1 \leftarrow \mathcal{D}_{\vec{x}_1}$, $\vec{x}_2 \leftarrow \mathcal{D}_{\vec{x}_2}$ and the random coins of $\mathcal{S}$.

Direction 1: The sets (I_1, I_2) are necessary to simulate $\vec{P}''$. Here we need to perfectly simulate the distribution of $\vec{P}''$ given the random samplings $\vec{x}_1 \leftarrow \mathcal{D}_{\vec{x}_1}$, $\vec{x}_2 \leftarrow \mathcal{D}_{\vec{x}_2}$, $\vec{r}_1 \leftarrow \mathbb{K}^{\rho_1}$, $\vec{r}_2 \leftarrow \mathbb{K}^{\rho_2}$ for any distributions $\mathcal{D}_{\vec{x}_1}$ and $\mathcal{D}_{\vec{x}_2}$ over $\mathbb{K}^n$. Let us consider the uniform distribution for $\mathcal{D}_{\vec{x}_1}$ then the m first coordinates of $\vec{P}''$ (i.e. the expressions of the form (8)) can be written as $\vec{P}''|_{[m]} = (\vec{u} \cdot \vec{h}_{p_1''}, \dots, \vec{u} \cdot \vec{h}_{p_m''})$, where $\vec{u}$ is a vector uniformly sampled on $\mathbb{K}^{n+\rho_1}$. Recall that the $\vec{h}_{p_i''}$'s coordinates are expressions of the form (6) w.r.t. $(\vec{x}_2, \vec{r}_2)$.

Given the values taken by the $\vec{h}_{p_i''}$'s we can have different distributions for $\vec{P}''|_{[m]}$. A particular case is the distribution “ $\vec{P}''|_{[m]} = \vec{0}$ with probability 1” which appears if and only if $\vec{h}_{p_1''} = \dots = \vec{h}_{p_m''} = \vec{0}$. In order to evaluate the probability of outputting $\vec{P}''|_{[m]} = \vec{0}$ (which must be exact for a perfect simulation), the simulator must hence evaluate the probability that $\vec{h}_{p_1''} = \dots = \vec{h}_{p_m''} = \vec{0}$ occurs, which must be further conditioned on the remaining expressions p_i'' of the form (6) w.r.t. $(\vec{x}_2, \vec{r}_2)$ (i.e. the probes on R_2). This precisely means solving the linear system obtained from the expressions in $\vec{P}_2$ which can be done by Gaussian elimination w.r.t. the $\vec{r}_2$ variables (just as what is actually performed by step 4 of the verification method). The resulting equations without $\vec{r}_2$ variables imply some linear constraints on some of the shares from $\vec{x}_2$. These shares must then be known by the simulator in order to decide if the system has a solution (and to evaluate the probability to get $\vec{h}_{p_1''} = \dots = \vec{h}_{p_m''} = \vec{0}$). Moreover, these shares are by construction the shares of indexes in I_2 .

The exact same proof apply to I_1 by taking the uniform distribution for $\mathcal{D}_{\vec{x}_2}$ and considering the expressions of the form (9) (together with the the probes on R_1).

Direction 2: The sets (I_1, I_2) are sufficient to simulate $\vec{P}''$. Suppose that we can perfectly simulate the tuples of algebraic expressions $\vec{P}_1, \vec{P}_2$ using sets of input shares I_1 on input sharing $\vec{x}_1$ and I_2 on $\vec{x}_2$ respectively as described in step 4 above. Let two new sets of input shares $\tilde{I}_1 = [n]$ on $\vec{x}_1$ and $\tilde{I}_2 = [n]$ on $\vec{x}_2$.

Observe first that we can perfectly simulate $\vec{P}''$ using the sets of input shares $\tilde{I}_1$ and I_2 . In fact, in the algebraic expression of each probe p_i'' in $\vec{P}''$ of the form (8), the coordinates of the $\vec{h}^{(p_i'')}$'s can all be perfectly simulated using I_2 since by hypothesis we can perfectly simulate $\vec{P}_2$. Also, the randoms in $\vec{r}_1$ are perfectly simulated by generating uniform random values, and all shares of input $\vec{x}_1$ are simulated using the full input sharing in $\tilde{I}_1 = [n]$. Since we can perfectly simulate each term in the expression of p_i'' , then we can perfectly simulate the expression p_i'' and hence we can perfectly simulate $\vec{P}''$ using $\tilde{I}_1$ and I_2 . Similarly, we can perfectly simulate $\vec{P}''$ using the sets of input shares I_1 and $\tilde{I}_2 = [n]$ by observing the expressions of p_i'' of the form (9).

Thanks to [11, Lemma 7.5] (which demonstrates that if a set of probes can be simulated from different sets of inputs shares, then it can also be simulated by the intersection of these sets), we get that $\vec{P}''$ can be perfectly simulated using the sets of input shares $\tilde{I}_1 \cap I_1 = I_1$ and $\tilde{I}_2 \cap I_2 = I_2$, which proves that by perfectly simulating the tuples $\vec{P}_1, \vec{P}_2$ using I_1, I_2 , we can perfectly simulate $\vec{P}''$ using I_1, I_2 . This concludes the proof for this direction.

B Dimension Reduction

We recall hereafter the principle of the dimension reduction optimizations from [11,19] and show how to make the first one work in the random probing model, and why the second one cannot be used in this model.

Removing elementary probes Elementary deterministic probes refer to input shares, and products of input shares. The idea behind the removal of those probes is that if a tuple P functionally depends on k input shares, we can always make it depend on $k + k'$ input shares (with $k + k'$ less or equal to the number of shares of the gadget) by adding elementary deterministic probes. For instance, if a tuple t does not depend on the input share a_0 , then the tuple (t, a_0) does, and so do any of the tuples of the form $(t, a_0 b_i)$.

The goal of our search procedure, instead of finding tuples of size k_1 that depend on t input shares, now becomes to find tuples of size $k_2 \leq k_1$ that depend on $t - (k_1 - k_2)$ input shares. In the probing model, the existence of such a tuple is enough to know that the property being checked does not hold. In the random probing model, however, we want to generate and count all failures. When we find such a tuple P , we thus generate all expansions of P combined with elementary deterministic probes that leak t input shares, thus making sure that all failures of the gadget are generated.

Similarly, we can remove elementary *random* probes: if, after the Gaussian elimination on a tuple P , some input shares are masked by random variables, we can make them appear by adding the corresponding randoms. For instance, consider the 1-element tuple $\mathbf{a0} + \mathbf{r1} + \mathbf{a1}$. It is easy to see that adding $\mathbf{r1}$ to this tuple would make it leak $\mathbf{a0}$ and $\mathbf{a1}$. For simplicity, we keep elementary random probes when checking random probing properties, and only perform this optimization for probing properties.

Using reduced sets This optimization is formally introduced and proven correct in [19]. For completeness, we informally recall its principle here.

Let P and P' be two sets of probes. P' is said to be a *reduced set* for P iff $\#P' \leq \#P$ and for every linear combination of probes of P , there exists a linear combination of P' using an equal or lower amount of probes, which contains exactly the same random dependencies, and at least as many input share dependencies. [19] proved that if P is a set of all wires of a gadget G and P' is a reduced set for P , then, to prove that G is NI or SNI, the set of probes P' can be used instead of P to enumerate all tuples. If no failure is found in P' , then none can be found in P either, and, conversely, if a failure is found in P' , the same failure exists in P .

For instance, if we consider a set $P = \{ r0, a0, a1, a0 + r0, a0 + r0 + a1 \}$, then the set $P' = \{ r0, a0, a1, a0 + r0 + a1 \}$ is a reduced set for P . Evaluating the probing security of a gadget using the latter would yield the same conclusion as with the former, while being faster since the latter contains one less probe.

This optimization is especially potent on ISW-like multiplications, which contain a lot of wires of the form $X + a_i b_j$: the wire X can often be omitted since (informally), $X + a_i b_j$ contains the same random dependencies as X , but, additionally, contains some additional input shares.

In the random probing model, this optimization cannot be used, because the a set P and a reduced set P' would yield different failures. For instance, consider the sets proposed earlier as example: $P = \{ r0, a0, a1, a0 + r0, a0 + r0 + a1 \}$, and $P' = \{ r0, a0, a1, a0 + r0 + a1 \}$. The tuple $(a0+r0, a0+r0+a1, a1)$, made of wires of P , reveals 2 input shares, and cannot be build from wires of P' , since $a0+r0$ is not in P' .

C Incompressible tuples

Checking whether a tuple is a failure or not with the SIS procedure is expensive, especially in multiplication gadgets with refreshes on the inputs, where we need to factorize multiplications and perform a total of 3 Gaussian eliminations. An idea from [13] to tackle this issue is to use the notion of *incompressible failure tuple* (Definition 4).

To speed up the verification, we can keep all incompressible failures in a data structure. Then, to check whether a tuple P is a failure, we can start by checking if any subtuple $P' \subset P$ is an incompressible failure. If so, then there is no need to perform the Gaussian eliminations. If no P' is an incompressible failure, then the full verification must be applied. If P is thus found to be a failure, then it is incompressible by definition, and is added to the set of incompressible failures.

For instance, consider the 2-share ISW multiplication of Figure 1a. When considering tuples of size 2, we would find that one of the failures is the tuple $P_1 = (a0*b1+r0+a1*b0, r0)$, since it leaks both shares of a (and b for that matter). Then, when moving on to tuples of size 3, we would eventually consider the tuple $P_2 = (a0*b1+r0+a1*b0, r0, a0)$. Instead of performing a Gaussian elimination to determine whether P_2 is a failure or not, we could simply observe that it contains P_1 , which we already know to be a failure. We can thus conclude that P_2 is a failure as well, without having to perform the Gaussian elimination.

VRAPS [13] uses a list to store the incompressible failures. As a result, the more incompressible failures had been computed, the more expensive membership-checking became. Instead, we use a *trie* (or *prefix tree*). Testing membership of a tuple P of size k inside a trie has complexity $\theta(k)$, and, if P is not in the trie, the search stops strictly before k steps. This is better than a hash table as well, which would require to hash the whole tuple in at least k steps, regardless of whether P is in the hash table or not.

Table 9: COLUMNS map for the constructive generation of incompressible tuples of Figure 1b

input shares			randoms		
a0	a1	a2	r0	r1	r2
a0	a1	a2	r0	r1	r2
a0 + r0	a1 + r0	a2 + r1	a0 + r0	a2 + r1	a1 + r0 + r2
a1 + r0 + r1	a1 + r0 + r2	a2 + r1 + r2	a1 + r0	a2 + r1 + r2	a2 + r1 + r2
			a0 + r0 + r1	a0 + r0 + r1	

The weakness of this optimization is that for a tuple P of size k , it requires to check all 2^k subtuple $P' \subset P$. This effective complexity can be improved because we use a trie. For instance, if the tuple (p_1, p_2) is not a valid path in the trie, then no tuples (p_1, p_2, p_i) can be in the trie either. Still, the benefits of this approach depend on the size of the gadget, the number of multiplications, and the size of the tuples.

VRAPS [13] is written in SageMath and uses expensive symbolic computation to determine whether tuples are failures or not. As a result, this optimization, albeit implemented with a list instead of a trie, was enough to speed up the verification. In our case, the benefits of this optimization are more situational, and can only be seen on some multiplication gadgets with input refreshes. Still, incompressible tuples are at the core of our new approach to compute failures, introduced in Section 4.5.

D Constructive algorithm

The enumerative approach of Section 4.2 generates a lot of tuples which are trivial non-failures because they do not contain enough shares to be failures, or their shares are masked by random variables. Thus, we designed a constructive algorithm to only generate potential failures. More precisely, our constructive algorithm aims at generating *incompressible failure tuples*, which are defined in [13] as follows:

Definition 4 (Incompressible failure tuple). *A tuple $\vec{P}$ is an incompressible failure tuple if it is a failure, and if no tuple $\vec{P}' \subset \vec{P}$ is a failure itself. ($\subset$ between two tuples means that all wires of $\vec{P}'$ are included in the tuple $\vec{P}$).*

Note: The notion of incompressible failure tuple was used in VRAPS [13] to speed up the enumerative verification of random probing properties. We investigated this technique, and improved on the implementation of [13] but observed that given the current high performance of our implementation, the optimization from [13] is not advantageous in our case (see Appendix C for more details).

We will start by describing our constructive algorithm for the well-adapted case of LR-gadgets and then explain how to extend it to NLR-gadgets. The idea is that given wires which are all of the form (5), a failure tuple of probes on these wires has a specific form: it contains some wires with input shares, and if those input shares are *masked by randoms*, it contains some additional wires to cancel out those randoms. The expression *masked by randoms* means that the perfect simulation of the considered probe amounts to generating a uniform random value. This is typically the case in a tuple of probes where a random value appears only once in one of the expressions, which then can be used to mask the expression of that probe. To cause a failure event and avoid masking the expressions, additional wires using the same randoms are added to cancel them out.

We start by giving the intuition of the algorithm on the 3-share refresh gadget presented in Figure 1b. We first build a map (Table 9), called COLUMNS, whose keys are the input shares and

Algorithm 5 Our constructive algorithm to generate failures. G is the gadget we are considering and n is the number of shares required for a tuple to be a failure.

```

1: procedure UNMASKTUPLE( $G, S, \vec{P}, unmask\_index$ )
2:   if  $\vec{P}$  is a failure then
3:     if  $\vec{P}$  is incompressible then
4:        $S \leftarrow S \cup \{\vec{P}\}$ 
5:     return
6:   if  $unmask\_index > \text{length}(\vec{P})$  then
7:     return
8:   UNMASKTUPLE( $G, S, \vec{P}, unmask\_index+1$ )
9:    $\vec{P}_{\text{Gauss}} \leftarrow \text{GAUSSELMINATION}(\vec{P})$ 
10:  if  $\vec{P}_{\text{Gauss}}[unmask\_index]$  contains no randoms then
11:    return
12:   $r \leftarrow$  any random from  $\vec{P}_{\text{Gauss}}[unmask\_index]$ 
13:  for each wire  $w$  of  $G$  containing  $r$  and not in  $\vec{P}$  do
14:     $\vec{P}' \leftarrow \vec{P} \cup w$ 
15:    UNMASKTUPLE( $G, S, \vec{P}', unmask\_index+1$ )
16: procedure CONSTRUCTIVEFAILURESGENLR( $G, n$ )
17:    $S \leftarrow \emptyset$ 
18:   for each tuple  $\vec{P}$  in  $\mathcal{L}$  do
19:     UNMASKTUPLE( $G, S, \vec{P}, 0$ )
20:   return  $S$ 

```

randoms of the gadget considered, and whose values are all the wires that depend on those inputs and randoms (a wire will be displayed in several columns if it depends on several shares and/or randoms). To build a failure tuple that leaks 3 shares, we can pick one wire from each of the **a0**, **a1** and **a2** buckets, say (**a0**, **a1+r0**, **a2+r1**) for instance. This tuple is not a failure because the shares **a1** and **a2** are masked by the randoms **r0** and **r1** (the two random values appear only once in the tuple and can be used to mask the corresponding expressions). We thus pick a wire from the **r0** bucket and add it to the tuple, say **a0 + r0 + r1** (which happens to cancel **r1** as well as **r0**). The resulting tuple is (**a0**, **a1+r0**, **a2+r1**, **a0+r0+r1**), which is a failure. By doing this for every possible wire of each column, we can generate all failures of the gadget of Figure 1b.

Algorithm 5 introduces more formally this procedure for LR-gadgets. This algorithm lists all of the tuples composed of one element from each input share column (line 18); we note the resulting list $\mathcal{L}$. Note that those tuples might have some duplicates since some wires appear in several columns: these duplicates are removed while building the tuples (which implies that the tuples in $\mathcal{L}$ contains possibly less than n elements).

Then, for each tuple in $\mathcal{L}$, the recursive procedure UNMASKTUPLE adds wires to the tuple so as to cancel the randoms that mask its input shares. This procedure takes as argument the circuit G , the set of incompressible tuples already computed S , a tuple $\vec{P}$ that needs to be turned into a failure and an integer $unmask\_index$ that contains the next index of $\vec{P}$ that we should try to unmask. First, UNMASKTUPLE checks if $\vec{P}$ is a failure (line 2). To do so, we can use the procedure SIS.LR (Algorithm 1). If $\vec{P}$ is a failure, we then check if it is incompressible (line 3) by checking if any tuple $\vec{P}' \subset \vec{P}$ is already in S . Ignoring line 8 for now, a Gaussian elimination is then performed on $\vec{P}$ (line 9). If the $unmask\_index^{\text{th}}$ element of the resulting tuple $\vec{P}_{\text{Gauss}}$ contains no random, then there is nothing to unmask, and we can move on to the next index (which was actually already done by line 8). Otherwise, we select any random r of $\vec{P}_{\text{Gauss}}$ and try to add to $\vec{P}$ each wire that

contains r (i.e., each wire of the r column of the COLUMNS map, and move to the next *unmask_index* (lines 12 to 15).

As a matter of fact, unmasking each element of $\vec{P}$ one by one misses some failures. This is the reason for line 8, which basically skips the unmasking of the element of $\vec{P}$ at index *unmask_index* to move directly to index *unmask_index*+1. Consider for instance a 2-share gadget and the tuple $\vec{P} = (\mathbf{a0}+\mathbf{r0}, \mathbf{a1}+\mathbf{r1}+\mathbf{r0})$. After the Gaussian elimination, the 1st element of $\vec{P}$ is $\mathbf{r0}$, and lines 12 to 15 of UNMASKTUPLE will thus try to add a wire containing $\mathbf{r1}$ to the tuple. However, this would be missing the fact that the 2nd element of $\vec{P}$, $\mathbf{a1}+\mathbf{r1}+\mathbf{r0}$, already contains $\mathbf{r1}$ and thus *somewhat* unmasks $\mathbf{a0}+\mathbf{r1}$. By skipping the first element of $\vec{P}$, we will then try to unmask its second element, by adding the wire $\mathbf{r1}$ to the tuple for instance. This will produce the tuple $(\mathbf{a0}+\mathbf{r0}, \mathbf{a1}+\mathbf{r1}+\mathbf{r0}, \mathbf{r1})$, which is a failure, and would have been missed without the recursive call of line 8.

This constructive method is exhaustive since any incompressible failure tuple $\vec{P}$ can be built by taking one elements in each column (and possibly remove duplicates) and then adding necessary elements to remove the masks. More precisely, consider a minimal sub-tuple of $\vec{P}$ which contains one element of each column. This sub-tuple $\vec{P}'$ will be listed in line 8. The other coordinates of $\vec{P}$ are necessary to remove the masks remaining after an application of the Gaussian elimination to $\vec{P}'$ (otherwise $\vec{P}$ would not be incompressible). Since Algorithm 5 is exhaustive in the removal of those masks, it will necessarily build $\vec{P}$ at some point.

Implementation Our implementation of UNMASKTUPLE in Algorithm 5 does not perform a full Gaussian elimination at every recursive call. Instead, the elimination is performed on the fly, similarly as we do for the enumerative algorithm (see Section 4.4). Likewise, we keep a variable *input_shares* containing the input shares already revealed by the current tuple $\vec{P}$, which enables to check if $\vec{P}$ is a failure in constant time, without having to call SIS_LR: we can simply check if *input_shares* contains n input shares.

Extension to gadgets with non-linear randomness The procedure UNMASKTUPLE of Algorithm 5 only considers gadgets with linear randomness. To adapt it for gadgets with non-linear randomness, we proceed in a similar manner as in SIS_NLR (Algorithm 2): a first step unmasks randoms that are used to refresh outputs, while a second step unmasks randoms that are used to refresh inputs. We call CONSTRUCTIVEFAILURESGENNLR this version of our constructive algorithm and define CONSTRUCTIVEFAILURESGEN as the function that chooses between CONSTRUCTIVEFAILURESGENLR and CONSTRUCTIVEFAILURESGENNLR depending on whether its input gadget contains linear or non-linear randomness.

Application Algorithm 6 shows how to use CONSTRUCTIVEFAILURESGEN to compute all failures of a gadget. While the latter returns all incompressible failures, to evaluate the failure function coefficients for random probing notions (RPC, RPE1, RPE2, RPS*), we need to count the number of all failures, regardless of their incompressibility. To do so, we expand all incompressible failures into regular failures by adding wires one by one (using the procedure EXPANDTUPLE, whose pseudo-code is trivial and left out for conciseness). However, doing so will lead to the same tuple being generated multiple times: for instance, if the tuples $(\mathbf{x1}, \mathbf{x2})$ and $(\mathbf{x1}, \mathbf{x3})$ are both incompressible failures, expanding them will generate $(\mathbf{x1}, \mathbf{x2}, \mathbf{x3})$ and $(\mathbf{x1}, \mathbf{x3}, \mathbf{x2})$, which are the same tuple. We thus use a hash table (called S_{failure} in Algorithm 6, and abstracted as a set for simplicity) to store the tuples that we generate and prevent counting multiple times the same tuple. In practice, our hash function returns the sum of the hashes of the indices of wires in the tuple, which results in a fairly low number of collisions.

Algorithm 6 GETCOEFFSRPCONSTR returns an array of c_{max} cells where the k^{th} index contains the number of failure tuples of size k in G

```

procedure GETCOEFFSRPCONSTR( $G, c_{max}$ )
   $coeffs \leftarrow$  empty array
   $t \leftarrow$  number of shares in  $G$ 
   $S_{incompr} \leftarrow$  CONSTRUCTIVEFAILURES_GEN( $G, t$ )
   $S_{failure} \leftarrow \emptyset$ 
  for  $k = 1$  to  $c_{max}$  do
     $S'_{failure} \leftarrow$  all tuples of  $S_{incompr}$  of size  $k$ 
    for each tuple  $\vec{P}$  of  $S_{failure}$  do
       $S'_{failure} \leftarrow S'_{failure} \cup \text{EXPANDTUPLE}(\vec{P})$ 
     $S_{failure} \leftarrow S'_{failure}$ 
     $coeffs[k] \leftarrow$  number of tuples in  $S_{failure}$ 
  return  $coeffs$ 

```

Remark We initially tried to count the failures from the incompressible failures without generating all of them. This problem can be formulated as follows: *Let W be a set of integers (the wires). Let S be a set of subsets of W of arbitrary sizes (the set of incompressible failures). How many subsets of W of size k are super-sets of elements of S (those subsets are non-incompressible failures)?* This is a problem of inclusion-exclusion, and solving it requires computing the intersections of all pairs of sets in the powerset of S . Since there are $2^{|S|}$ such sets, this approach would be prohibitively expensive for any gadget with more than a few incompressible failures.

Limitations Many tuples are generated multiple times by this constructive algorithms, each time through a different path in the recursion. For instance, on the refresh gadget of Figure 1b, the tuple $(a_0, a_1+r_0, a_2+r_1, a_1+r_0+r_1)$ can be generated by selecting (a_0, a_1+r_0, a_2+r_1) as the initial tuple (line 18 of Algorithm 5), and then adding $a_1+r_0+r_1$ at line 14. However, the same tuple can also be generated by selecting $(a_0, a_1+r_0+r_1, a_2+r_1)$ line 18 and then adding a_1+r_0 line 14. This phenomenon is even more impactful when dealing with multiplication gadgets, because multiple shares of the same input will appear on the same wire, resulting in larger columns (in particular in the “inputs” part of the COLUMNS map), which can lead to a worst complexity than simply enumerating all tuples.

Additionally, when checking properties t -NI with $t < n - 1$ or other properties where a tuple can reveal less than $n - 1$ input shares and yet be a failure (e.g., SNI, RPC, RPE), the constructive algorithm is often slower than the traditional enumerative one. Regardless of the property being checked and the failure condition, the constructive algorithm enumerates all tuples. However, with the constructive algorithm, for an n -share gadget, to generate tuples of that leak k shares, all $\binom{n}{k}$ possible combination of shares must be tested. For instance, on our running example of Figure 1b, to generate all failures leaking 2 shares, we would generate all failures leaking the 1st and the 2nd shares, then the ones leaking the 1st and the 2nd share, and, finally, the ones leaking the 2nd and the 3rd share. This formula is also true when checking NI and RP, except that in that case, all n shares must leak, which means that only $\binom{n}{n} = 1$ combination needs to be tested.

Appendix E

Formal Definition and Verification for Combined Random Fault and Random Probing Security

Hereafter is appended the full version of our paper [BFG⁺24], joint work with Jakob Feldtkeller, Tim Güneysu, Anna Guinet, Jan Richter-Brockmann, Matthieu Rivain, Pascal Sasdrich, and Abdul Rahman Taleb, published at **ASIACRYPT 2024**.

Formal Definition and Verification for Combined Random Fault and Random Probing Security

Sonia Belaïd¹ , Jakob Feldtkeller² , Tim Güneysu^{2,3} , Anna Guinet² ,
Jan Richter-Brockmann² , Matthieu Rivain¹ , Pascal Sasdrich² , and
Abdul Rahman Taleb¹

¹ CryptoExperts, Paris, France
{sonia.belaid, matthieu.rivain}@cryptoexperts.com
taleb.abdulrahman1@gmail.com

² Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
{jakob.feldtkeller, tim.guneysu, anna.guinet, jan.richter-brockmann,
pascal.sasdrich}@rub.de

³ DFKI, Bremen, Germany

Abstract. In our highly digitalized world, an adversary is not constrained to purely digital attacks but can monitor or influence the physical execution environment of a target computing device. Such side-channel or fault-injection analysis poses a significant threat to otherwise secure cryptographic implementations. Hence, it is important to consider additional adversarial capabilities when analyzing the security of cryptographic implementations besides the default black-box model. For side-channel analysis, this is done by providing the adversary with knowledge of some internal values, while for fault-injection analysis the capabilities of the adversaries include manipulation of some internal values.

In this work, we extend probabilistic security models for physical attacks, by introducing a *general random probing model* and a *general random fault model* to enable security analysis with arbitrary leakage and fault distributions, as well as the combination of these models. Our aim is to enable a more accurate modeling of low-level physical effects. We then analyze important properties, such as the impact of adversarial knowledge on faults and compositions, and provide tool-based formal verification methods that allow the security assessment of design components. These methods are introduced as extension of previous tools VERICA and IronMask which are implemented, evaluated and compared.

Keywords: Physical Security · Random Probing Model · Random Fault Model · Combined Analysis · VERICA · IronMask

1 Introduction

Advanced cryptographic schemes are typically examined within the common black-box model, which assumes that the internal values of the scheme remain concealed and protected from any adversary. However, over the past 25 years,

physical attacks that exploit the actual implementation of cryptographic algorithms have challenged these assumptions. In particular, Side-Channel Analysis (SCA) exploits dependencies between processed values and physical execution characteristics such as timing behavior [34], instantaneous power consumption [35], or electromagnetic emanations [28], to conclude information about a secret. Similarly, Fault Injection Analysis (FIA) manipulates the physical execution environment to create a faulty intermediate state such that the resulting output gives a hint about the processed secret. Common fault injection methods include clock and voltage glitching [51,21], targeted electromagnetic (EM) pulses [18,23], or focused laser beams [49]. Both attack vectors question a different assumption of the cryptographic black-box model: SCA that internal values are *confidential* and hidden from the adversary and FIA that the internal values have *integrity* and cannot be manipulated by the adversary. However, in a real attack scenario, the adversary is not restricted to performing only one of the two attack types. Indeed, first practical attack for Combined Analysis (CA), i.e., the combination of FIA and SCA, are emerging [3,15,43,44,45,46].

Constructing effective countermeasures against such physical attacks requires a deep understanding of attack properties and leakage behavior. To this end, the research community endeavors to theoretically model the leakage emanating from the victim device which is susceptible to exploitation by the adversary. In particular for SCA, in the famous d -probing model introduced by Ishai, Sahai, and Wagner [33], the leakage is modeled by the exact values of d internal variables of the adversary's choice. A victim device is then deemed d -probing secure if any such set of d intermediate variables is statistically independent on the processed secret. While this probing model facilitates security proofs, it sometimes fails to closely reflect the reality of embedded devices. For instance, it does not capture horizontal attacks [8], which exploit repeated manipulation of variables within an execution. As a consequence, the community is beginning to focus on more realistic leakage models, like the ϵ -random probing model [33,2,10]. In this model, the exact value carried by each wire of the circuit is leaked with probability p to the adversary. The security is then determined by the probability ϵ to obtain a secret-dependent probe combination. This model tightly reduces to the security in the practical noisy leakage model [38,22], where each variable leaks a noisy function of its value. Nevertheless, the random probing model is still insufficient in modeling low-level physical effects because it assigns the same independent probability p to each intermediate variable whereas the underlying noise is likely to be different and the independence assumption not verified in practice.

For modeling of FIA the adversary is given the ability to manipulate a set of intermediate variables in a predefined way and then their impact on the system output is analyzed. Models from the literature often take inspiration from probing models for SCA. Specifically, the k -fault model [32] allows the adversary to manipulate up to k intermediate values, while the random fault model [20] defines a fault probability q and manipulates each internal value with probability q . In both models, the adversary wins when they get a faulty output that cannot be detected or corrected, whereas the random fault model determines

the probability μ with which this happens. There again, while beneficial for a first approximation of physical security, the current models come with significant abstractions. In particular, the deterministic k -fault model cannot capture the imprecise nature of physical attacks due to the probabilistic fault behavior, while the random fault model fails to precisely model low-level physical characteristics of fault injection due to the same independent probability for each fault. Naturally, models for CA combine the capabilities of the individual models. All those models allow the analysis of certain types of secret-dependent leakages and enable pre-silicon evaluation.

Contribution. In this work, we provide a generalization of the probabilistic models for physical attacks by considering arbitrary leakage and fault distributions. In particular, we introduce a *general random probing model*, a *general random fault model*, and a *general random combined model* in Section 3, where the occurrence of a probe and a fault is not restricted to the same independent probability and instead arbitrary distributions can be chosen to model low-level circuit and noise effects. All our models, but especially our random combined model, are analyzed for interesting properties. Specifically, we analyze the impact of the adversarial knowledge on the injected faults on combined security. Since the resulting analysis complexity is very high, we introduce notions for compositions for all three models in Section 4. For our general random probing model, this is a straightforward extension of an existing notion for the ϵ -random probing model (details are given in Appendix A). For the other two models, to the best of our knowledge, this is the first attempt at composition under probabilistic leakage and fault behavior. We then continue to investigate the formal verification of the proposed combined composition notions for gadgets. Specifically, we introduce two methods of tool-based analysis using VERICA [40] and IronMask [11]. We explain how to extend these tools to verify the proposed notions and present the implementation details in Section 5. In particular, our extension of VERICA can analyze arbitrary circuits with exact precision (up to some threshold) but is comparably slow. In contrast, our extension of IronMask is much faster but restricted to (N)LR gadgets [11] with intermediate corrections and performs further approximations. We finally provide an extensive evaluation and comparison of the tools in Section 6.

2 Preliminaries

In the following, we give a short overview of the used notation and circuit model. Afterwards, we provide the basic concepts required to understand the contribution of this paper. Specifically, we introduce countermeasures for SCA and FIA and discuss security proofs via simulation.

2.1 Notation

Throughout the paper, we use a sans-serif font for functions (e.g., f) and an upper-case calligraphic font for sets (e.g., $\mathcal{S}$). We designate $\bar{\mathcal{S}}$ the complement

of a set $\mathcal{S}$, $|\mathcal{S}|$ its cardinality, and $\mathbb{D}\mathcal{S}$ a discrete probability distribution defined over the event set $\mathcal{S}$. Further, we name $\prod_i \mathbb{D}\mathcal{S}_i$ the joint probability distribution of independent distributions $\mathbb{D}\mathcal{S}_i$ where the joint probability is computed as the multiplication of the individual probabilities, i.e., $\Pr[a \cap b] = \Pr[a] \cdot \Pr[b]$, and with $\equiv$ the equality of distributions. To simplify the notation for n -times replication we denote with $\mathbb{V}_n = \{(0)^n, (1)^n\}$ the set that contains the zero vector $(0)^n$ and the one vector $(1)^n$ of size n . Other notations will be introduced throughout the paper where necessary.

2.2 Circuit Model

In this work, we model a circuit C as a *Directed Acyclic Graph (DAG)* $\mathcal{C} = \{\mathcal{G}, \mathcal{W}\}$, where vertices $g \in \mathcal{G}$ represent logical gates and edges $w \in \mathcal{W}$ represent wires carrying a Boolean value and connecting individual gates. We restrict the set of combinational gates to $\mathcal{G}_c = \{\text{inv}, \text{and}, \text{xor}, \text{or}\}$ and the set of memory gates to $\mathcal{G}_m = \{\text{reg}\}$. Further, we define a set of input and output gates $\mathcal{G}_{io} = \{\text{in}, \text{out}\}$, where *in* produces and *out* absorbs a Boolean value, and a set of probabilistic gates $\mathcal{G}_{\text{rand}} = \{\text{rand}\}$, where *rand* produces a uniform-random Boolean value. Finally, we define the set of constant gates $\mathcal{G}_{\text{const}} = \{\text{zero}, \text{one}\}$, where each gate produces the respective Boolean value. Hence, each gate is from the set $\mathcal{G}_{\text{all}} = \mathcal{G}_c \cup \mathcal{G}_m \cup \mathcal{G}_{io} \cup \mathcal{G}_{\text{rand}} \cup \mathcal{G}_{\text{const}}$. Given this, each gate implements a deterministic or probabilistic Boolean function $f_g : \mathbb{F}_2^h \rightarrow \mathbb{F}_2$, with $0 \leq h \leq 2$, of the respective functionality. Further, let $\mathcal{G}_f = \{\mathbb{F}_2^h \rightarrow \mathbb{F}_2 \mid h \leq 2\}$ be the set of all possible Boolean functions for this purpose. With respect to the fan-out of wires, we consider two different scenarios depending on the analysis of hardware or software. For hardware (cf. Section 5.2), we create a copy of the wire for each gate that is connected, i.e., for a wire with fan-out n we create n copies. Here a gate output can be connected to any number of copies of the same wire. For software (cf. Section 5.3), we say that each gate can have at most one output and introduce a special *copy* operation that outputs the input twice. Hence, to represent a wire with fan-out n we require $2n - 1$ wires to construct a tree of copys.

2.3 Countermeasures

Masking. A popular countermeasure against SCA is *Boolean masking* [14, 29], due to its sound formal foundation. The core idea is to split a secret $x \in \mathbb{F}_2$ into a vector $\langle x_0, \dots, x_{s-1} \rangle \in \mathbb{F}_2^s$, with $x_i \in \mathbb{F}_2$, such that $x = \bigoplus_{i=0}^{s-1} x_i$ and each subset $\{x_i \mid i \in [0, s-1]\}$ with cardinality smaller than s is statistically independent of x . We refer to a component x_i as a *share* of x with *share index* i . Similarly, a circuit is transformed to a *masked circuit*, which operates over shares of the inputs. In this paper, we assume the initial encoding *enc* of inputs and the final decoding *dec* of outputs (generating shares or recreating the secret, respectively) not as part of the masked circuit.

Replication. A popular countermeasure against FIA is the replication of the circuit in combination with a majority function for error correction purposes. In particular, a value $x \in \mathbb{F}_2$ is replicated to a vector $\langle x_0, \dots, x_{n-1} \rangle \in \mathbb{F}_2^n$ with $n = 2k + 1$, such that $\forall i, j \in [0, n - 1] : x_i = x_j$. Then, up to k faults can be corrected with a majority function maj . Likewise, a circuit is replicated n times, where every replication operates on a unique set of value replications.

2.4 Security Proofs via Simulation

Security proofs in the context of SCA are often conducted based on *simulation*. For this, two worlds are introduced. The first world represents a real implementation, while the second world is made trivially secure by removing the secret that an adversary tries to learn. If an adversary is not able to distinguish between the two worlds then the view of the adversary is proven to be independent of the secret. The proof works by construction of a simulator that recreates the distribution of the observed values without access to any secret.

However, in this work, we do not require perfect simulation but allow the simulator to be wrong with a small probability. In particular, we require a simulator to create a distribution that is ϵ close to the observed distribution. We say that any two probability distributions $\mathbb{D}_1$ and $\mathbb{D}_2$ are ϵ -close ($\mathbb{D}_1 \approx_\epsilon \mathbb{D}_2$) if their statistical distance is upper-bounded by ϵ , i.e.,

$$\frac{1}{2} \sum_x |\Pr_{\mathbb{D}_1}[x] - \Pr_{\mathbb{D}_2}[x]| \leq \epsilon.$$

3 Security Model

We start our contribution by defining probabilistic security models for SCA, FIA, and CA. For this, we build on existing models but capture a more general attack scenario.

3.1 General Random Probing Security

Adversary Model. We introduce a new generalization of the random probing model [33,2,10] called *general random probing model*. Here, a probing adversary $\mathcal{A}_p$ can invoke a circuit C multiple times and on each invocation, the exact values of a random subset of wires in C are leaked to $\mathcal{A}_p$. We denote the leaking combination of wires, i.e., the subset of the wires of the circuit that is given to $\mathcal{A}_p$, with $\tilde{\mathcal{W}} \subseteq \mathcal{W}$. Let $\mathcal{W}^\infty = \{\tilde{\mathcal{W}} \subseteq \mathcal{W}\}$ be the set of all wire combinations in C and $\mathbb{D}\mathcal{W}^\infty$ an arbitrary discrete probability distribution defined over $\mathcal{W}^\infty$. Further, we define the following two functions to first select a random element from $\mathcal{W}^\infty$ and then determine the values carried by the selected wires for a given input:

LeakingWires($C, \mathbb{D}\mathcal{W}^\infty$): The leaking-wire sampler selects for a given circuit C a wire combination $\tilde{\mathcal{W}}$ with probability $\Pr_{\mathbb{D}\mathcal{W}^\infty}[\tilde{\mathcal{W}}]$.

AssignWires($C, \tilde{\mathcal{W}}, x$): The assign-wire sampler takes a fixed input x for C and outputs the values assigned to the wires $w \in \tilde{\mathcal{W}}$ as a tuple in $\mathbb{F}_2^{|\tilde{\mathcal{W}}|}$. If C is probabilistic so is **AssignWires**().

Then, the view of $\mathcal{A}_p$ is formally defined as the *random probing leakage* $\mathcal{L}_{\tilde{\mathcal{W}}}(C, x)$, which is given by the random experiment

$$\begin{aligned}\tilde{\mathcal{W}} &\leftarrow \text{LeakingWires}(C, \mathbb{D}\mathcal{W}^\infty), \\ \mathcal{L}_{\tilde{\mathcal{W}}}(C, x) &\leftarrow \text{AssignWires}(C, \tilde{\mathcal{W}}, x).\end{aligned}$$

Security Definition. Intuitively, a circuit C is secure in our model if the view of $\mathcal{A}_p$ can be simulated with high probability without access to the secret, i.e., there exists a simulator **Sim** that recreates the distribution of the leaking wires $\tilde{\mathcal{W}}$ without knowledge of the secret, such that the failure probability of **Sim** is bounded by some (small) ϵ . A more formal definition is given in Definition 1. Throughout this work, we consider **enc** to be some encoding function, e.g., defined by Boolean masking (cf. Section 2.3).

Definition 1 (General Random Probing Security). *A circuit C is said to be $(\mathbb{D}\mathcal{W}^\infty, \epsilon)$ -random probing secure with respect to an encoding **enc** if there exists a simulator **Sim** such that for all inputs x :*

$$\text{Sim}(C, \tilde{\mathcal{W}}) \approx_\epsilon \mathcal{L}_{\tilde{\mathcal{W}}}(C, \text{enc}(x)).$$

The required simulator **Sim** can be constructed by returning a *simulation failure* $\perp$ whenever the exact distribution $\mathcal{L}_{\tilde{\mathcal{W}}}(C, \text{enc}(x))$ cannot be recreated without access to x and ensuring that

$$\Pr[\text{Sim}(C, \tilde{\mathcal{W}}) = \perp] = \epsilon,$$

and, conditioned to the event $\text{Sim}(C, \tilde{\mathcal{W}}) \neq \perp$,

$$\text{Sim}(C, \tilde{\mathcal{W}}) \equiv \mathcal{L}_{\tilde{\mathcal{W}}}(C, \text{enc}(x)).$$

Relation to Random Probing Model. The *random probing model* [33,2,10] is a specific instance of the above-defined general random probing model. Specifically, the probability distribution $\mathbb{D}\mathcal{W}^\infty$ is selected such that each wire $w \in \mathcal{W}$ leaks with the same probability p independent of all other wires.

As shown by Duc et al. [22], the random probing model can be seen as an intermediate model between the noisy leakage model [38] and the d -threshold probing model [33]. Here, the assumption of the mutually independent leakage probability p for all wires can be traced to the assumption of equal and mutually independent noise in the noisy leakage model. However, the latter does not necessarily hold in practice, neither in hardware nor in software [9]. By defining the leakage over an arbitrary discrete probability distribution $\mathbb{D}\mathcal{W}^\infty$, we can model scenarios where the noise of two wires is not independent (e.g.,

because they operate in parallel with the same background computation) and, hence, the leakage probability of the two wires is dependent. Similarly, dependencies in the leakage probability can also be caused by physical defaults such as glitches [36,39] and couplings [17,39] or by shared structures like the Power Distribution Network (PDN) [47].

The introduction of the arbitrary discrete probability distribution $\mathbb{D}\mathcal{W}^\infty$ also allows two different wire combinations of one wire $\{w_i\}$ and $\{w_j\}$ to have different leakage probabilities p_i and p_j , respectively. With that, it enables more fine-grained modeling for the contribution of individual wires to the occurring leakage of wire combinations (by a different weight for the wires w_i and w_j to occur). For example, an EM probe usually does not capture the entire circuit but only a subset of neighboring circuit elements. Hence, a subset of wires does not contribute to the leakage at all and can be modeled by wire combinations with zero probability. Modeling this correctly can lead to place-and-route algorithms that take EM probing into account and minimize the observable leakage. Other differences in the leakage probability may be caused by the difference in the driver strengths of individual gates, or even by the type of operation a wire is used for.

By introducing this general model, we hope to fuel research into the nature of physical leakage by looking specifically into the leakage dependencies and contributions of different circuit structures. However, due to the computational blow-up, we will stick to mutually independent leakage probabilities in the practical implementation of our evaluation tools (cf. Section 5).

3.2 General Random Fault Security

Adversary Model. We use the adversary model proposed by Feldtkeller et al. [26] that is based on a fault model from Richter-Brockmann et al. [42]. Here, a faulting adversary can invoke a circuit C multiple times, where on each invocation, a random subset of gates is manipulated according to a specified fault transformation. More specifically, a fault consists of a *fault location* $g \in \mathcal{G}$, i.e., a gate of the circuit, and a *fault transformation* $\tau : \mathcal{G}_f \rightarrow \mathcal{G}_f$, i.e., a transformation of the Boolean function a gate implements, where the fault model restricts the allowed transformations. Popular fault transformations are, e.g., $\tau_{\text{set}}(g) = \text{one}$, $\tau_{\text{reset}}(g) = \text{zero}$, or $\tau_{\text{flip}}(g) = \text{inv}(g)$. Hence, a fault is a pair $f = (g, \tau)$ and we denote by $\mathcal{F}$ the set of all possible faults, by $\mathcal{F}^\infty = \{\tilde{\mathcal{F}} \subseteq \mathcal{F}\}$ the set of all possible fault combinations $\tilde{\mathcal{F}}$ with distinct locations (i.e., each $g \in \mathcal{G}$ occurs at most once in $\tilde{\mathcal{F}}$), and by $\mathbb{D}\mathcal{F}^\infty$ an arbitrary distribution defined over $\mathcal{F}^\infty$. We define the following function to select a fault combination for a single circuit invocation:

AssignFaultGates($C, \mathbb{D}\mathcal{F}^\infty$): For a given circuit C , the faulty-gate sampler selects a fault combination $\tilde{\mathcal{F}}$ with probability $\Pr_{\mathbb{D}\mathcal{F}^\infty}[\tilde{\mathcal{F}}]$ and outputs the modified circuit $C^{\tilde{\mathcal{F}}}$ that we refer to as a *faulty circuit*.

Note that $\tilde{\mathcal{F}}$ can be empty and, if $\Pr_{\mathbb{D}\mathcal{F}^\infty}[\emptyset] > 0$, then the sampler can output the original circuit $C = C^\emptyset$. For the sake of simplicity, it still falls within the definition of a faulty circuit.

The original adversary model introduced by Feldtkeller et al. provides the adversary with a correct and a faulty output of the circuit C . For our purposes, we define the leakage of faults by the correctness of the output, which is a conservative but popular choice for fault security [5,19,40,41]. For that, we define a decoding or correction gadget for the context of faulty circuits:

G^D : The decoding gadget realizes a function such that, given an input with at most k bit faults, outputs a corrected result.

Then, we define the leakage by the correctness of the output of the faulty circuit. More formally, we define the *random fault leakage* $\mathcal{L}_{\tilde{\mathcal{F}}}(C, x)$ as the output of the random experiment, with

$$\begin{aligned} C^{\tilde{\mathcal{F}}} &\leftarrow \text{AssignFaultGates}(C, \mathbb{D}\mathcal{F}^\infty), \\ \mathcal{L}_{\tilde{\mathcal{F}}}(C) &\leftarrow \begin{cases} 0 & \text{if } \forall x : C(x) = G^D(C^{\tilde{\mathcal{F}}}(x)), \\ 1 & \text{else.} \end{cases} \end{aligned}$$

In contrast to $\mathcal{L}_{\mathcal{W}}(C, \text{enc}(x))$, the random fault leakage is not a distribution but, for each fault combination $\tilde{\mathcal{F}}$, a constant Boolean value. With this, we define the view of the adversary $\mathcal{A}_f$, after injecting a random fault into a circuit C , as $\mathcal{L}_{\tilde{\mathcal{F}}}(C)$. Note, that we focus on correction-based countermeasures here. However, detection-based countermeasures can be treated analogously by considering the result secure if a fault was detected correctly.

As with the general random probing model, this adversary model allows the modeling of a wide range of different adversarial capabilities [26]. For example, the set of fault combinations $\mathcal{F}^\infty$ can be set to register combinations with long computation paths to model clock glitches. Similarly, it can be set to a set of adjacent gates to model a laser attack. Similarly, a distribution of faults $\mathbb{D}\mathcal{F}^\infty$ with a small variance can be used to model an adversary with precise faulting capabilities, while a broader distribution can be used for a more dispersed fault behavior.

Security Definition. We extend the above adversary models by providing an appropriate definition for security. Intuitively, we say a fault combination leads to an insecure circuit behavior if there exists some input assignment that cannot be corrected at the output. This is a very conservative assumption, in that it assumes an adversary who can exploit every effective fault at the output to gain full knowledge of the secret, and is popular in the literature [5,19,40,41]. To account for the random behavior of our adversary model, we say a circuit is *random fault secure* if the probability that the adversary will get exploitable information is bounded by some (small) μ .

Definition 2 (General Random Fault Security). A circuit C is $(\mathbb{DF}^\infty, \mu)$ -random fault secure with respect to a decoding G^D if:

$$\Pr[\mathcal{L}_{\tilde{\mathcal{F}}}(C) = 1] \leq \mu,$$

where $\mathcal{L}_{\tilde{\mathcal{F}}}()$ is computed from the random experiment

$$C^{\tilde{\mathcal{F}}} \leftarrow \text{AssignFaultGates}(C, \mathbb{DF}^\infty).$$

In this definition, the decoding gadget required for the computation of the fault leakage $\mathcal{L}_{\tilde{\mathcal{F}}}(C)$ cannot be faulted by $\mathcal{A}_f$. This is symmetric to the encoding enc in random probing security, which is not probed, and can be justified by the fact that a fault in a final correction can only leak the output of the circuit. Note, however, that any correction implemented within the circuit is subject to faults in our model.

Relation to Random Fault Model. Dhooghe and Nikova proposed in [20] a *random fault model* that is inspired by the random probing model. Note that their model differs from our proposal in two key features: (i) They consider faults in wires allowing only the fault transformations **set**, **reset**, and **flip**. In contrast, we model faults by a transformation of gate functions which allows a wide range of possible fault scenarios, including **set**, **reset**, and **flip** [42]. (ii) They consider an adversary that has precise control over the location of the fault where the occurrence of the fault is randomly determined by an independent probability κ . In contrast, we allow an arbitrary distribution over fault combinations. Hence, different fault locations can be dependent and the fault type can be uncertain as well. This allows a wide range of possible adversarial scenarios [26]. Therefore, our new model is a generalization of the previously proposed random fault model.

3.3 General Random Combined Security

Adversary Model. We now introduce a model for an adversary that can both inject faults and place probes simultaneously. As such, the resulting *combined* adversary will have the capability to manipulate a random set of gates of a circuit. Then, both the exact values of a random subset of wires and the correctness of the circuit are leaked to the adversary. To formally capture the view of the adversary, we define the *random combined leakage* $\mathcal{L}_{\tilde{\mathcal{W}}, \tilde{\mathcal{F}}}(C, x)$ as the output of the random experiment

$$\begin{aligned} C^{\tilde{\mathcal{F}}} &\leftarrow \text{AssignFaultGates}(C, \mathbb{DF}^\infty), \\ \tilde{\mathcal{W}} &\leftarrow \text{LeakingWires}(C^{\tilde{\mathcal{F}}}, \mathbb{DW}^\infty), \\ \mathcal{L}_{\tilde{\mathcal{W}}, \tilde{\mathcal{F}}}(C, x) &\leftarrow \text{AssignWires}(C^{\tilde{\mathcal{F}}}, \tilde{\mathcal{W}}, x) \parallel \mathcal{L}_{\tilde{\mathcal{F}}}(C). \end{aligned}$$

Again, we assume a correction-based countermeasure for simplicity. When detection is used, it is important to include the detection flag in the random combined leakage (which then needs to be simulated alongside the leaking wires).

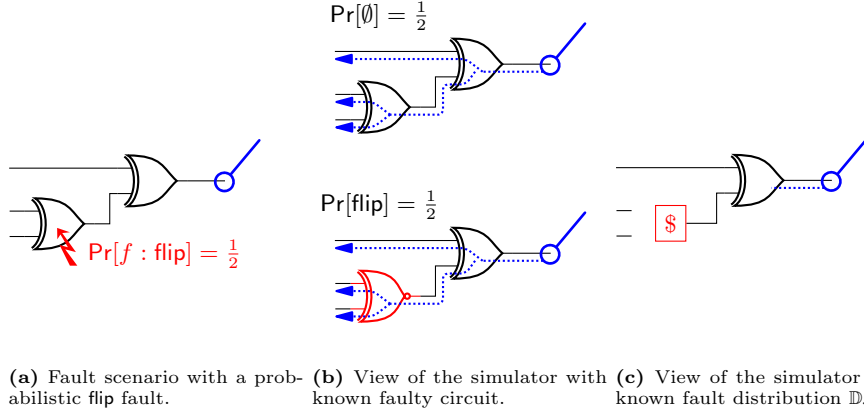


Fig. 1. Difference when knowing the injected fault versus knowing only the fault distribution. When knowing the fault the circuit is deterministic (for deterministic inputs) and a probe on the output propagates to all inputs. When only the fault distribution is known, the circuit is probabilistic (even with deterministic inputs) stopping the propagation of probes.

Considering the SCA aspect of CA, we require for security that the leaking wires $\tilde{\mathcal{W}}$ can be simulated without knowledge of the secret. However, for simulation, the adversarial knowledge of the faulty circuit makes a difference. Imagine the scenario in Figure 1, where a flip fault is injected into a xor chain with probability $\frac{1}{2}$. If the adversary knows the faulty circuit $C^{\tilde{\mathcal{F}}}$ (and, hence, $C^{\tilde{\mathcal{F}}}$ is given to the simulator) then all deterministic inputs to the xor chain are required to simulate the probe at the end in both cases (non-faulty/faulty). In contrast, if the adversary does not know $C^{\tilde{\mathcal{F}}}$ (and only the fault distribution $\mathbb{D}\tilde{\mathcal{F}}^\infty$ is given to the simulator) then the fault randomizes the intermediate value and the output of the final xor can be simulated by a uniform random value. Hence, the fault effectively works as a mask refreshing. For this, we introduce two different combined adversaries: one without and one with knowledge of the faulty circuit.

i) Unknown-Fault Random Combined Adversary. Our first adversary $\mathcal{A}_{uc}$ is the combination of $\mathcal{A}_p$ and $\mathcal{A}_f$ without knowledge of the randomly chosen fault combination $\tilde{\mathcal{F}}$. Specifically, the adversary has no access to the faulty circuit $C^{\tilde{\mathcal{F}}}$ and the corresponding simulator receives just the circuit C and the fault distribution $\mathbb{D}\tilde{\mathcal{F}}^\infty$ as input. Hence, the view of $\mathcal{A}_{uc}$ is defined by $\mathcal{L}_{\tilde{\mathcal{W}}, \tilde{\mathcal{F}}}(C, x)$ and the effects of fault injection and probing are interleaved, i.e., we analyze the circuit in Figure 1c.

ii) Known-Fault Random Combined Adversary. The second adversary $\mathcal{A}_{kc}$ is the combination of $\mathcal{A}_p$ and $\mathcal{A}_f$ with additional knowledge of the faulty circuit $C^{\tilde{\mathcal{F}}}$ (or equivalently, the selected fault combination $\tilde{\mathcal{F}}$). For this, the corresponding simulator has access to $C^{\tilde{\mathcal{F}}}$ for the simulation of the leaking wires. With this, the

view of $\mathcal{A}_{kc}$ is defined by $\mathcal{L}_{\tilde{\mathcal{W}}, \tilde{\mathcal{F}}}(C, x) \parallel \tilde{\mathcal{F}}$. Since the faulty circuit is known, we can analyze $\mathcal{L}_{\tilde{\mathcal{W}}}(C^{\tilde{\mathcal{F}}}, x)$ and $\mathcal{L}_{\tilde{\mathcal{F}}}(C)$ independently, i.e., after the fault injection, we only consider one of the two circuits in Figure 1b.

When comparing the two adversaries, it becomes apparent that $\mathcal{A}_{uc}$ is the more realistic adversary model for CA because, in a real-world circuit, the adversary usually does not know the exact effect of an injected fault. However, the uncertainty about the faults makes the analysis of combined security much more complex, due to the reciprocal effects of faults and probes. Fortunately, we can show that any circuit secure against $\mathcal{A}_{kc}$ is also secure against $\mathcal{A}_{uc}$. For this, we see $\mathcal{A}_{kc}$ mostly as a useful abstraction for analysis, allowing a clear path to security verification.

In this sense, we can make a further distinction in the knowledge an adversary has about the effect of fault injection. Specifically, we can separate the knowledge about the fault location and its effect. For this, we use the known-fault adversary $\mathcal{A}_{kc}$ in combination with a fault transformation to a probabilistic gate function $\mathbf{f}_g$, i.e., $\mathbf{f}_g$ has an internal random tape that impacts the output of the gate. While this pushes the analysis closer to $\mathcal{A}_{uc}$, it is important to evaluate the correctness of the circuit for all values of the random tape, i.e., $\mathcal{L}_{\tilde{\mathcal{F}}}(C) = 0$ if the output can be corrected for all inputs and random values of probabilistic gate functions.

Our model also allows us to go in the opposite direction by empowering $\mathcal{A}_{kc}$ with direct control over the injected fault. Hence, we can model a *chosen-fault* combined adversary as the adversary $\mathcal{A}_{kc}$ that can freely choose the fault distribution $\mathbb{D}\mathcal{F}^\infty$ of a (restrict) set of fault combinations $\mathcal{F}^\infty$. In most cases, this will lead to an adversary that directly chooses the location and transformation of the injected fault, since any uncertainty will reduce the advantage of the adversary, which removes the random nature of the model. Note, if the set $\mathcal{F}^\infty$ is restricted to the set of all gate combinations with up to k faults, this model is equivalent to the popular threshold fault model [1,5,32,41,48]. However, since the semantics of the security definition do not change (we do not care how the fault distribution is generated), we do not introduce a specific adversary model for this case.

Security Definition. We provide security definitions for the setting of combined adversaries. Intuitively, we say that a circuit is combined-secure if the view of the corresponding adversary can be simulated and the output can be corrected with high probability. For this, we introduce a new security parameter γ that represents the advantage of the adversary, i.e., the probability that the adversary gains some knowledge about the secret. In the combined setting, the adversary wins if either there exists some fault leakage ($\mathcal{L}_{\tilde{\mathcal{F}}}(C) = 1$) or the leaking wires in the faulty circuit cannot be simulated without knowledge of the secret. To overcome the dependencies between those two events, we only conduct the simulation of the leaking wires if there is no fault leakage. Hence, we get the following two parameters (where we usually choose the lowest possible value):

$$\mu \geq \Pr[\mathcal{L}_{\tilde{\mathcal{F}}}(C) = 1],$$

$$\epsilon \geq \Pr[\text{Sim}(C, \tilde{W}) \equiv \perp \mid \mathcal{L}_{\tilde{\mathcal{F}}}(C) = 0].$$

Here, μ is defined exactly as in Definition 2, while ϵ is the probability that the simulation fails knowing that $\mathcal{L}_{\tilde{\mathcal{F}}}(C) = 0$. We then express the advantage of the combined adversary by

$$\gamma \geq \mu + (1 - \mu)\epsilon.$$

To compute ϵ , we introduce a new subset of the fault combination $\mathcal{B} \subseteq \mathcal{F}^\infty$, such that $\mathcal{B}$ captures all fault combinations that always yield a result that can be corrected, i.e., $\mathcal{B} = \{\tilde{\mathcal{F}} \in \mathcal{F}^\infty \mid \mathcal{L}_{\tilde{\mathcal{F}}}(C) = 0\}$. Then, we define the distribution $\mathbb{D}\mathcal{B}$ as the scaled distribution $\mathbb{D}\mathcal{F}^\infty$ conditioned to the event $\mathcal{L}_{\tilde{\mathcal{F}}}(C) = 0$.

We start with the security definition for the adversary $\mathcal{A}_{uc}$, where the random combined leakage needs to be simulated. In a sense, this is the most natural definition as the adversary has no knowledge about the circuit transformation caused by the injected fault.

Definition 3 (Unknown-Fault Random Combined Security). *A circuit C is $(\mathbb{D}\mathcal{W}^\infty, \mathbb{D}\mathcal{F}^\infty, \gamma)$ -Unknown-Fault Random Combined Secure (RCS_{UF}) with respect to an value encoding enc and a error decoding G^D if there exists some $\mu, \epsilon \leq 1$ such that C is $(\mathbb{D}\mathcal{F}^\infty, \mu)$ -random fault secure with respect to G^D , there exists a simulator Sim such that for all inputs x :*

$$\text{Sim}(C, \mathbb{D}\mathcal{F}^\infty) \approx_\epsilon \mathcal{L}_{\mathbb{D}\mathcal{W}^\infty, \mathbb{D}\mathcal{B}}(C, \text{enc}(x)) \quad \text{and} \quad \mu + (1 - \mu)\epsilon \leq \gamma,$$

where $\mathcal{L}_{\mathbb{D}\mathcal{W}^\infty, \mathbb{D}\mathcal{B}}()$ is computed from the random experiment

$$\begin{aligned} \tilde{W} &\leftarrow \text{LeakingWires}(C, \mathbb{D}\mathcal{W}^\infty), \\ C^{\tilde{\mathcal{F}}} &\leftarrow \text{AssignFaultGates}(C, \mathbb{D}\mathcal{B}). \end{aligned}$$

While simple, this definition is difficult to analyze due to the interleaving of probes and faults within the simulator. Therefore, we provide a second security definition tailored to $\mathcal{A}_{kc}$. Because this adversary knows the faulty circuit $C^{\tilde{\mathcal{F}}}$, the interleaving of probes and faults is eliminated. Hence, the analysis gets simpler and we will later show that security in this model implies security with unknown faults.

Definition 4 (Known-Fault Random Combined Security). *A circuit C is $(\mathbb{D}\mathcal{W}^\infty, \mathbb{D}\mathcal{F}^\infty, \gamma)$ -Known-Fault Random Combined Secure (RCS_{KF}) with respect to an value encoding enc and an error decoding G^D if there exists some $\mu, \epsilon \leq 1$ such that C is $(\mathbb{D}\mathcal{F}^\infty, \mu)$ -random fault secure with respect to G^D , there exists a simulator Sim such that for all inputs x it holds that*

$$\begin{aligned} C^{\tilde{\mathcal{F}}} &\leftarrow \text{AssignFaultGates}(C, \mathbb{D}\mathcal{B}), \\ \text{Sim}(C^{\tilde{\mathcal{F}}}) &\approx_\epsilon \mathcal{L}_{\tilde{W}}(C^{\tilde{\mathcal{F}}}, x), \end{aligned}$$

and

$$\mu + (1 - \mu)\epsilon \leq \gamma,$$

where $\mathcal{L}_{\tilde{W}}()$ is computed from the random experiment

$$\tilde{W} \leftarrow \text{LeakingWires}(C^{\tilde{\mathcal{F}}}, \mathbb{D}\mathcal{W}^\infty).$$

Reduction between Security Definitions. It is evident that the adversary $\mathcal{A}_{kc}$ has more knowledge about the probed circuit structure than $\mathcal{A}_{uc}$. Hence, it seems reasonable that RCS_{UF} is the more general security notion and we can reduce its security to RCS_{KF} . In other words, if a circuit is RCS_{KF} , then it is always RCS_{UF} .

In particular, we can show that if there exists a simulator Sim_{KF} for the adversary $\mathcal{A}_{kc}$ then we can always construct a simulator Sim_{UF} for the adversary $\mathcal{A}_{uc}$ with at most the failure probability of Sim_{KF} . For that, consider that RCS_{KF} requires the simulation for a given but randomly chosen faulty circuit $C^{\tilde{\mathcal{F}}}$. Hence, we can compute the failure probability of Sim_{KF} as the sum of the failure probability for each faulty circuit weighted by the probability of that circuit to occur, i.e.,

$$\epsilon_{kf} = \sum_{\tilde{\mathcal{F}}} \Pr[\tilde{\mathcal{F}}] \epsilon_{kc}^{\tilde{\mathcal{F}}}, \quad (1)$$

where $\epsilon_{kc}^{\tilde{\mathcal{F}}}$ is the failure probability of Sim_{KF} given the faulty circuit $C^{\tilde{\mathcal{F}}}$. We can then construct a simulator that randomly selects a faulty circuit and calls Sim_{KF} for the wire simulation. Note, that standalone fault security is not affected by the choice of the adversary. We show a more formal argumentation below.

Theorem 1. *Let C be a circuit that is $(\mathbb{DW}^\infty, \mathbb{DF}^\infty, \gamma_{kc})$ - RCS_{KF} . Then C is $(\mathbb{DW}^\infty, \mathbb{DF}^\infty, \gamma_{uc})$ - RCS_{UF} with $\gamma_{uc} \leq \gamma_{kc}$.*

Proof. Let C be a $(\mathbb{DW}^\infty, \mathbb{DF}^\infty, \gamma_{kc})$ - RCS_{KF} circuit. Then, by definition of RCS_{KF} , C is $(\mathbb{DF}^\infty, \mu)$ -random fault secure and there exists a simulator Sim_{KF} with failure probability ϵ_{kc} for the probing leakage $\mathcal{L}_{\tilde{\mathcal{W}}}(C^{\tilde{\mathcal{F}}}, x)$ such that $\gamma_{kc} \geq \mu + (1 - \mu)\epsilon_{kc}$.

Now we construct a simulator Sim_{UF} for RCS_{UF} out of Sim_{KF} . This is possible because Sim_{UF} has to simulate the same events as Sim_{KF} since the set of fault combinations with $\mathcal{L}_{\tilde{\mathcal{F}}}(C) = 1$ remains untouched by the knowledge of the adversary. The simulator Sim_{UF} is constructed by the following two steps:

1. Call $C^{\tilde{\mathcal{F}}} \leftarrow \text{AssignFaultGates}(C, \mathbb{DB})$,
2. Return $\text{Sim}_{\text{KF}}(C^{\tilde{\mathcal{F}}}, \tilde{\mathcal{W}})$.

When this simulator does not fail it is a perfect simulation of the leaking wires because it draws $\tilde{\mathcal{F}}$ from the correct distribution and Sim_{KF} produces a perfect simulation for any faulty circuit $C^{\tilde{\mathcal{F}}}$ (if it does not fail). Given this, the failure probability of Sim_{UF} is given by $\epsilon_{uf} = \sum_{\tilde{\mathcal{F}}} \Pr[\tilde{\mathcal{F}}] \epsilon_{kc}^{\tilde{\mathcal{F}}}$, where $\epsilon_{kc}^{\tilde{\mathcal{F}}}$ is the failure probability of Sim_{KF} given the faulty circuit $C^{\tilde{\mathcal{F}}}$. Hence, we have $\epsilon_{uc} = \epsilon_{kc}$ (see Equation 1) for this simulator (however, there could be a simulator with a smaller failure probability). Together with $(\mathcal{F}^\infty, \mu)$ -random fault security of C , it follows that C is $(\mathbb{DW}^\infty, \mathbb{DF}^\infty, \gamma_{uc})$ - RCS_{UF} with $\gamma_{uc} \leq \gamma_{kc}$. $\square$

In the remainder of this paper, we will mostly focus on RCS_{KF} for the sake of simplicity.

Related Work on the Random Combined Model. In addition to the random fault model, Dhooghe and Nikova propose a combined version of the random probing and random fault model [20]. Their model does not leak the exact occurring fault to the adversary (similar to RCS_{UF}). However, since they use their version of the random fault model and the traditional random probing model, the combination has the same limitations as the individual models. Our model introduces a new security parameter γ that captures the overall advantage of the combined adversary. Notably, the parameters of our combined model can be chosen to align with the definitions provided by Dhooghe and Nikova. While they propose this model in an appendix, they do not perform any analysis or further investigations into it.

Related Work on the Combined Attacks. In the literature, several combined attacks have been proposed [15,43,44,45,46,50] that exploit both fault injection and side-channel leakage. These attacks leverage additional SCA leakage caused by conditional fault propagation or the disruption of masking schemes through fault injection. Our model captures these attacks, as simulating the corresponding probes is not possible without access to the secret, reflecting the inherent vulnerabilities. For instance, works [44, Section 4.2] and [46] describe combined attacks on gadgets involving logical AND operations protected by countermeasures like DOMREP [31]. The strategy typically involves injecting a fault into one share of a sensitive variable a , which is multiplied by all shares of another sensitive variable b . This causes a conditional fault propagation where effective faults propagate only if $b = 1$, and, hence, the observation of the effectiveness of the fault is sufficient to learn the secret b . With our new combined model, this attack can be captured using a single fault (set, reset, or bit-flip), and two probes placed in different replications or a single probe in the correction module that combines two replications. By probing the correction module, we can observe the fault’s propagation based on whether b is zero (when the fault does not propagate) or one (when it does), enabling the extraction of sensitive information about b . In response to this vulnerability, DOMREP-II [37] was introduced, which performs a masked correction such that a single probe cannot determine the fault’s effectiveness.

4 Compositional Notions

Analyzing entire circuits for their security is often prohibitively complex. As a result, the research community focuses on the construction of so-called *gadgets*, i.e., small circuits implementing a small function (often single binary operations) in a secure manner such that security is guaranteed even under composition [7,10,13,19,27,40]. To abstractly argue about the security via composition, we first define a *composability notion*, which defines the properties a gadget must fulfill, usually, by restricting the propagation of leakage or faults. Second, we outline and prove the conditions of composition, i.e., how the gadgets can be securely combined, in a *composition theorem*.

For composition in the general random probing model, we can extend a notion from the ϵ -random probing model to arbitrary leakage distributions. In particular, we extend the notion of Random Probing Composability (RPC) [10] such that there is an arbitrary but independent leakage distribution for each gadget. This independence of leakage distributions between gadgets is required to preserve the advantages of the gadget-based approach, namely the independent analysis of gadgets. The definition and compositional properties are straightforward extensions of the original statements. In particular, the random set of leaking wires is extended by a bounded set of output wires, which together are only allowed to provide information about a bounded set of input shares. Then, the failure probability ϵ is determined by the maximum failure probability over all possible leaking output combinations. We provide more details in Appendix A and continue with composition in the general random fault model.

4.1 Composition in the Random Fault Model

In contrast to the random probing model, to the best of our knowledge, there exists no prior work on composition in the random fault model. Due to the duality of probes and faults, we can adapt the notation from the random probing to the random fault case. However, instead of adding probes on outputs we now consider additional faults on inputs and check for correctness not simulation.

To properly argue about the composition we need to specify the used countermeasure (similar to masking for probing). Hence, in the following, we will only consider circuits secured by simple repetition against a fault adversary. In particular, we will use $2k + 1$ repetitions such that a majority vote can be used for the correction of up to k faults. Then, for composition, we consider an adversary that can inject a random fault combination into the gadget where additionally a tuple $\mathcal{I}'$ of arbitrarily but bounded sets of inputs are potentially manipulated by faults. More specifically, for each input with index i , there is a bounded set $\mathcal{I}'_i$ that contains the replication indices of inputs that may be affected by a fault. Here, we consider each replication in $\mathcal{I}'_i$ to be a unique and independent input, to account for all possible distribution changes due to a fault in previous parts of the composed circuit. Hence, a fault on an input wire can be seen as an arbitrary change in the value distribution over $\mathbb{F}_2$. Then, a gadget is composable, if the output can be corrected with high probability.

To formally define our notion of composition in the presence of a random fault adversary, we start by defining Restricted Random Fault Composability (RRFC), which states the required property for a fixed tuple of input faults $\mathcal{I}'$.

Definition 5 (Restricted Random Fault Composability). *Let $n = 2k + 1$ and $\mathcal{I}' = (\mathcal{I}'_i)_{i \in [h]}$ be a tuple of sets such that $\mathcal{I}'_i \subseteq [n]$, for all i : $|\mathcal{I}'_i| \leq k$. A gadget $G : (\mathbb{F}_2^n)^h \rightarrow (\mathbb{F}_2^n)^m$ is $(\mathcal{I}', k, \mathbb{D}\mathcal{F}^\infty, \mu)$ -RRFC if there exists a deterministic*

algorithm **ReplicationSelect** such that the random experiment

$$\begin{aligned} G^{\tilde{\mathcal{F}}} &\leftarrow \text{AssignFaultGates}(G, \mathbb{DF}^\infty), \\ \mathcal{O}' &:= (\mathcal{O}'_1, \dots, \mathcal{O}'_m) \leftarrow \text{ReplicationSelect}(G^{\tilde{\mathcal{F}}}, \mathcal{I}') \\ \mathcal{L}_{\mathcal{I}', \tilde{\mathcal{F}}}(G) &\leftarrow \begin{cases} 0 & \text{if } \forall i \leq m : |\mathcal{O}'_i| \leq k, \\ 1 & \text{else.} \end{cases} \end{aligned}$$

yields

$$\Pr[\mathcal{L}_{\mathcal{I}', \tilde{\mathcal{F}}}(G) = 1] \leq \mu$$

$$G^{\tilde{\mathcal{F}}}(x')|_{\overline{\mathcal{O}'}} \equiv G(x)|_{\overline{\mathcal{O}'}}$$

for all inputs $x \in \mathbb{V}_n^h$ and faulty inputs $x' \in (\mathbb{F}_2^n)^h$ with $x|_{\overline{\mathcal{I}'}} = x'|_{\overline{\mathcal{I}'}}$.

Then, a gadget supports Random Fault Composability (RFC) with some bound μ if for all possible tuples $\mathcal{I}'$ the RRFC failure probability is bounded by μ .

Definition 6 (Random Fault Composability). A gadget $G : (\mathbb{F}_2^n)^h \rightarrow (\mathbb{F}_2^n)^m$ with $n = 2k + 1$ is $(k, \mathbb{DF}^\infty, \mu)$ -RFC if for all tuples of sets $\mathcal{I}' = (\mathcal{I}'_i)_{i \in [h]}$ with $\forall i : \mathcal{I}'_i \subseteq [n]$ and $|\mathcal{I}'_i| \leq k$ the gadget G is $(\mathcal{I}', k, \mathbb{DF}^\infty, \mu_{\mathcal{I}'})$ -RRFC and $\max_{\mathcal{I}'} \{\mu_{\mathcal{I}'}\} \leq \mu$.

While we split the definition of RFC into two parts (for reasons that become apparent when we look at composition under a combined adversary) it is easy to see that there is a close symmetry with the definition of General Random Probing Composability (GRPC). Where GRPC goes through all possible tuples of output probes $\mathcal{O}$, RFC goes through all possible tuples of input faults. Where GRPC restricts the number of shares per input the simulator can use for a successful simulation, RFC restricts the number of replications per output that can be affected by a fault.

We now show that gadgets supporting RFC can be composed arbitrarily. Under the assumption of mutually independent fault distributions for each gadget, the failure probability increases linearly with the number of gadgets in the circuit.

Theorem 2. Let C be a circuit constructed by composition of $(k, \mathbb{DF}_i^\infty, \mu_i)$ -RFC gadgets G_i , for $i \in \{1, \dots, |C|\}$. Then, C is $(\prod_{i=1}^{|C|} \mathbb{DF}_i^\infty, 1 - \prod_{i=1}^{|C|} (1 - \mu_i))$ -random fault secure.

Proof. Let the faulty circuit be $C^{\tilde{\mathcal{F}}} \leftarrow \text{AssignFaultGates}(C, \prod_{i=1}^{|C|} \mathbb{DF}_i^\infty)$, with $\tilde{\mathcal{F}}$ the set of faults selected with $\Pr_{\prod_i \mathbb{DF}_i^\infty}[\tilde{\mathcal{F}}]$. We can divide $\tilde{\mathcal{F}}$ into $|C|$ disjoint fault sets $\tilde{\mathcal{F}}_i \subseteq \tilde{\mathcal{F}}$ such that $\tilde{\mathcal{F}}_i$ belongs to G_i and is selected with $\Pr_{\mathbb{DF}_i^\infty}[\tilde{\mathcal{F}}_i]$.

We go iteratively through the gadgets, starting with the gadgets only connected to the inputs of C . Let G_i be such a gadget. Then, by definition of $(k, \mathbb{DF}_i^\infty, \mu_i)$ -RFC, the gadget G_i is $(\mathcal{I}'_{G_i}, k, \mathbb{DF}_i^\infty, \mu_i)$ -RRFC for the tuple of empty sets $\mathcal{I}'_{G_i} = (\mathcal{I}'_{G_i, j} = \emptyset)_{j \in [h]}$. Hence, there exists a tuple of sets $\mathcal{O}'_{G_i}$ such that $\Pr[\mathcal{L}_{\mathcal{I}'_{G_i}, \tilde{\mathcal{F}}}(G) = 1] \leq \mu$ and $G_i^{\tilde{\mathcal{F}}_i}(x)|_{\overline{\mathcal{O}'_{G_i}}} \equiv G_i(x)|_{\overline{\mathcal{O}'_{G_i}}}$.

We continue with the child gadgets, i.e., gadgets that have inputs connected to outputs of the just handled gadgets and (potentially) inputs of the circuit. Let G_j be such a gadget. Again, we can create a tuple of sets $\mathcal{I}'_{G_j}$ out of the respective tuples $\mathcal{O}'_{G_i}$ of the parent gadgets G_i . Because of $(k, \mathbb{DF}_j^\infty, \mu_j)$ -RFC there exists tuple of sets $\mathcal{O}'_{G_j}$ such that $\Pr[\mathcal{L}_{\mathcal{I}'_{G_j}, \tilde{\mathcal{F}}}(G) = 1] \leq \mu$ and $G_j^{\tilde{\mathcal{F}}}(x)|_{\overline{\mathcal{O}'_{G_j}}} \equiv G_j(x)|_{\overline{\mathcal{O}_{G_j}}}$. We repeat this process until we reach the outputs of C .

Since we have $n = 2k+1$ replications, we can construct a decoding gadget G^D for C by computing the majority of the output wires $w_i^1, \dots, w_i^n$ for all $i \in [m]$. This decoding gadget will correct an output value as long as the number of faulty replications is smaller or equal to k , which is always true if none of the gadgets G_i , for $i = 1, \dots, |C|$, fail with respect to RFC. The probability that at least one of $|C|$ gadgets fail is $1 - \prod_{i=1}^{|C|} (1 - \mu_i)$. Hence, $\Pr[C(x) \equiv G^D(C^{\tilde{\mathcal{F}}}(x))] \leq 1 - \prod_{i=1}^{|C|} (1 - \mu_i)$, which shows random fault security of C . $\square$

Note, that the above composition loses some tightness in μ because a failure of RFC in one gadget does not necessarily mean that the entire circuit is insecure, e.g., if some faults cancel each other out in a later gadget.

Relation to Fault-Secure Composition. To the best of our knowledge, this is the first work establishing a compositional property in the random fault model. However, in the threshold fault model [32], where an adversary can place up to k faults, composition is already discussed and notions usually have a strong symmetry to notions in the Ishai-Sahai-Wagner (ISW) probing model. In particular, (Strong) Non-Accumulation ((S)NA) [19] (later refined to Fault (Strong) Non-Interference (F-(S)NI) [40]) restricts the number of faults that can propagate to the output of a gadget, while Fault-Isolating Non-Interference (FINI) [27] limits fault propagation within so-called redundancy domains. Both variants ensure that the number of faults in the replication of a single value does not exceed the threshold that allows correction (detection) of faults. In this sense, our proposed notion has some similarity with (S)NA/F-(S)NI, in that it restricts the number of output replications that are allowed to be affected by a fault. However, similar to the contrast between (G)RPC and (Strong) Non-Interference ((S)NI), the number of allowed faulty outputs is not dependent on the amount of injected faults. Also, we use the same symmetry between faults and probes for the definition of our notion of composition.

4.2 Composition in the Random Combined Model

Finally, we provide a compositional statement for the random combined model, i.e., under an adversary with both faulting and probing capabilities. Here, we consider composition under an adversary that knows the injected fault and use the reduction from Section 3.3 for the adversary with unknown faults. We leave the tighter compositional statement in the setting with unknown faults for future work. In principle, the following notion is a combination of the two notions for the individual cases, however, with subtle differences.

First, we count faults in inputs and outputs per input and output share. More specifically, the tuple of indices for potentially faulty inputs has now a set for each share of each input, i.e., $\mathcal{I}' = ((\mathcal{I}'_{i,j})_{i \in [s]})_{j \in [h]}$, where j is the index of the input and i the share index. The same holds for the tuple of potential faulty outputs $\mathcal{O}' = ((\mathcal{O}'_{i,j})_{i \in [s]})_{j \in [m]}$, which is constructed by **ReplicationSelect**. Hence, we bound the number of allowed faults per input and output share by k .

Second, and in contrast, probes on outputs are extended to all replications of the probed value⁴. Hence, the tuple $\mathcal{O} = (\mathcal{O}_i)_{i \in [m]}$, where each $\mathcal{O}_i$ contains the probed share indices of the i 'th output, does not change. However, the meaning of $j \in \mathcal{O}_i$ changes in so far as now all replications of the j 'th share of the i 'th output need to be simulated. This is necessary to allow for a wide range of gadget implementations where there are potential interdependencies between different replications, e.g., via a correction module, that allows probe propagation across replications [25]. Similarly, the simulator gets access to all replications of the shares indicated in the tuple $\mathcal{I} = (\mathcal{I}_i)_{i \in [h]}$.

Third, we define the failure probability for probe simulation $\epsilon_{\mathcal{I}'}$ in dependence on the tuple of faulty-input indices $\mathcal{I}'$. This is analogous to the definition of RRFC (Definition 5) and allows us to iterate over all possible tuples $\mathcal{I}'$ for our combined composition. In particular, this enables a precise definition of the conditions for Random Probing Composition under Faults (RPCUF) under a given tuple of potentially faulty inputs $\mathcal{I}'$ and a random fault combination $\tilde{\mathcal{F}} \in \mathcal{F}^\infty$.

Definition 7 (Random Probing Composition under Faults). *Let $\mathcal{I}' = ((\mathcal{I}'_{i,j})_{i \in [s]})_{j \in [h]}$ be a tuple such that $\forall i, j : \mathcal{I}'_{i,j} \subseteq [n]$ and $|\mathcal{I}'_{i,j}| \leq k$. A gadget $G : ((\mathbb{F}_2^n)^s)^h \rightarrow ((\mathbb{F}_2^n)^s)^m$ with $n = 2k + 1$ is $(\mathcal{I}', \mathbb{DF}^\infty, d, \mathbb{DW}^\infty, \epsilon)$ -RPCUF if there exists a deterministic algorithm **ShareSelect** and a probabilistic simulator **Sim** such that for all faulty inputs $x' \in ((\mathbb{F}_2^n)^s)^h$, for which there exists an $x \in (\mathbb{V}_n^s)^h$ with $x|_{\overline{\mathcal{I}'}} = x'|_{\overline{\mathcal{I}'}}$, and every tuple of sets $\mathcal{O} = (\mathcal{O}_1, \dots, \mathcal{O}_m)$, with $\forall i : \mathcal{O}_i \subseteq [s]$ and $|\mathcal{O}_i| \leq d$, the random experiment*

$$\begin{aligned} G^{\tilde{\mathcal{F}}} &\leftarrow \text{AssignFaultGates}(G, \mathbb{DF}^\infty) \\ \tilde{\mathcal{W}} &\leftarrow \text{LeakingWires}(G^{\tilde{\mathcal{F}}}, \mathbb{DW}^\infty) \\ \mathcal{I} &:= (\mathcal{I}_1 \dots \mathcal{I}_h) \leftarrow \text{ShareSelect}(\tilde{\mathcal{W}}, \mathcal{O}) \\ \text{out} &\leftarrow \text{Sim}(x'|_{\mathcal{I}}, \mathcal{I}, \mathcal{I}', \mathcal{O}, \tilde{\mathcal{W}}) \end{aligned}$$

yields

$$\begin{aligned} \Pr[|\mathcal{I}_1| > d \vee \dots \vee (|\mathcal{I}_h| > d)] &\leq \epsilon \\ \text{out} &\equiv (\text{AssignWires}(G^{\tilde{\mathcal{F}}}, \tilde{\mathcal{W}}, x'), y'|_{\mathcal{O}}) \end{aligned}$$

with $y' \leftarrow G^{\tilde{\mathcal{F}}}(x')$.

⁴ The same procedure should be used when analyzing replicated circuits for stand-alone GRPC.

Fourth, and similar to Section 3.3, we only check for side-channel security if the gadget is fault secure to keep the failure probabilities for probe simulation $\epsilon_{\mathcal{I}'}$ and correctness $\mu_{\mathcal{I}'}$ independent. For this, we again introduce a subset of fault combinations $\mathcal{B}_{\mathcal{I}'} \subseteq \mathcal{F}^\infty$, such that $\mathcal{B}_{\mathcal{I}'}$ captures all fault combinations that, in combination with potential faults on the inputs with indices in $\mathcal{I}'$, lead to a gadget output that can be corrected, i.e., $\mathcal{B}_{\mathcal{I}'} = \{\tilde{\mathcal{F}} \in \mathcal{F}^\infty \mid \mathcal{L}_{\mathcal{I}', \tilde{\mathcal{F}}}(G) = 0\}$. Note, that we define this set to be dependent on the tuple $\mathcal{I}'$. The corresponding distribution $\mathbb{D}\mathcal{B}_{\mathcal{I}'}$ is defined as the scaled distribution $\mathbb{D}\mathcal{F}^\infty$ conditioned on the event $\mathcal{L}_{\mathcal{I}', \tilde{\mathcal{F}}}(G) = 0$.

With this, we say that a gadget supports Known-Fault Random Combined Composability (RCC_{KF}) if, for any tuple $\mathcal{I}'$, the gadget supports RRFC and RPCUF with negligible advantage for the adversary. To again compute a unified failure probability we chose the maximum combined failure probability (i.e., the probability that either RRFC fails or RPCUF fails under the condition that RRFC holds) over all tuples $\mathcal{I}'$.

Definition 8 (Known-Fault Random Combined Composability). *A gadget $G : ((\mathbb{F}_2^n)^s)^h \rightarrow ((\mathbb{F}_2^n)^s)^m$ is $(d, k, \mathbb{D}\mathcal{W}^\infty, \mathbb{D}\mathcal{F}^\infty, \gamma)$ - RCC_{KF} if for all tuples of sets $\mathcal{I}' = ((\mathcal{I}'_{i,j})_{i \in [s]})_{j \in [h]}$, such that $\forall i, j : \mathcal{I}'_{i,j} \subseteq [n]$ and $|\mathcal{I}'_{i,j}| \leq k$, there exists some $\mu_{\mathcal{I}'}, \epsilon_{\mathcal{I}'} \leq 1$ such that the gadget G is $(\mathcal{I}', k, \mathbb{D}\mathcal{F}^\infty, \mu_{\mathcal{I}'})$ -RRFC and $(\mathcal{I}', \mathbb{D}\mathcal{B}_{\mathcal{I}'}, d, \mathbb{D}\mathcal{W}^\infty, \epsilon_{\mathcal{I}'})$ -RPCUF and it holds that $\max_{\mathcal{I}'} \{\mu_{\mathcal{I}'} + (1 - \mu_{\mathcal{I}'})\epsilon_{\mathcal{I}'}\} \leq \gamma$.*

Under this definition, we can compose any RCC_{KF} gadgets as long as each output of a gadget is only used once as input to another gadget. This restriction comes from the composition under probes. Then, under the assumption of mutually independent fault and probing distributions per gadget, the combined failure probability increases linearly in the number of gadgets in the circuit.

Theorem 3. *Let C be a circuit constructed by composition of gadgets G that are $(d, k, \mathbb{D}\mathcal{W}_i^\infty, \mathbb{D}\mathcal{F}_i^\infty, \gamma_i)$ - RCC_{KF} , for $i \in \{1, \dots, |C|\}$, such that each output of G_i is used as input of at most one other gadget G_j or as output of C . Then, C is*

$$\left(\prod_{i=1}^{|C|} \mathbb{D}\mathcal{W}_i^\infty, \prod_{i=1}^{|C|} \mathbb{D}\mathcal{F}_i^\infty, 1 - \prod_{i=1}^{|C|} (1 - \gamma_i) \right)\text{-RCS}_{\text{KF}}.$$

Intuitively, the proof follows the lines of the compositional statements for stand-alone probing and faulting. Specifically, we first go from inputs to outputs through the circuit to construct the respective tuples of potentially faulty inputs to each gadget to show composition under faults. Then, we go backward, i.e., from outputs to inputs, through the gadgets and use the fact that the set of faulty inputs is bounded in case of a fault combination that can be corrected to show random probing security under faults.

Proof. Let the faulty circuit be $C^{\tilde{\mathcal{F}}} \leftarrow \text{AssignFaultGates}(C, \prod_{i=1}^{|C|} \mathbb{D}\mathcal{F}_i^\infty)$, with $\tilde{\mathcal{F}}$ the set of faults selected with $\Pr_{\prod_i \mathbb{D}\mathcal{F}_i^\infty}[\tilde{\mathcal{F}}]$. We can divide $\tilde{\mathcal{F}}$ into $|C|$ disjoint fault sets $\tilde{\mathcal{F}}_i \subseteq \tilde{\mathcal{F}}$ such that $\tilde{\mathcal{F}}_i$ belongs to G_i and is selected with $\Pr_{\mathbb{D}\mathcal{F}_i^\infty}[\tilde{\mathcal{F}}_i]$.

Further, let $\tilde{\mathcal{W}}$ be the set of leaking wires of C selected with $\Pr_{\prod_i \mathbb{D}\mathcal{W}_i^\infty}[\tilde{\mathcal{W}}]$. We can divide $\tilde{\mathcal{W}}$ into $|C|$ disjoint parts $\tilde{\mathcal{W}}_i \subseteq \tilde{\mathcal{W}}$, each belonging to the gadget G_i such that each $\tilde{\mathcal{W}}_i$ was selected with $\Pr_{\mathbb{D}\mathcal{W}_i^\infty}[\tilde{\mathcal{W}}_i]$.

Since each gadget G_i is $(\mathcal{I}', k, \mathbb{D}\mathcal{F}^\infty, \mu_{\mathcal{I}'})$ -RRFC for all tuples $\mathcal{I}'$ it follows with Theorem 2 that the circuit is random fault secure with $\mu = 1 - \prod_{i=1}^{|C|} (1 - \mu_i)$.

Let $\mathcal{B} = \{\tilde{\mathcal{F}} \mid \tilde{\mathcal{F}} = \bigcup_{i=1}^{|C|} \tilde{\mathcal{F}}_i, \forall \mathcal{I}'_{G_i} : \mathcal{L}_{\mathcal{I}'_{G_i}, \tilde{\mathcal{F}}_i}(G_i) = 0\}$ be the set of faults considered secure in Theorem 2, i.e., all fault combinations that can be corrected at the output of the respective gadgets. We denote by $\mathcal{B}_{\mathcal{I}'_{G_i}} = \{\tilde{\mathcal{F}}_i \in \mathcal{F}^\infty \mid \mathcal{L}_{\mathcal{I}'_{G_i}, \tilde{\mathcal{F}}_i}(G_i) = 0\} \subseteq \mathcal{B}$ the set of fault combinations that can be corrected in a gadget G_i under faults in the input indices in $\mathcal{I}'_{G_i}$, with $\forall j : |\mathcal{I}'_{G_i, j}| \leq k$.

We now go backward through the circuit, starting with gadgets connected to the outputs of C , considering only faults $\tilde{\mathcal{F}} \in \mathcal{B}$. Let G_i be a gadget only connected to outputs of C . By RPCUF of G_i , we can construct a simulator Sim_{G_i} that requires a subset of inputs (defined by the tuple $\mathcal{I}_{G_i}$) for the simulation of the wires in $\tilde{\mathcal{W}}_i$ under the faults $\tilde{\mathcal{F}}_i$ and with faulty inputs with indices in $\mathcal{I}'_{G_i}$. Let the failure probability of Sim_{G_i} be ϵ_i .

We continue with the parent gadgets, i.e., gadgets that have outputs connected to the inputs of just handled gadgets and (potentially) outputs of the circuit. Let G_j be such a gadget. By RPCUF of G_j , we can construct a simulator Sim_{G_j} that requires a subset of inputs (defined by the tuple $\mathcal{I}_{G_j}$) for the simulation of the wires in $\tilde{\mathcal{W}}_j$ and the output wires defined by the tuples $\mathcal{I}_{G_i}$ of the child gadgets G_i under the faults $\tilde{\mathcal{F}}_j$ and with faulty inputs with indices in $\mathcal{I}'_{G_j}$. In particular, each gadget output with index ℓ is only used once in the circuit and we use the corresponding set $\mathcal{I}_{G_i, \ell'}$ of the child gadget as $\mathcal{O}_{G_j, \ell}$. Again, we denote the failure probability of Sim_{G_i} by ϵ_j . We repeat this process until we reach the inputs of C .

Given this, we can construct the simulator Sim for C by composition of the gadget simulators Sim_{G_i} . The failure probability ϵ of Sim is the probability that at least one simulator Sim_{G_i} fails, i.e., $\epsilon = 1 - \prod_{i=1}^{|C|} (1 - \epsilon_i)$. Therefore, with the random fault security advantage $\mu = 1 - \prod_{i=1}^{|C|} (1 - \mu_i)$, we have

$$\begin{aligned} \mu + (1 - \mu)\epsilon &= 1 - \prod_{i=1}^{|C|} (1 - \mu_i) + \prod_{i=1}^{|C|} (1 - \mu_i) \left(1 - \prod_{i=1}^{|C|} (1 - \epsilon_i)\right) \\ &= 1 - \prod_{i=1}^{|C|} (1 - \mu_i)(1 - \epsilon_i) \leq 1 - \prod_{i=1}^{|C|} (1 - \gamma_i), \end{aligned}$$

with $\gamma_i = \mu_i + (1 - \mu_i)\epsilon_i$ being the combined failure probability of each gadget G_i . It follows $(\prod_{i=1}^{|C|} \mathbb{D}\mathcal{W}_i^\infty, \prod_{i=1}^{|C|} \mathbb{D}\mathcal{F}_i, 1 - \prod_{i=1}^{|C|} (1 - \gamma_i))$ -RCS_{KF} of C . $\square$

Similar to Theorem 2, the above composition loses some tightness by only considering fault combinations that can be corrected after each gadget. While the set $\mathcal{B}$ gets bigger, and hence ϵ increases, when considering all fault combinations that can be corrected at the output of C , the overall γ gets smaller. The reason

is that the respective fault combination is entirely captured in μ of the above argument, however, it would only be partially captured (multiplied by some $\epsilon \leq 1$) when considering the tighter definition of $\mathcal{B}$. Hence, the provided γ is indeed an upper bound of the combined failure probability.

Relation to Combined-Secure Composition. As with RFC, to the best of our knowledge, this is the first work discussing compositional properties in the context of the random combined model. However, again in the threshold model, several compositional notions were discussed. Most of them are a combination of compositional notions for probing and faulting, respectively. In this respect, (Strong) Non-Interference Non-Accumulation ((S)NINA) [19] (later refined to Combined (Strong) Non-Interference (C-(S)NI) [40]) is the combination of (S)NA with (S)NI, and Combined-Isolating Non-Interference (CINI) [27] is the combination of FINI and Probe-Isolating Non-Interference (PINI), respectively. In the context of polynomial masking, Berndt et al. [12] coined the notion of fault-resilient (S)NI, which is the usual (S)NI notion that is invariant to fault injection. Our proposed compositional notation has some similarities with (S)NINA/C-(S)NI in that it is a combination of stand-alone faulting and probing composition and the underlying stand-alone notions relate to the respective stand-alone notions of (S)NA and (S)NI.

An interesting direction for future research into the composition in the random combined model is the notion of expansion (as considered in the random probing model) and the investigation of gadgets where the random probing security is invariant to faults.

5 Automatic Verification of Protected Implementations

We implement the verification of the new random combined security notion (and its composability variants) for cryptographic circuits by extending the verification tools VERICA [40] and IronMask [11]. At the state of the art, VERICA can be employed for combined hardware security verification considering the glitch-extended probing model [24] and the zeta fault-injection model [42], while IronMask is able to efficiently verify various (random, glitch-extended) probing security notions by relying on an algebraic characterization for specific gadgets.

In our work, we first aim to establish a common foundation for the practical verification of circuits under the new security models. Subsequently, we provide detailed insights into our extensions, denoted as VERICA⁺⁵ and IronMask⁺⁶, enabling the verification of combined security properties in cryptographic circuits.

⁵ The code for VERICA⁺ can be accessed at

<https://github.com/Chair-for-Security-Engineering/VERICA/tree/verica%2B>.

⁶ The code for IronMask⁺ can be accessed at <https://github.com/CryptoExperts/IronMask>.

5.1 Verification of the Generalized Security Models

We provide explicit formulas and practical verification choices for the computation of the simulation and correction failure probabilities in the general random probing model, the general random fault model, and the known-fault random combined security. These then serve as a basis for the extensions **VERICA**⁺ and **IronMask**⁺. For our implementations, we assume independent leakage and fault probabilities (p_w and q_f) for wires w and faults f to simplify the specification.

General Random Probing Security. The probability of a leaking wire combination $\tilde{\mathcal{W}}$ is the product of leakage probabilities of each wire $w \in \tilde{\mathcal{W}}$ times the product of $(1 - p_{w'})$ of the remaining wires $w' \in \mathcal{W} \setminus \tilde{\mathcal{W}}$. Consequently, let ϵ be the simulation failure probability such that

$$\epsilon = \sum_{i=1}^{|\mathcal{W}|} \sum_{\tilde{\mathcal{W}} \in \mathcal{W}_{\#i}^\infty} \prod_{w \in \tilde{\mathcal{W}}} p_w \prod_{w' \in \mathcal{W} \setminus \tilde{\mathcal{W}}} (1 - p_{w'}), \quad (2)$$

where $\mathcal{W}_{\#i}^\infty \subseteq \mathcal{W}^\infty$ is the set of wire combinations of exactly i wires that lead to a simulation failure, i.e., $\tilde{\mathcal{W}}_{\#i}^\infty = \{\tilde{\mathcal{W}} \in \mathcal{W}^\infty \mid |\tilde{\mathcal{W}}| = i \text{ and } \text{Sim}(C, \tilde{\mathcal{W}}) = \perp\}$. The above equation generalizes the computation of the simulation failure probability in the random probing security definition from Belaïd et al. [10]. Precisely, Equation 2 is only equivalent to the latter if we consider that all wires leak with the same probability p .

Practically, a circuit may be too large for exhaustively checking all wire combinations. Therefore, **VERICA**⁺ and **IronMask**⁺ compute the outer sum of Equation 2 only up to a threshold α , i.e., for $i \in [1, \alpha]$. For any combinations of more than α wires, we can either consider that (1) they do not result in a simulation failure ($\forall i \in]\alpha, |\mathcal{W}|], \mathcal{W}_{\#i}^\infty = \emptyset$) to obtain a lower bound $\epsilon_{\min}$, or (2) they all lead to a simulation failure ($\forall i \in]\alpha, |\mathcal{W}|], \tilde{\mathcal{W}}_{\#i}^\infty = \{\tilde{\mathcal{W}} \in \mathcal{W}^\infty \mid |\tilde{\mathcal{W}}| = i\}$) to obtain an upper bound $\epsilon_{\max}$. Those lower and upper bound computations follow the same method used for random probing security by Belaïd et al. in [10].

General Random Fault Security. The probability of a fault combination $\tilde{\mathcal{F}}$ is the product of the individual fault probabilities of the said combination times the product of $(1 - q_{f'})$ for all faults f' not present in it. Let μ be the correction failure probability defined by

$$\mu = \sum_{i=1}^{|\mathcal{F}|} \sum_{\tilde{\mathcal{F}} \in \mathcal{F}_{\#i}^\infty} \prod_{f \in \tilde{\mathcal{F}}} q_f \prod_{f' \in \mathcal{F} \setminus \tilde{\mathcal{F}}} (1 - q_{f'}), \quad (3)$$

where $\mathcal{F}_{\#i}^\infty \subseteq \mathcal{F}^\infty$ is the set of fault combinations of exactly i faults that cannot be corrected (such that $\tilde{\mathcal{F}} \in \mathcal{F}_{\#i}^\infty$ iff $\mathcal{L}_{\tilde{\mathcal{F}}}(C) = 1$).

For practical reasons, Equation 3 is computed for up to β faults during security verification; the outer sum thus reduces to $\sum_{i=1}^\beta$. Similarly to the previous

models, we derive the lower bound $\mu_{\min}$ and the upper bound $\mu_{\max}$ of μ by assuming that any combination of more than β faults can be, respectively, corrected or not.

Known-Fault Random Combined Security. For simplicity, we restrict ourselves to RCS_{KF} (cf. Definition 4) in the practical tool implementations. We then rely on the discussed security reduction for RCS_{UF} (cf. Section 3.3) and leave room for a tighter RCS_{UF} security analysis for future work. That said, three parameters are reported for RCS_{KF} :

1. the correction failure probability μ as described in Equation 3,
2. the known-fault simulation failure probability ϵ_{kf} if fault combinations can be corrected, such that

$$\epsilon_{kf} = \frac{1}{1-\mu} \sum_{i=1}^{|\mathcal{F}|} \sum_{\tilde{\mathcal{F}} \in \mathcal{B}_{\#i}} \epsilon_{kc}^{\tilde{\mathcal{F}}} \prod_{f \in \tilde{\mathcal{F}}} q_f \prod_{f' \in \mathcal{F} \setminus \tilde{\mathcal{F}}} (1 - q_{f'}), \quad (4)$$

where $\mathcal{B}_{\#i}$ is the set of fault combinations of i faults that can be corrected and $\epsilon_{kc}^{\tilde{\mathcal{F}}}$ is the known-corrected-fault simulation failure probability (Equation 2).

The above formula is obtained by observing that for any $\tilde{\mathcal{F}} \in \mathcal{B}$, we have $\Pr_{\mathbb{D}\mathcal{B}}[\tilde{\mathcal{F}}] = \frac{1}{1-\mu} \Pr_{\mathbb{D}\mathcal{F}^\infty}[\tilde{\mathcal{F}}] = \frac{1}{1-\mu} \prod_{f \in \tilde{\mathcal{F}}} q_f \prod_{f' \in \mathcal{F} \setminus \tilde{\mathcal{F}}} (1 - q_{f'})$.

3. and the advantage of the combined adversary γ_{kc} :

$$\gamma_{kc} = \mu + \epsilon_{kf} \cdot (1 - \mu). \quad (5)$$

Again, due to practical limitations, we compute the lower and upper bounds of μ up to β faults and the lower and upper bounds of $\epsilon_{kc}^{\tilde{\mathcal{F}}}$ up to α wires, if fault combinations can be corrected. Then, the lower and upper bounds of ϵ_{kf} are derived from the ones of $\epsilon_{kc}^{\tilde{\mathcal{F}}}$. Note, however, that those bounds on $\epsilon_{kc}^{\tilde{\mathcal{F}}}$ are related to the upper bound of μ . The reason is, that we cannot compute the respective $\epsilon_{kc}^{\tilde{\mathcal{F}}}$ for $\tilde{\mathcal{F}} \in \mathcal{B}_{\#i}$ with $i > \beta$, since this would mean to iterate over all faulty circuits with more than β faults. Then, the lower and upper bounds of γ_{kc} are computed from the corresponding bounds of both ϵ_{kf} and μ . The respective bounds of those three parameters are then returned.

Compositional Notions. We provide algorithms in Appendix B to describe how to perform the security verification of the compositional notions from Section 4, which have been implemented VERICA^+ and IronMask^+ .

5.2 Extension of VERICA Verification

We briefly present an optimization to count leaking wire combinations that fail simulations in the (general) random probing model in VERICA^+ with binomial trees, to provide a tighter lower bound of the simulation failure probability. Moreover, we outline additional considerations to assess the fault probability for the general random fault and known-fault random combined security. More details are given in Appendix C.

A Tighter Lower Bound in the (General) Random Probing Model. VERICA⁺ implements a novel approach, relying on the traversal of binomial trees, to identify larger wire combinations containing a failing wire combination detected by the tool, in a deterministic way. It also does not include redundant combinations or use extra memory. In short, VERICA⁺ finds failing wire combinations in a depth-first manner for up to α wires, $\alpha \in [1, |\mathcal{W}|]$, contrary to the naive breadth-first approach of VERICA. It results in the computation of a tighter lower bound of the simulation failure probability in the (general) random probing model, than previously mentioned in Section 5.1 and in [10], while concurrently providing an upper bound.

The core insight is that adding wires to a failing wire combination will result in another simulation failure. Thus, by representing the intermediate wires of a circuit as nodes of a binomial tree, VERICA⁺ identifies the larger wire combinations containing the failing one. Succinctly, VERICA⁺ traverses a binomial tree whose nodes represent the intermediate wires of a circuit, in pre-order fashion, from the root to a chosen depth α . During the traversal, VERICA⁺ updates the set of leaking wires to verify. Depending on the outcome of the verification, VERICA⁺ updates the computation of the lower and upper bounds of the simulation failure probability, in the considered security model. The latter values are returned at the end of the binomial tree traversal.

Probability of a Fault Combination. VERICA⁺ makes two assumptions to compute the probability of a non-corrected fault combination $\tilde{\mathcal{F}}$ for the evaluation of the correction failure probability μ . Firstly, VERICA⁺ allows a different number of fault transformations τ per gate g , which are equally likely to occur (*assumption 1*). Secondly, it assumes that up to one fault transformation τ is injected per gate and per verification analysis (*assumption 2*). As a result, to compute the probability of a fault combination, VERICA⁺ just iterates over the gates in the circuit, distinguishing between faulted and non-faulted ones.

5.3 Extension of IronMask Verification

IronMask verifies the implementation of (N)LR-gadgets. In essence, the gadgets accept up to two inputs, each potentially undergoing linear refreshing, which are then processed by a non-linear function (homogeneous multi-linear form) whose output may also be linearly refreshed. As argued in [11], a wide majority of masking gadgets existing in the literature are (N)LR. Our extension not only supports these gadgets but also accommodates more complex ones, termed C(N)LR-gadgets (for Correction blocks). These C(N)LR-gadgets are constructed by parallel replication of (N)LR-gadgets and incorporate intermediate non-linear correction blocks (e.g., implementing the majority function), with gates that may be affected by *set* or *reset* faults.

To handle C(N)LR-gadgets, we introduce a novel verification methodology that preserves the tool’s performance, at the expense of sometimes allowing false positives. This trade-off is necessary to benefit from the efficient tool’s verification paradigm, while allowing more complex constructions. This methodology

operates under the assumption that each output of a correction block remains correct as long as no fault occurs along its corresponding path within the block. Initially, the symbolic expression of each wire that depends on at least one output copy of a correction block, unaffected by a fault within the block, is modified to its correct symbolic expression (i.e., a hypothetical input of the correction block without faults along its path). Then, in the IronMask^+ verification process, output or internal variables from correction blocks impacted by faults are substituted with all the $2k + 1$ corresponding inputs of their correction block. While IronMask provides complete verification without false positives, our new methodology may produce artificial attack paths by conservatively replacing some variables with all their dependencies. This approach enhances the attacker’s potential reach by granting access to sets of variables instead of their functions.

IronMask^+ includes a Python script to generate correctable faulty scenarios for a circuit. It takes a circuit description, a fault type (*set* or *reset*), and the numbers of faults and tolerated faults (usually linked to the number of replications, $2k + 1$). The script evaluates circuit outputs in SageMath using symbolic computation with randoms and input shares. If any output share has more than k faulty copies, it is deemed uncorrectable. Faulty input shares become new symbolic variables, while faulty randoms are compared to a circuit with identical random faults (per [40]). Further details are provided in Appendix D

6 Evaluation

We select five different gadgets for experiments in the new known-fault random combined model (see Definitions 4 and 8) all designed for security against one fault and one probe. First, we consider the DOMREP [31] gadget, which is a simple replication of a masked multiplication gadget according to the Domain-Oriented Masking (DOM) [30] principle. To get a reasonable analysis, we add a majority vote at the gadget’s output to account for a correction that is required at some point in the circuit. Since DOMREP-based implementations were shown to be vulnerable to combined attacks (which the original design did not claim to protect against, cf. Section 3), a new variant, DOMREP-II [37], was recently introduced. This version uses two independent bits of randomness for mask refreshing and a masked correction module. To ensure a more accurate analysis, we once again add the masked correction at the gadget’s output. As a third design, we analyze the SININA gadget originally proposed for software in [19] and modified for hardware in [40]. This gadget is similarly based on a replicated DOM multiplication, however, with corrections introduced after the mask refreshing and before the compression. Fourth, we analyze HPC_1^C gadget from [27], which is a replicated DOM gadget with an additional refresh and correction of one of the shared inputs. Finally, the CPC_1 gadget from [25] is a variant of the HPC_1^C gadget with additional corrections after the mask refreshing in the multiplication. Note that the DOMREP, SININA, and HPC_1^C gadgets were shown to be insecure under composition [25,40,44,46]. Our case studies aim to highlight the strengths and limitations of VERICA^+ and IronMask^+ , and give a qualitative

comparison between the gadgets. We intentionally do not conduct an extensive security analysis of the gadgets under realistic leakage and fault distributions, as determining these distributions is left for future work. Nevertheless, such an analysis would only affect the absolute failure probabilities, without altering the qualitative expressiveness of the data.

Both tables presenting our results follow the same structure. The first column identifies the evaluated gadget, while the second and third columns show the number of gates (FIA locations) and the number of wires (SCA locations), respectively. The type of fault transformation is denoted by τ , while p represents the random probing leakage probability, and q denotes the random fault probability for each gate. In this setting, faults and leakages on different wires are independent, and the probabilities are equal across wires, except when only a subset of SCA and FIA locations is selected, where some probabilities are set to zero for efficiency reasons. The remaining columns display our results: $\mu_{\min}$ and $\mu_{\max}$ denote the bounds on the attacker’s advantage in the random fault model or for RFC, while $\epsilon_{\min}$ and $\epsilon_{\max}$ correspond to the bounds in the random probing model or for RPCUF. For the computation, we selected thresholds of $\alpha = 2$ for the number of probes and $\beta = 2$ for the number of faults. Lastly, $\gamma_{\min}$ and $\gamma_{\max}$ indicate the overall attacker advantage in the RCS_{KF} or RCC_{KF} models. Each table specifies whether the bounds are computed for general security or for composition. In general, a lower probability reflects a more secure implementation, and when the lower and upper bounds are close, we have a reasonable approximation of the real probability and, hence, a meaningful analysis. Note that the bounds on ϵ correlate to the bounds on $\mu_{\max}$. We first present our results in terms of security parameters and verification timings for each tool individually (note, some more results are presented in Appendix E). Lastly, we run the same experiments on both tools to validate correctness and compare timings in specific scenarios.

6.1 Results on VERICA⁺

All our experiments for VERICA⁺ have been conducted on a machine equipped with two AMD EPYC 7742 64-Core processors and 512 GB memory. This allowed us to perform all experiments on 256 threads. The analysis results and performance of VERICA⁺ for RCC_{KF} of the five gadgets under different scenarios can be found in Table 1. We first observe that, under the same fault and leakage conditions, the CPC_1 gadget is more secure than the other four gadgets. This is expected since CPC_1 is shown composable in the threshold combined model (CINI [27]), while three of the others have shown to be flawed under composition in the same model [25,40,44,46] and the fifth (DOMREP-II) seems to provide no further advantage. The discrepancy between the CPC_1 and the HPC_1^C gadget is mainly caused by the probing factor, while the discrepancy to the other gadgets is caused entirely by the faulting factor. The reason is, that for those gadgets, there exist some input-fault combinations that cannot be corrected properly (even without internal faults), specifically, when there are k faults in all the inputs. Those input-fault combinations are always considered insecure and, hence,

Table 1. RCC_{KF} analysis of gadgets on VERICA⁺.

Design	#Loc.		Model			Probabilities			Time
	FIA	SCA	τ	q	p	$\mu_{\min}/\mu_{\max}$	$\epsilon_{\min}/\epsilon_{\max}$	$\gamma_{\min}/\gamma_{\max}$	t
DOMREP [31]	67	114	<i>set</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	2.1 min
			<i>set</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	2.1 min
			<i>reset</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	2.1 min
			<i>reset</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	2.1 min
			<i>flip</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	2.0 min
			<i>flip</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	2.0 min
DOMREP-II [37]	298	444	<i>set</i>	2^{-10}	2^{-10}	9.96e-1/1.00e-0	0.00e-0/0.00e-0	9.96e-1/1.00e-0	22.8 h
			<i>set</i>	2^{-15}	2^{-15}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	22.6 h
			<i>reset</i>	2^{-10}	2^{-10}	9.96e-1/1.00e-0	0.00e-0/0.00e-0	9.96e-1/1.00e-0	22.6 h
			<i>reset</i>	2^{-15}	2^{-15}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	22.8 h
			<i>flip</i>	2^{-10}	2^{-10}	9.96e-1/1.00e-0	0.00e-0/0.00e-0	9.96e-1/1.00e-0	24.2 h
			<i>flip</i>	2^{-15}	2^{-15}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	24.4 h
SININA [40,19]	122	198	<i>set</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	46.0 min
			<i>set</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	46.1 min
			<i>reset</i>	2^{-10}	2^{-10}	9.99e-1/9.99e-1	2.99e-4/4.52e-2	9.99e-1/9.99e-1	48.2 min
			<i>reset</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	2.44e-4/1.34e-3	1.00e-0/1.00e-0	48.1 min
			<i>flip</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	45.0 min
			<i>flip</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	44.9 min
HPC _r ^C [27]	104	180	<i>set</i>	2^{-10}	2^{-10}	5.68e-2/5.69e-2	1.61e-1/1.61e-1	2.09e-1/2.09e-1	46.6 min
			<i>set</i>	2^{-15}	2^{-15}	1.83e-3/1.83e-3	5.48e-3/5.48e-3	7.30e-3/7.30e-3	46.4 min
			<i>reset</i>	2^{-10}	2^{-10}	6.41e-2/6.43e-2	1.61e-1/1.61e-1	2.15e-1/2.15e-1	40.5 min
			<i>reset</i>	2^{-15}	2^{-15}	2.07e-3/2.07e-3	5.48e-3/5.48e-3	7.54e-3/7.54e-3	40.6 min
			<i>flip</i>	2^{-10}	2^{-10}	6.41e-2/6.42e-2	1.61e-1/1.61e-1	2.15e-1/2.15e-1	39.1 min
			<i>flip</i>	2^{-15}	2^{-15}	2.07e-3/2.07e-3	5.48e-3/5.48e-3	7.54e-3/7.54e-3	39.0 min
CPC ₁ [25]	98	174	<i>set</i>	2^{-10}	2^{-10}	5.13e-2/5.14e-2	8.19e-2/8.21e-2	1.29e-1/1.29e-1	53.3 min
			<i>set</i>	2^{-15}	2^{-15}	1.65e-3/1.65e-3	2.65e-3/2.65e-3	4.29e-3/4.29e-3	52.6 min
			<i>reset</i>	2^{-10}	2^{-10}	6.78e-2/6.79e-2	8.17e-2/8.19e-2	1.44e-1/1.44e-1	40.9 min
			<i>reset</i>	2^{-15}	2^{-15}	2.19e-3/2.19e-3	2.65e-3/2.65e-3	4.84e-3/4.84e-3	40.8 min
			<i>flip</i>	2^{-10}	2^{-10}	6.96e-2/6.97e-2	8.18e-2/8.20e-2	1.46e-1/1.46e-1	38.8 min
			<i>flip</i>	2^{-15}	2^{-15}	2.26e-3/2.26e-3	2.65e-3/2.65e-3	4.90e-3/4.90e-3	38.5 min

dominate the failure probability for RFC because we determine the maximum over the input faults. This also explains why some of the ϵ bounds are zero. Since those bounds are related to $\mu_{\max}$, which is one, there is no side-channel evaluation performed. In this case, any internal faults can at most make the gadget more secure (by compensating the impact of input faults) and hence, the respective μ gets larger with smaller fault probability q .

To get a more in depth analysis, we analysed the DOMREP and DOMREP-II gadgets with (masked) correction for RCS_{KF} i.e., without composition. The DOMREP-II gadget is intended to be more secure as it prevents a known flaw in DOMREP. However, we find that the opposite is the case. Specifically, for $\tau = \text{flip}$, $q = 2^{-10}$, and $p = 2^{-10}$, we get $\gamma = [1.88e-3, 2.07e-3]$ for DOMREP and $\gamma = [1.79e-2, 3.05e-2]$ for DOMREP-II. Other fault scenarios provide similar results. This additional vulnerability of DOMREP-II can be explained through the application of the masked correction. Specifically, the use of DOM gadgets in the correction means that by faulting a share of the input a a fault at the gadget output is conditioned on the input b (similar to the attack explained

Table 2. RCS_{KF} analysis of the CPC_1 gadget [25] on VERICA^+ , with computation threshold for probes $\alpha = 2$ and for faults $\beta = 2$, where only a subset of gates and wires are faulted and potentially leaked to the adversary.

Location	#Loc.		Model			Probabilities			Time
	FIA	SCA	τ	q	p	$\mu_{\min}/\mu_{\max}$	$\epsilon_{\min}/\epsilon_{\max}$	$\gamma_{\min}/\gamma_{\max}$	t
all; all	98	174	<i>reset</i>	2^{-10}	2^{-10}	7.40e-4/8.72e-4	2.44e-3/3.00e-3	3.18e-3/3.87e-3	13.6 s
reg; rep. index 0	18	106	<i>reset</i>	2^{-10}	2^{-10}	5.07e-5/5.15e-5	8.89e-4/1.03e-3	9.39e-4/1.08e-3	0.8 s
reg; cone of c_0^0	18	103	<i>reset</i>	2^{-10}	2^{-10}	5.07e-5/5.15e-5	7.50e-4/8.76e-4	8.00e-4/9.27e-4	0.8 s

in Section 3 - last paragraph). The effectiveness of the respective fault at the output can be learned by the adversary through two probes, which are available with a certain probability. Hence, while DOMREP-II indeed prevents the specific attack as intended it introduced new vulnerabilities on the way. The practical exploitation of this leakage remains an avenue for future research.

With respect to performance, we can see that VERICA^+ can estimate the failure probabilities, by enumerating all fault and probe combinations with up to two faults and probes, for four of the gadgets in less than one hour. This is only possible because of the massive parallelization exploited by VERICA^+ . For the fifth gadget (DOMREP-II) the analysis requires nearly an entire day due to much larger designs because of the masked correction. As can be seen in Table 2 the evaluation of RCS_{KF} is much faster. Specifically, the CPC_1 gadget can be analyzed within seconds. This speedup is possible because RCC_{KF} needs to perform essentially the same analysis as for RCS_{KF} , however, for all input-fault and output-probe combinations.

In addition, we showcase the ability of VERICA^+ to analyze a design under different fault and leakage probabilities (where each location is faulted or leaks with independent probabilities). Specifically, we analyze the CPC_1 gadget [25] where we restrict the fault locations to registers only and the location of leaking wires to either all wires of the first replication (*rep. index 0*) or all wires that influence the output c_0^0 (*cone of c_0^0*). This could resemble an adversary that uses clock glitching for fault injection and an EM probe for side-channel analysis. Then, the two cases can be seen as representations of two possible layouts of the chip design, such that either all values of one replication index or all the values that influence the same output value are placed in close proximity. To give some context, we also provide in Table 2 the results of an analysis where all gates are faulted and all wires can leak. As can be seen, this allows a tighter analysis for specific attack scenarios, potentially enabling a reduction of the implementation cost if the attacker can be restricted in the possible fault and leakage distributions.

6.2 Results on IronMask⁺

All experiments for IronMask⁺ have been conducted on a single thread of a machine equipped with an AMD Ryzen Threadripper PRO 7995WX processor with

Table 3. RCC_{KF} analysis of gadgets on IronMask^+ .

Design	#Loc.		Model			Probabilities			Time	
	FIA	SCA	τ	q	p	$\mu_{\min}/\mu_{\max}$	$\epsilon_{\min}/\epsilon_{\max}$	$\gamma_{\min}/\gamma_{\max}$	t	
DOMREP [31]	127	203	<i>set</i>	2^{-10}	2^{-10}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	16min	11s
			<i>set</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	16min	10s
			<i>reset</i>	2^{-10}	2^{-10}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	16min	24s
			<i>reset</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	16min	24s
SININA [40,19]	230	364	<i>set</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	3h 12min	53s
			<i>set</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	3h 33min	52s
			<i>reset</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	1.18e-4/1.09e-1	9.99e-1/1.00e-0	3h 21min	59s
			<i>reset</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	1.22e-4/3.30e-3	1.00e-0/1.00e-0	3h 13min	49s
HPC ₁ ^C [27]	188	322	<i>set</i>	2^{-10}	2^{-10}	3.68e-2/3.76e-2	9.93e-1/9.97e-1	9.93e-1/9.97e-1	23h 54min	38s
			<i>set</i>	2^{-15}	2^{-15}	1.16e-3/1.16e-3	1.00e-0/1.00e-0	1.00e-0/1.00e-0	23h 55min	18s
			<i>reset</i>	2^{-10}	2^{-10}	8.53e-2/8.60e-2	9.95e-1/1.00e-0	9.96e-1/1.00e-0	23h 32min	20s
			<i>reset</i>	2^{-15}	2^{-15}	2.68e-3/2.68e-3	1.00e-0/1.00e-0	1.00e-1/1.00e-0	23h 33min	10s
CPC ₁ [25]	182	316	<i>set</i>	2^{-10}	2^{-10}	8.89e-2/8.95e-2	1.62e-1/1.66e-1	2.37e-1/2.41e-1	21h 42min	38s
			<i>set</i>	2^{-15}	2^{-15}	2.93e-3/2.93e-3	5.66e-3/5.66e-3	8.57e-3/8.57e-3	21h 41min	48s
			<i>reset</i>	2^{-10}	2^{-10}	8.89e-2/8.96e-2	1.62e-1/1.65e-1	2.36e-1/2.40e-1	20h 33min	24s
			<i>reset</i>	2^{-15}	2^{-15}	2.93e-3/2.93e-3	5.63e-3/5.63e-3	8.54e-3/8.54e-3	20h 33min	15s

96 cores (192 threads) and 512GB of RAM. The analysis outcomes and performance results of IronMask^+ for RCC_{KF} across various scenarios are presented in Table 3. Unlike VERICA^+ , IronMask^+ cannot evaluate DOMREP-II, as it does not conform to the C(N)LR structure described in Section 5.3. Specifically, the gadget employs a masked correction mechanism using DOM gadgets, which deviates from the majority-vote structure required by IronMask^+ .

In this extension, we exclusively focus on *set* and *reset* faults, deferring the inclusion of other fault types for future work. It is notable that IronMask^+ performs quite efficiently (especially given its execution on a single thread), thanks to its evaluation of symbolic variables with efficient rules. At the same time, this allows for some tolerance of false positives, as detailed in Section 5. Specifically, although the bounds on the final γ advantages are pretty close to VERICA^+ for the CPC gadget, IronMask^+ outputs higher bounds on ϵ for HPC_1^C . This is because with probed output and faulty input shares, we quickly multiply the failures when considering all the copies. Furthermore, note that the increase in the lower bound of ϵ as p decreases (e.g., for HPC_1^C), is solely a consequence of the method used which restricts the execution to computing failures of size at most two, whereas the correct ϵ would actually increase with p .

Regarding performance, the four computations involving different probabilities under identical settings (gadget, fault type) can be partially generated concurrently. For example, when considering the CPC₁ gadget [25] for *set* faults, the selection of faulty circuits that can be corrected takes around 9 minutes. Subsequently, identifying failure tuples within these correctable circuits requires around 20 hours, 43 minutes. Both of these operations are common to any combination of faults and probe probabilities. Finally, computing security advantages based on the probabilities of faults and probes requires 50 to 52 minutes.

6.3 Tool Comparison

We provide an extensive comparison between our two tool implementations in Appendix F. In general, both tools yield the same bounds for μ , with only minor differences in the computed ϵ , which is expected due to variations in computational tightness. This reinforces our confidence in the correctness of both implementations. In general, **VERICA**⁺ achieves tighter computations of failure probabilities, while **IronMask**⁺ leverages design-specific structures for efficient computations, leading to higher performance. Thus, the tools offer different trade-offs between accuracy and efficiency.

7 Conclusion

In this work, we extended and refined probabilistic models for physical attacks with general leaking and fault probabilities. Specifically, we examined the random probing model, the random fault model, and their combinations. For the resulting models, we explored key properties, such as the impact of adversarial knowledge and composition. Additionally, we developed two tool-based analysis methods that enable the security assessment of design components. Ultimately, we hope our model will stimulate further research into the low-level physical effects of leakage and their expressions in probability distributions.

Acknowledgments. This work was co-funded by the European Union (ERC, AMaskZONE, 101077506). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them. In addition, the work described in this paper has been supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany’s Excellence Strategy - EXC 2092 CASA - 390781972 and through the project CAVE (510964147), and by the German Federal Ministry of Education and Research BMBF through the project 6GEM (16KISK038), DI-SIGN-HEP (16KIS2073), and DI-OWAS (16ME0967).

References

1. Aghaie, A., Moradi, A., Rasoolzadeh, S., Shahmirzadi, A.R., Schellenberg, F., Schneider, T.: Impeccable circuits. *IEEE Trans. Computers* **69**(3), 361–376 (2020)
2. Ajtai, M.: Secure computation with information leaking to an adversary. In: Fortnow, L., Vadhan, S.P. (eds.) 43rd ACM STOC. pp. 715–724. ACM Press (Jun 2011). <https://doi.org/10.1145/1993636.1993731>
3. Amiel, F., Villegas, K., Feix, B., Marcel, L.: Passive and active combined attacks: Combining fault attacks and side channel analysis. In: FDTC 2007: Vienna, Austria. pp. 92–102 (2007)
4. Ananth, P., Ishai, Y., Sahai, A.: Private circuits: A modular approach. In: Shacham, H., Boldyreva, A. (eds.) CRYPTO 2018, Part III. LNCS, vol. 10993, pp. 427–455. Springer, Cham (Aug 2018). https://doi.org/10.1007/978-3-319-96878-0_15

5. Arribas, V., Wegener, F., Moradi, A., Nikova, S.: Cryptographic Fault Diagnosis using VerFI. In: HOST 2020. pp. 229–240. IEEE (2020)
6. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y.: Verified proofs of higher-order masking. In: Oswald, E., Fischlin, M. (eds.) EUROCRYPT 2015, Part I. LNCS, vol. 9056, pp. 457–485. Springer, Berlin, Heidelberg (Apr 2015). https://doi.org/10.1007/978-3-662-46800-5_18
7. Barthe, G., Belaïd, S., Dupressoir, F., Fouque, P.A., Grégoire, B., Strub, P.Y., Zucchini, R.: Strong non-interference and type-directed higher-order masking. In: Weippl, E.R., Katzenbeisser, S., Kruegel, C., Myers, A.C., Halevi, S. (eds.) ACM CCS 2016. pp. 116–129. ACM Press (Oct 2016). <https://doi.org/10.1145/2976749.2978427>
8. Battistello, A., Coron, J.S., Prouff, E., Zeitoun, R.: Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In: Gierlichs, B., Poschmann, A.Y. (eds.) CHES 2016. LNCS, vol. 9813, pp. 23–39. Springer, Berlin, Heidelberg (Aug 2016). https://doi.org/10.1007/978-3-662-53140-2_2
9. Belaïd, S., Cassiers, G., Mutschler, C., Rivain, M., Roche, T., Standaert, F., Taleb, A.R.: Towards achieving provable side-channel security in practice. IACR Cryptol. ePrint Arch. p. 1198 (2023)
10. Belaïd, S., Coron, J.S., Prouff, E., Rivain, M., Taleb, A.R.: Random probing security: Verification, composition, expansion and new constructions. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020, Part I. LNCS, vol. 12170, pp. 339–368. Springer, Cham (Aug 2020). https://doi.org/10.1007/978-3-030-56784-2_12
11. Belaïd, S., Mercadier, D., Rivain, M., Taleb, A.R.: IronMask: Versatile verification of masking security. In: 2022 IEEE Symposium on Security and Privacy. pp. 142–160. IEEE Computer Society Press (May 2022). <https://doi.org/10.1109/SP46214.2022.9833600>
12. Berndt, S., Eisenbarth, T., Faust, S., Gourjon, M., Orlt, M., Seker, O.: Combined fault and leakage resilience: Composability, constructions and compiler. In: Handschuh, H., Lysyanskaya, A. (eds.) CRYPTO 2023, Santa Barbara, CA, USA. LNCS, vol. 14083, pp. 377–409. Springer (2023)
13. Cassiers, G., Standaert, F.: Trivially and Efficiently Composing Masked Gadgets With Probe Isolating Non-Interference. IEEE Trans. Inf. Forensics Secur. **15**, 2542–2555 (2020)
14. Chari, S., Jutla, C.S., Rao, J.R., Rohatgi, P.: Towards Sound Approaches to Counteract Power-Analysis Attacks. In: Wiener, M.J. (ed.) CRYPTO 1999. LNCS, vol. 1666, pp. 398–412. Springer (1999)
15. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M.: Passive and Active Combined Attacks on AES: Combining Fault Attacks and Side Channel Analysis. In: FDTC 2010, Santa Barbara, California, USA. pp. 10–19 (2010)
16. Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms, Second Edition. The MIT Press and McGraw-Hill Book Company (2001)
17. De Cnudde, T., Bilgin, B., Gierlichs, B., Nikov, V., Nikova, S., Rijmen, V.: Does coupling affect the security of masked implementations? In: Guilley, S. (ed.) COSADE 2017. LNCS, vol. 10348, pp. 1–18. Springer, Cham (Apr 2017). https://doi.org/10.1007/978-3-319-64647-3_1
18. Dehbaoui, A., Dutertre, J., Robisson, B., Tria, A.: Electromagnetic Transient Faults Injection on a Hardware and a Software Implementations of AES. In: FDTC 2012. pp. 7–15. IEEE Computer Society (2012)
19. Dhooghe, S., Nikova, S.: My gadget just cares for me - how NINA can prove security against combined attacks. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS,

- vol. 12006, pp. 35–55. Springer, Cham (Feb 2020). https://doi.org/10.1007/978-3-030-40186-3_3
20. Dhooghe, S., Nikova, S.: The random fault model. In: Carlet, C., Mandal, K., Rijmen, V. (eds.) SAC 2023, Fredericton, Canada. LNCS, vol. 14201, pp. 191–212. Springer (2023)
 21. Dobraunig, C., Eichlseder, M., Groß, H., Mangard, S., Mendel, F., Primas, R.: Statistical ineffective fault attacks on masked AES with fault countermeasures. In: Peyrin, T., Galbraith, S. (eds.) ASIACRYPT 2018, Part II. LNCS, vol. 11273, pp. 315–342. Springer, Cham (Dec 2018). https://doi.org/10.1007/978-3-030-03329-3_11
 22. Duc, A., Dziembowski, S., Faust, S.: Unifying leakage models: From probing attacks to noisy leakage. In: Nguyen, P.Q., Oswald, E. (eds.) EUROCRYPT 2014. LNCS, vol. 8441, pp. 423–440. Springer, Berlin, Heidelberg (May 2014). https://doi.org/10.1007/978-3-642-55220-5_24
 23. Dumont, M., Lisart, M., Maurine, P.: Electromagnetic Fault Injection : How Faults Occur. In: FDTC 2019. pp. 9–16. IEEE (2019)
 24. Faust, S., Grosso, V., Pozo, S.M.D., Paglialonga, C., Standaert, F.: Composable Masking Schemes in the Presence of Physical Defaults & the Robust Probing Model. IACR TCHES **2018**(3), 89–120 (2018)
 25. Feldtkeller, J., Güneysu, T., Moos, T., Richter-Brockmann, J., Saha, S., Sasdrich, P., Standaert, F.: Combined private circuits - combined security refurbished. In: Meng, W., Jensen, C.D., Cremers, C., Kirda, E. (eds.) ACM CCS 2023, Copenhagen, Denmark. pp. 990–1004. ACM (2023)
 26. Feldtkeller, J., Güneysu, T., Schaumont, P.: Quantitative fault injection analysis. In: Guo, J., Steinfeld, R. (eds.) ASIACRYPT 2023, Guangzhou, China. LNCS, vol. 14441, pp. 302–336. Springer (2023)
 27. Feldtkeller, J., Richter-Brockmann, J., Sasdrich, P., Güneysu, T.: CINI MINIS: Domain isolation for fault and combined security. In: Yin, H., Stavrou, A., Cremers, C., Shi, E. (eds.) ACM CCS 2022. pp. 1023–1036. ACM Press (Nov 2022). <https://doi.org/10.1145/3548606.3560614>
 28. Gandolfi, K., Moutrel, C., Olivier, F.: Electromagnetic analysis: Concrete results. In: Koç, Çetin Kaya., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 251–261. Springer, Berlin, Heidelberg (May 2001). https://doi.org/10.1007/3-540-44709-1_21
 29. Goubin, L., Patarin, J.: DES and differential power analysis (the “duplication” method). In: Koç, Çetin Kaya., Paar, C. (eds.) CHES’99. LNCS, vol. 1717, pp. 158–172. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48059-5_15
 30. Groß, H., Mangard, S., Korak, T.: Domain-Oriented Masking: Compact Masked Hardware Implementations with Arbitrary Protection Order. In: ACM TIS@CCS 2016. p. 3. ACM (2016)
 31. Gruber, M., Probst, M., Karl, P., Schamberger, T., Tebelmann, L., Tempelmeier, M., Sigl, G.: DOMREP-An Orthogonal Countermeasure for Arbitrary Order Side-Channel and Fault Attack Protection. IEEE Trans. Inf. Forensics Secur. **16**, 4321–4335 (2021)
 32. Ishai, Y., Prabhakaran, M., Sahai, A., Wagner, D.: Private circuits II: Keeping secrets in tamperable circuits. In: Vaudenay, S. (ed.) EUROCRYPT 2006. LNCS, vol. 4004, pp. 308–327. Springer, Berlin, Heidelberg (May / Jun 2006). https://doi.org/10.1007/11761679_19

33. Ishai, Y., Sahai, A., Wagner, D.: Private circuits: Securing hardware against probing attacks. In: Boneh, D. (ed.) CRYPTO 2003. LNCS, vol. 2729, pp. 463–481. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/978-3-540-45146-4_27
34. Kocher, P.C.: Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In: Koblitz, N. (ed.) CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer, Berlin, Heidelberg (Aug 1996). https://doi.org/10.1007/3-540-68697-5_9
35. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) CRYPTO'99. LNCS, vol. 1666, pp. 388–397. Springer, Berlin, Heidelberg (Aug 1999). https://doi.org/10.1007/3-540-48405-1_25
36. Mangard, S., Popp, T., Gammel, B.M.: Side-channel leakage of masked CMOS gates. In: Menezes, A. (ed.) CT-RSA 2005, San Francisco, CA, USA. LNCS, vol. 3376, pp. 351–365. Springer (2005)
37. Probst, M., Brosch, M., Gruber, M., Sigl, G.: DOMREP II. In: IEEE HOST 2024, Tysons Corner, VA, USA. pp. 112–121. IEEE (2024)
38. Prouff, E., Rivain, M.: Masking against side-channel attacks: A formal security proof. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 142–159. Springer, Berlin, Heidelberg (May 2013). https://doi.org/10.1007/978-3-642-38348-9_9
39. Renauld, M., Standaert, F., Veyrat-Charvillon, N., Kamel, D., Flandre, D.: A formal study of power variability issues and side-channel attacks for nanoscale devices. In: Paterson, K.G. (ed.) EUROCRYPT 2011, Tallinn, Estonia. LNCS, vol. 6632, pp. 109–128. Springer (2011)
40. Richter-Brockmann, J., Feldtkeller, J., Sasdrich, P., Güneysu, T.: VERICA - verification of combined attacks automated formal verification of security against simultaneous information leakage and tampering. IACR TCHES **2022**(4), 255–284 (2022). <https://doi.org/10.46586/tches.v2022.i4.255-284>
41. Richter-Brockmann, J., Rezaei Shahmirzadi, A., Sasdrich, P., Moradi, A., Güneysu, T.: FIVER – Robust Verification of Countermeasures against Fault Injections. IACR TCES **2021**(4), 447–473 (2021)
42. Richter-Brockmann, J., Sasdrich, P., Güneysu, T.: Revisiting Fault Adversary Models - Hardware Faults in Theory and Practice. IEEE Trans. Computers pp. 1 – 14 (2022)
43. Roche, T., Lomné, V., Khalfallah, K.: Combined Fault and Side-Channel Attack on Protected Implementations of AES. In: CARDIS 2011, Leuven, Belgium. pp. 65–83 (2011)
44. Saha, S., Bag, A., Jap, D., Mukhopadhyay, D., Bhasin, S.: Divided we stand, united we fall: Security analysis of some SCA+SIFA countermeasures against SCA-enhanced fault template attacks. In: Tibouchi, M., Wang, H. (eds.) ASIACRYPT 2021, Part II. LNCS, vol. 13091, pp. 62–94. Springer, Cham (Dec 2021). https://doi.org/10.1007/978-3-030-92075-3_3
45. Saha, S., Jap, D., Breier, J., Bhasin, S., Mukhopadhyay, D., Dasgupta, P.: Breaking Redundancy-Based Countermeasures with Random Faults and Power Side Channel. In: FDTC 2018, Amsterdam, The Netherlands. pp. 15–22 (2018)
46. Saha, S., Ravi, P., Jap, D., Bhasin, S.: Non-Profiled Side-Channel Assisted Fault Attack: A Case Study on DOMREP. In: DATE 2023. pp. 1–6. IEEE, Antwerp, Belgium (2023)
47. Schellenberg, F., Gnad, D.R.E., Moradi, A., Tahoori, M.B.: Remote inter-chip power analysis side-channel attacks at board-level. In: Bahar, I. (ed.) ICCAD 2018, San Diego, CA, USA. p. 114. ACM (2018)

48. Shahmirzadi, A.R., Rasoolzadeh, S., Moradi, A.: Impeccable Circuits II. In: DAC 2020. pp. 1–6. IEEE (2020)
49. Skorobogatov, S.P., Anderson, R.J.: Optical fault induction attacks. In: Kaliski Jr., B.S., Koç, Çetin Kaya., Paar, C. (eds.) CHES 2002. LNCS, vol. 2523, pp. 2–12. Springer, Berlin, Heidelberg (Aug 2003). https://doi.org/10.1007/3-540-36400-5_2
50. Yao, Y., Yang, M., Patrick, C., Yuce, B., Schaumont, P.: Fault-assisted side-channel analysis of masked implementations. In: 2018 IEEE International Symposium on Hardware Oriented Security and Trust, HOST 2018, Washington, DC, USA, April 30 - May 4, 2018. pp. 57–64. IEEE Computer Society (2018). <https://doi.org/10.1109/HST.2018.8383891>, <https://doi.org/10.1109/HST.2018.8383891>
51. Zussa, L., Dutertre, J., Clédière, J., Tria, A.: Power supply glitch induced faults on FPGA: An in-depth analysis of the injection mechanism. In: IOLTS 2013. pp. 110–115. IEEE (2013)

Appendix A Composition in the Random Probing Model

For the general random probing model, we can build upon the notion of RPC proposed by Belaïd et al. [10] and extend it for general distributions. At a high level, RPC provides $\mathcal{A}_p$ with the usual random probing leakage for the gadget and additionally with a tuple $\mathcal{O}$ of arbitrary but bounded sets of probes on shares of each output. More specifically, for each unshared output with index i , there is a bounded set $\mathcal{O}_i$ that contains the share indices of the probed output shares. Then, a gadget is composable if the view of $\mathcal{A}_p$ can be simulated with high probability using only bounded sets of input shares, i.e., for each unshared input i the simulator requires only a bounded set of share indices $\mathcal{I}_i$. Here, the set of input and output shares are bounded by the same parameter d , i.e., $|\mathcal{O}_i| \leq d$ and $|\mathcal{I}_i| \leq d$. The extension to arbitrary probability distributions is straightforward: the internal wire selection uses $\mathbb{DW}^\infty$ instead of a wire-independent leakage probability.

Definition 9 (GRPC). A gadget $G : (\mathbb{F}_2^s)^h \rightarrow (\mathbb{F}_2^s)^m$ is $(d, \mathbb{DW}^\infty, \epsilon)$ -GRPC if there exists a deterministic algorithm **ShareSelect** and a probabilistic simulator **Sim** such that for every input x and for every tuple of sets $\mathcal{O} = (\mathcal{O}_i)_{i \in [m]}$, with $\forall i : \mathcal{O}_i \subseteq [s]$ and $|\mathcal{O}_i| \leq d$, there exists a $\epsilon \leq 1$ such that the random experiment

$$\begin{aligned}
 \tilde{\mathcal{W}} &\leftarrow \text{LeakingWires}(G, \mathbb{DW}^\infty) \\
 \mathcal{I} &:= (\mathcal{I}_1 \dots \mathcal{I}_h) \leftarrow \text{ShareSelect}(\tilde{\mathcal{W}}, \mathcal{O}) \\
 \text{out} &\leftarrow \text{Sim}(x|_{\mathcal{I}}, \mathcal{I}, \mathcal{O}, \tilde{\mathcal{W}})
 \end{aligned}$$

yields

$$\begin{aligned}
 \Pr[(|\mathcal{I}_1| > d) \vee \dots \vee (|\mathcal{I}_h| > d)] &\leq \epsilon \\
 \text{out} &\equiv (\text{AssignWires}(G, \tilde{\mathcal{W}}, x), y|_{\mathcal{O}}),
 \end{aligned}$$

with $y \leftarrow G(x)$.

Arguing about the composition of gadgets requires knowledge of the joint probability distribution for the leakage in multiple gadgets. Using a complex (and for some scenarios more realistic) joint probability distribution, where the leakage of different gadgets is mutually dependent, contradicts the core idea of composition, i.e., the independent analysis of individual gadgets. Hence, in the following, we assume mutually independent leakage distributions for each gadget. While this is certainly a restriction in the generality of our notion, we think it is justified by the significant gains of the gadget-based approach.

Gadgets supporting GRPC can be composed in an arbitrary way as long as each output of a gadget is used only once as input to another gadget. However, the failure probability increases linearly with the number of used gadgets. Our theorem is actually a generalization of Theorem 1 from Belaïd et al. [10], and the proof is similar except for the computation of the global failure probability. Instead of being $1 - (1 - \epsilon)^{|C|}$, bounded by $|C| \cdot \epsilon$, the latter naturally becomes $1 - \prod_{i=1}^{|C|} (1 - \epsilon_i)$ to reflect more tightly the individual gadget-simulation failures.

Theorem 4. *Let C be a circuit constructed by composition of $(d, \mathbb{D}\mathcal{W}_i^\infty, \epsilon_i)$ -GRPC gadgets G_i , for $i \in \{1, \dots, |C|\}$, such that each output of G_i is used as input of at most one other gadget G_j or as output of C . Given this, C is $(\prod_{i=1}^{|C|} \mathbb{D}\mathcal{W}_i^\infty, 1 - \prod_{i=1}^{|C|} (1 - \epsilon_i))$ -random probing secure.*

Relation to (Random) Probing-Secure Composition. As mentioned, our notion is a direct extension of RPC [10] to arbitrary probe distributions. Another concept for the construction of random probing secure circuits is *expansion* [4,10]. For this, a circuit compiler, which replaces each gate with a respective secure gadget, is applied e -times to a circuit. Hence, for $e = 1$, we get the normal gadget-based approach. However, for $e = 2$, each share in the previous masked circuit is shared again and each gate of a gadget is replaced with the respective gadget of the gate again. In essence, this reduces the leakage probability of wires as now the simulation of entire gadgets needs to fail in order to leak a single share (of the originally masked circuit). Hence, the leakage probability p is replaced by the simulation-failure probability ϵ . Using the right gadgets can result in an exponential (in e) security increase, but also an exponential increase in implementation costs. We leave the generalization to arbitrary probe distributions for future work.

In the standard ISW probing model [33], where an adversary can choose up to d wires to be probed, the requirements for compositions are now well understood. In particular, it requires to restrict and guide the flow of probe propagation through the circuit. While (S)NI [6,7] restricts the number of input shares that can be used for the simulation of probes, PINI [13] restricts the location (in terms of share domains) of input shares the simulator can use. Hence, (G)RPC has some similarity with (S)NI, in that it restricts the number of input shares a simulator can access. However, in contrast to (S)NI, the number of shares the simulator can access is a fixed parameter that is independent of the amount of leakage within the gadget.

Appendix B Algorithms for Compositional Notions

The computation of the simulation failure probability for the composition of gadgets in the general random probing model is described in Algorithm 1 (cf. Definition 9). Similarly, Algorithm 2 specifies how to generate the correction failure probability for the composition in the general random fault model based on Definition 6. Finally, Algorithm 3 details the computation of the correction and the simulation failure probabilities for the composition in the combined random model as well as the advantage of the combined adversary (cf. Definition 8).

Algorithm 1: General Random Probing Composability Verification

```

1: Inputs: gadget  $G$  with the associated set of all wire combinations  $\mathcal{W}^\infty$  and wire
   probabilities  $p_{w,w \in \mathcal{W}}$ , masking order  $d$ 
2: Output: simulation failure probability  $\epsilon$ 
3:  $E \leftarrow []$  {** List of simulation failure probabilities }
4: {** Loop over all possible  $\mathcal{O}$  **}
5: for each  $\mathcal{O} = (\mathcal{O}_1, \dots, \mathcal{O}_m)$ , with  $\forall i : \mathcal{O}_i \subseteq [s]$  and  $|\mathcal{O}_i| \leq d$  do
6:    $\epsilon \leftarrow 0$ 
7:   for each  $\tilde{\mathcal{W}}$  in  $\mathcal{W}^\infty$  do
8:      $\mathcal{I} = (\mathcal{I}_1, \dots, \mathcal{I}_h) \leftarrow \text{ShareSelect}(\tilde{\mathcal{W}}, \mathcal{O})$  {** Get input shares }
9:     if there exists  $j \in [1, h]$  such that  $|\mathcal{I}_j| > d$  then
10:        $\epsilon \leftarrow \epsilon + \prod_{w \in \tilde{\mathcal{W}}} p_w \cdot \prod_{w' \in \mathcal{W} \setminus \tilde{\mathcal{W}}} (1 - p_{w'})$  {** Equation 2 }
11:     end if
12:   end for
13:   Append  $\epsilon$  to  $E$ 
14: end for
15: return maximum( $E$ )

```

Algorithm 2: General Random Fault Composability Verification

```

1: Inputs: gadget  $G$  with the associated set of fault combinations  $\mathcal{F}^\infty$  and fault
   probabilities  $q_{f,f \in \mathcal{F}}$ , replication order  $k$ 
2: Output: correction failure probability  $\mu$ 
3:  $M \leftarrow []$  {** List of correction failure probabilities }
4: {** Loop over all possible  $\mathcal{I}'$  **}
5: for each  $\mathcal{I}' = (\mathcal{I}'_1, \dots, \mathcal{I}'_h)$  with  $\forall i : \mathcal{I}'_i \subseteq [n]$  and  $|\mathcal{I}'_i| \leq k$  do
6:    $\mu \leftarrow 0$ 
7:   for each  $\tilde{\mathcal{F}}$  in  $\mathcal{F}^\infty$  do
8:      $\mathcal{O}' = (\mathcal{O}'_1, \dots, \mathcal{O}'_m) \leftarrow \text{ReplicationSelect}(G^{\tilde{\mathcal{F}}}, \mathcal{I}')$  {** Get output faulty
       replications for the faulted gadget }
9:     if there exists  $j \in [1, m]$  such that  $|\mathcal{O}'_j| > k$  then
10:       $\mu \leftarrow \mu + \prod_{f \in \tilde{\mathcal{F}}} q_f \cdot \prod_{f' \in \mathcal{F} \setminus \tilde{\mathcal{F}}} (1 - q_{f'})$  {** Equation 3 }
11:     end if
12:   end for
13:   Append  $\mu$  to  $M$ 
14: end for
15: return maximum( $M$ )

```

Algorithm 3: Combined Random Composability Verification

```

1: Inputs: gadget  $G$  with  $\mathcal{F}^\infty$  and fault probabilities  $q_{f,f \in \mathcal{F}}$ , and with  $\mathcal{W}^\infty$  and
   wire probabilities  $p_{w,w \in \mathcal{W}}$ , masking order  $d$ , replication order  $k$ 
2: Outputs: probabilities  $\mu$ ,  $\epsilon_{kf}$  and  $\gamma_{kc}$ 
3:  $G \leftarrow []$  {** List to store advantage combined adversary }
4: {** Loop over all possible  $\mathcal{I}'$  **}
5: for each  $\mathcal{I}' = (\mathcal{I}'_1, \dots, \mathcal{I}'_h)$  with  $\forall i : \mathcal{I}'_i \subseteq [n]$  and  $|\mathcal{I}'_i| \leq k$  do
6:    $\mu \leftarrow 0$ ,  $\epsilon_{kf} \leftarrow 0$ 
7:   for each  $\tilde{\mathcal{F}}$  in  $\mathcal{F}^\infty$  do
8:      $\mathcal{O}' = (\mathcal{O}'_1, \dots, \mathcal{O}'_m) \leftarrow \text{ReplicationSelect}(G^{\tilde{\mathcal{F}}}, \mathcal{I}')$  {** Get output faulty
       replications for the faulted gadget }
9:     if there exists  $j \in [1, m]$  such that  $|\mathcal{O}'_j| > k$  then
10:       $\mu \leftarrow \mu + \prod_{f \in \tilde{\mathcal{F}}} q_f \cdot \prod_{f' \in \mathcal{F} \setminus \tilde{\mathcal{F}}} (1 - q_{f'})$  {** Equation 3 }
11:     else
12:      Return  $\epsilon_{kc}^{\tilde{\mathcal{F}}}$  from Algorithm 1 with inputs  $\mathcal{W}^\infty$ ,  $p_{w,w \in \mathcal{W}}$  and  $d$ 
13:       $\epsilon_{kf} \leftarrow \epsilon_{kf} + \epsilon_{kc}^{\tilde{\mathcal{F}}} \cdot \prod_{f \in \tilde{\mathcal{F}}} q_f \cdot \prod_{f' \in \mathcal{F} \setminus \tilde{\mathcal{F}}} (1 - q_{f'})$  {** Equation 4 }
14:     end if
15:   end for
16:    $\gamma_{kc} \leftarrow \mu + \epsilon_{kf} \cdot (1 - \mu)$  {** Equation 5 }
17:   Append  $(\gamma_{kc}, \mu, \epsilon_{kf})$  to  $G$ 
18: end for
19: return maximum value  $\gamma_{kc}$  in  $G$  and its corresponding  $\mu$  and  $\epsilon_{kf}$ 

```

Appendix C Extension of VERICA Verification

C.1 Implementation of the (General) Random Probing Model

As mentioned in Section 5.2, VERICA^+ implements a novel approach, using a binomial tree traversal, to compute the lower and upper bounds of the simulation failure probability ϵ in the general random probing and random probing model. It results in a tighter lower bound in both random probing models than the method mentioned in Section 5.1, which is an extension of the one from [10].

The main insight is that adding wires to a leaking wire combination that fails simulation $\tilde{\mathcal{W}}_{\text{fail}}$ will lead to another simulation failure. The idea is thus to derive the set of larger wire combinations containing this leaking wire combination that is found to fail simulation by VERICA^+ . In addition, we want to build this set without including redundant combinations or using additional memory⁷, in a deterministic way.

More precisely, VERICA^+ considers only the intermediate wires of a given circuit C , and labels them in topological order, i.e., the wire w_1 is close to an input of C and $w_{|\mathcal{W}|}$ to an output. However, finding the sets $\mathcal{W}_{\#i}^\infty$ from Equation 2 is infeasible for a large number of wires i . Contrary to the naive breadth-first approach that is implemented in the original version of VERICA [40], the idea now is to find failing leaking wire combinations $\tilde{\mathcal{W}}_{\text{fail}}$ in a depth-first manner for up to α wires, $\alpha \in [1, |\mathcal{W}|]$, and deriving the larger wire combinations containing them (up to $|\mathcal{W}|$ wires), by incrementally walking through a binomial tree $B_{|\mathcal{W}|}$.

Definition 10 (Binomial tree [16]). Let $l \in \mathbb{N}$ be an index, $l > 0$. A binomial tree is defined recursively as an ordered tree such that a binomial tree of order 0 is a single node, and a binomial tree B_l of order l has a root node whose children are root nodes of binomial (sub)trees of descending orders from $(l - 1)$ to 1.

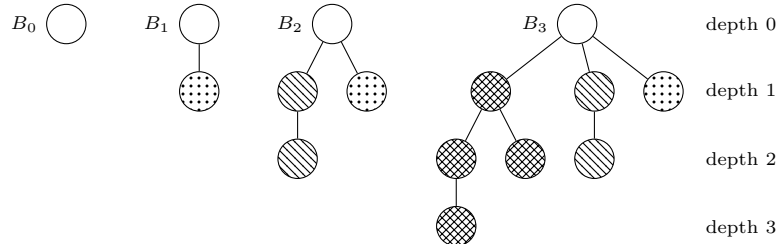


Fig. 2. Recursive definition of binomial trees, from B_0 to B_3 .

Definition 10 is illustrated in Figure 2. For VERICA^+ , we define the binomial tree $B_{|\mathcal{W}|}$ of order $|\mathcal{W}|$ that has a height $|\mathcal{W}|$ and $2^{|\mathcal{W}|}$ nodes. By definition, the number of nodes at depth l equals the binomial coefficient $\binom{|\mathcal{W}|}{l}$. The nodes of

⁷ It is not desirable to store all the found wire combinations that lead to a simulation failure, due to the computational blow-up.

$B_{|\mathcal{W}|}$ are labeled by the collection of indices of the wires in $\mathcal{W}$, i.e., each label represents a unique wire $w \in \mathcal{W}$, except the root node. At depth l , the nodes sharing a parent are labeled in ascending order from l to $|\mathcal{W}|$, from leftmost to rightmost node.

VERICA⁺ traverses $B_{|\mathcal{W}|}$ in pre-order fashion⁸, from the root to a chosen depth α , which is the threshold number of wires to verify (cf. Section 5.1). During the traversal, if VERICA⁺ visits the child of a node, it adds the wire labeled by the said child to a set of leaking wires, which represents the currently considered wire combination $\tilde{\mathcal{W}}$. VERICA⁺ then runs the verification on the latter. If it fails, VERICA⁺ updates the computation of the lower and upper bounds of the simulation failure probability, in the considered security model, as detailed in the following. Directly after, VERICA⁺ goes back to the parent node by removing the wire indexed by the current node from the set of leaking wires; the children nodes are not visited even if the depth α has not been reached. However, if the simulation does not fail, and the depth α is not attained, the tool continues the procedure on the children nodes. The root of $B_{|\mathcal{W}|}$ represents the empty set $\emptyset$; thus, VERICA⁺ verifies l wires at depth l . At the end of the tree traversal, the lower and upper bounds of the simulation failure probability are returned.

In the rest of this section, we first explain the computation of the lower bound of the simulation failure probability, followed by the one of the upper bound. Then, we present a bound evaluation for a (1,1)-CPC₁ gadget in the random probing model. Finally, we provide the pseudo-code to conduct the binomial tree traversal and update the bounds.

Remark 1. The approach in this section is not applicable to injected faults, since adding faults to a non-correctable fault combination does not necessarily result in a failed correction. Thus, it is not used for the general random fault model.

A Tighter Lower Bound. If the currently considered wire combination fails simulation, VERICA⁺ proceeds with the following computation for the lower bound of the simulation failure probability for the random probing and general random probing security.

In the Random Probing Model. If a failing wire combination is found at depth $l_{\text{fail}} \in [1, \alpha]$ (i.e., $\tilde{\mathcal{W}}_{\text{fail}}$ contains l_{fail} wires), VERICA⁺ determines the depth of the subtree of the current node $l_{\text{sub}} \in [0, |\mathcal{W}| - l_{\text{fail}}]$. It then updates the computation of the lower bound by adding the following:

$$\begin{aligned}
 & p^{l_{\text{fail}}} \cdot (1-p)^{|\mathcal{W}| - l_{\text{fail}} - l_{\text{sub}}} \cdot \overbrace{\sum_{k=0}^{l_{\text{sub}}} \binom{l_{\text{sub}}}{k} p^k (1-p)^{l_{\text{sub}} - k}}^{\text{subtree cut (binomial CDF)=1}} \quad (6) \\
 \Leftrightarrow & p^{l_{\text{fail}}} \cdot (1-p)^{|\mathcal{W}| - l_{\text{fail}} - l_{\text{sub}}}.
 \end{aligned}$$

⁸ In pre-order traversal, the root of the tree is first visited, then the leftmost subtree is recursively traversed, and afterward, the right subtree up to the rightmost one.

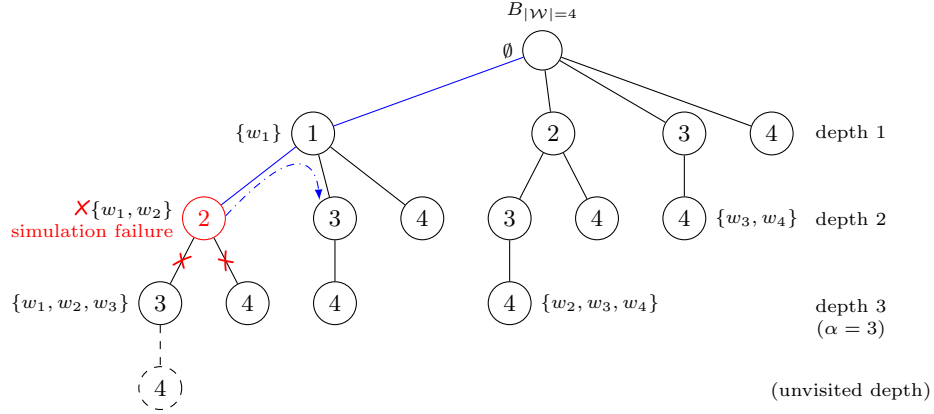


Fig. 3. Example of a binomial tree $B_{|W|=4}$ to find the exact failing leaking combinations up to $\alpha = 3$ wires (depth 3). The dashed node will not be visited in any case. The blue edges denote the path walked by VERICA^+ . At the node representing w_2 at depth 2 (in red), the leaking set is $\{w_1, w_2\}$ and fails simulation. Thus, VERICA^+ does not visit the children nodes and updates the computation of $\epsilon_{\min}$. It continues by verifying the set $\{w_1, w_3\}$ (blue dashed arrow).

and continues the walk back to the parent node. VERICA^+ thus ‘cuts’ the subtree of the current node to include it entirely in the computation of the lower bound. This is explicitly reflected in Equation 6 with the binomial Cumulative Distribution Function (CDF) for a sample of size l_{sub} (all combinations of wires in the subtree are considered) and probability p . We evaluate for l_{sub} , thus it equals one.

In the General Random Probing Model. The ‘cut’ of the subtree of the current node, when there is a found $\tilde{\mathcal{W}}_{\text{fail}}$, is similar to the one in the random probing model. However, since each wire has a different probability to leak, we adapt the update of the computation of $\epsilon_{\min}$ with:

$$\begin{aligned}
 & \prod_{w \in \tilde{\mathcal{W}}_{\text{fail}}} p_w \cdot \prod_{\substack{w' \in \mathcal{W} \\ w' \notin \tilde{\mathcal{W}}_{\text{fail}} \\ w' \notin \mathcal{W}_{\text{sub}}}} (1 - p_{w'}) \cdot \overbrace{\sum_{k=0}^{|\mathcal{W}_{\text{sub}}|} \sum_{\substack{\tilde{\mathcal{W}} \in \mathcal{W}_{\#k}^\infty \\ \tilde{\mathcal{W}} \subseteq \mathcal{W}_{\text{sub}}}} \prod_{w \in \tilde{\mathcal{W}}} p_w \prod_{\substack{w' \in \mathcal{W}_{\text{sub}} \\ w' \notin \tilde{\mathcal{W}}}} (1 - p_{w'})}^{\text{subtree cut (Poisson binomial CDF)=1}} \quad (7) \\
 \Leftrightarrow & \prod_{w \in \tilde{\mathcal{W}}_{\text{fail}}} p_w \cdot \prod_{\substack{w' \in \mathcal{W} \\ w' \notin \tilde{\mathcal{W}}_{\text{fail}} \\ w' \notin \mathcal{W}_{\text{sub}}}} (1 - p_{w'}),
 \end{aligned}$$

where $\mathcal{W}_{\text{sub}}$ is the set of wires in the children nodes (subtree). It thus ‘cuts’ the subtree of the current node to include it entirely in the computation of $\epsilon_{\min}$ with the Poisson binomial CDF for a sample of size $|\mathcal{W}_{\text{sub}}|$. Indeed, with the double

summation, we construct all wire combinations from the wires in $\mathcal{W}_{\text{sub}}$ that fail the simulation along with the wires in $\tilde{\mathcal{W}}_{\text{fail}}$. And since the Poisson binomial CDF is evaluated for $|\mathcal{W}_{\text{sub}}|$, it equals one. Therefore, Equation 7 is simplified to the product of wire probabilities in the failing wire combination $\tilde{\mathcal{W}}_{\text{fail}}$ times the product of probabilities of wires that are neither in the failing combination nor in the subtree of the current node.

Example. Figure 3 depicts an example of the binomial tree traversal for $\alpha = 3$ and $|\mathcal{W}| = 4$. For the random probing security, Equation 6 would be calculated with $l_{\text{fail}} = l_{\text{sub}} = 2$ and $|\mathcal{W}| = 4$, which is equal to p^2 . In the general random probing model, VERICA^+ updates ϵ_{min} with $p_{w_1}p_{w_2}$ for the first product of Equation 7, times 1 for the second product (all wires are already included in the failing wire combination or the children nodes).

If no subtree can be cut for optimization ($l_{\text{sub}} = 0$ in Equation 6 or $\mathcal{W}_{\text{sub}} = \emptyset$ in Equation 7), the verification is equivalent to iteratively testing all combinations of $1, \dots, \alpha$ wires. This is only the case if $w_{|\mathcal{W}|}$ leaks or there is no leakage. In general, the more the failing wires or wire combinations are located to the rightmost part of the tree, the less tight the lower bound is or the closer this approach is to the one presented in Section 5.1. That is because the binomial tree is not symmetric. However, if $\alpha = |\mathcal{W}|$, we ensure to find the exact value of the simulation failure probability with this method, since failing wire combination are picked in a deterministic way without redundancy.

An Upper Bound. VERICA^+ computes an upper bound for the simulation failure probability, by considering that if combinations of α wires did not lead to a simulation failure, then combinations of $\alpha + 1$ wires will (cf. Section 5.1).

In short, VERICA^+ performs the binomial tree traversal and the computation of the lower bound of the simulation failure as previously explained. However, in addition, if it visits a node at depth α and the verification set does not lead to a simulation failure, it considers that the remaining nodes in the un-visited subtree of depth $l_{\text{sub}_\alpha} \in [0, |\mathcal{W}| - \alpha]$ will lead to a simulation failure with the parent nodes. The exact computation depends on the security model.

In the Random Probing Model. If $l_{\text{sub}_\alpha} > 0$, VERICA^+ additionally updates the computation of the upper bound of the simulation failure probability with:

$$p^\alpha \cdot \sum_{k=1}^{l_{\text{sub}_\alpha}} \binom{l_{\text{sub}_\alpha}}{k} p^k (1-p)^{|\mathcal{W}|-\alpha-k} \quad (8)$$

$$\Leftrightarrow p^\alpha \cdot (1-p)^{|\mathcal{W}|-\alpha-l_{\text{sub}_\alpha}} \cdot \left(1 - (1-p)^{l_{\text{sub}_\alpha}}\right).$$

Note that the sum from the first equation starts at $k = 1$ since we consider that the combinations of the α wires from the root to the current node do not fail simulation, but their combinations with any wire represented in the children nodes will.

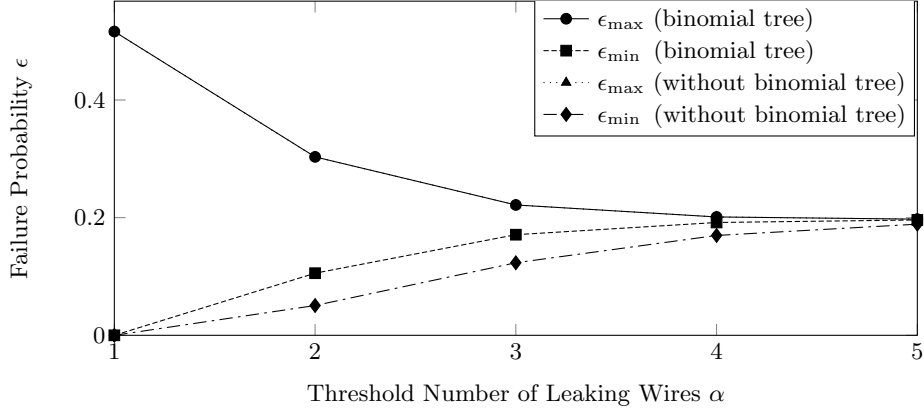


Fig. 4. Comparison of the computation of the lower and upper bounds of the simulation failure probability ϵ in the random probing model ($p = 0.010$), computed with and without the optimization based on the binomial trees, for a $(1,1)$ -CPC₁ gadget.

In the General Random Probing Model. Similarly, VERICA⁺ adds to the computation of $\epsilon_{\max}$:

$$\begin{aligned}
 & \prod_{w \in \tilde{\mathcal{W}}_\alpha} p_w \cdot \prod_{\substack{w' \in \mathcal{W} \\ w' \notin \tilde{\mathcal{W}}_\alpha \\ w' \notin \mathcal{W}_{\text{sub}_\alpha}}} (1 - p_{w'}) \cdot \sum_{k=1}^{l_{\text{sub}_\alpha}} \sum_{\substack{\tilde{\mathcal{W}} \in \mathcal{W}_{\#k}^\infty \\ \tilde{\mathcal{W}} \subseteq \mathcal{W}_{\text{sub}_\alpha}}} \prod_{w \in \tilde{\mathcal{W}}} p_w \prod_{\substack{w' \in \mathcal{W}_{\text{sub}_\alpha} \\ w' \notin \tilde{\mathcal{W}}}} (1 - p_{w'}) \quad (9) \\
 & \Leftrightarrow \prod_{w \in \tilde{\mathcal{W}}_\alpha} p_w \prod_{\substack{w' \in \mathcal{W} \\ w' \notin \tilde{\mathcal{W}}_\alpha \\ w' \notin \mathcal{W}_{\text{sub}_\alpha}}} (1 - p_{w'}) \cdot \left(1 - \prod_{w'' \in \mathcal{W}_{\text{sub}_\alpha}} (1 - p_{w''}) \right),
 \end{aligned}$$

where $\tilde{\mathcal{W}}_\alpha$ are the α wires from the root to the current node that do not lead to a simulation failure, and $\mathcal{W}_{\text{sub}_\alpha}$ is the set of the wires that are represented by the children nodes of the current node at depth α . Note that the first expression is similar to Equation 7, but the outer sum starts at $k = 1$ for the same reason stated in the random probing model. Therefore, with the double summation, we construct all wire combinations of size $k > 1$ from the wires in $\mathcal{W}_{\text{sub}_\alpha}$ that are assumed to fail the simulation. Its probability simplifies to one minus the probability to not take any wire in $\mathcal{W}_{\text{sub}_\alpha}$.

Evaluation of Tighter Bound Computation. We present a case study where we compare the computation of the lower and upper bounds of the simulation failure probability for the random probing security with and without the optimization based on binomial trees, for the $(1,1)$ -CPC₁ gadget, in Figure 4. Each wire is assumed to leak with the same leaking wire probability $p = 0.010$, and no fault is injected. Note that we choose a higher leakage probability to get more

pronounced differences between the upper and lower bounds. The bounds of the simulation failure probability are evaluated for combinations of $\alpha = 1, \dots, 5$ wires for a more in-depth analysis.

We observe that the two methods provide similar probabilities for the upper bound $\epsilon_{\max}$ (appearing as a unique solid line). However, the binomial tree approach gives a higher value for the lower bound $\epsilon_{\min}$ than without. Hence, we see that the optimization provides a tighter lower bound and converges faster to the actual value of the simulation failure probability ϵ .

Binomial Tree Traversal Algorithm. A pseudo-code of the binomial tree traversal with the calculation of the bounds is provided in Algorithm 4. Practically, VERICA⁺ utilizes the fact that the nodes of the binomial tree $B_{|\mathcal{W}|}$ represent the intermediate wires $\mathcal{W}$ of a given circuit C labeled in topological order, to perform the tree traversal through a recursive function.

Algorithm 4: Binomial Tree Preorder Traversal to Compute Simulation Failure Probability

```

1: Inputs: circuit  $C$  with the set of wires  $\mathcal{W}$  and corresponding wire
   probabilities  $p_{w,w \in \mathcal{W}}$ , threshold number of wires  $\alpha$ 
2: Output: bounds of simulation failure probability  $\epsilon_{\min}, \epsilon_{\max}$ 
3:  $\epsilon_{\min} \leftarrow 0, \epsilon_{\max} \leftarrow 0$  {** Lower and upper bounds}
4:  $\tilde{\mathcal{W}} \leftarrow \{\}$  {** Leaking wire combination to verify}
5: {** Define recursive function **}
6: procedure PREORDER( $\epsilon_{\min}, \epsilon_{\max}, \tilde{\mathcal{W}}, k$ )
7:   for  $i = k$  to  $|\mathcal{W}|$  do
8:     if  $|\tilde{\mathcal{W}}| = \alpha$  then
9:        $\epsilon_{\max} \leftarrow \epsilon_{\max} + \text{update\_upp\_bound}(\mathcal{W}, \alpha, \tilde{\mathcal{W}}, p_{w,w \in \mathcal{W}})$ 
       {** Equation 8, or Equation 9 with  $\tilde{\mathcal{W}} = \tilde{\mathcal{W}}_\alpha$ }
10:      break
11:    else
12:      Append  $w_i$  to  $\tilde{\mathcal{W}}$ 
13:      if VERICA+ returns a simulation failure for  $\tilde{\mathcal{W}}$  then
14:         $\epsilon_{\min} \leftarrow \epsilon_{\min} + \text{update\_low\_bound}(\mathcal{W}, \alpha, \tilde{\mathcal{W}}, p_{w,w \in \mathcal{W}})$ 
15:         $\epsilon_{\max} \leftarrow \epsilon_{\max} + \text{update\_low\_bound}(\mathcal{W}, \alpha, \tilde{\mathcal{W}}, p_{w,w \in \mathcal{W}})$ 
        {** Equation 6, or Equation 7 with  $\tilde{\mathcal{W}} = \tilde{\mathcal{W}}_{\text{fail}}$ }
16:      else
17:         $\epsilon_{\min}, \epsilon_{\max} \leftarrow \text{PREORDER}(\epsilon_{\min}, \epsilon_{\max}, \tilde{\mathcal{W}}, i + 1)$ 
18:      end if
19:      Remove  $w_i$  from  $\tilde{\mathcal{W}}$ 
20:    end if
21:  end for
22:  return  $\epsilon_{\min}, \epsilon_{\max}$ 
23: end procedure
24: {** Call recursive function **}
25:  $\epsilon_{\min}, \epsilon_{\max} \leftarrow \text{PREORDER}(\epsilon_{\min}, \epsilon_{\max}, \tilde{\mathcal{W}}, 1)$ 

```

C.2 Probability of a Fault Combination.

In this section, as outlined in Section 5.2, we explain how VERICA^+ exactly computes the probability of a fault combination $\tilde{\mathcal{F}}$ which is required for the evaluation of the correction failure probability μ (cf. Section 5.1), if the said fault combination can not be corrected.

We recall that each fault f is a pair (g, τ) , where g is a gate and $\tau : \mathcal{G}_f \rightarrow \mathcal{G}_f$ is the transformation of the Boolean function encoded by the gate. As explained in Section 5.1, for practicality reasons, VERICA^+ considers that each fault f has a probability q_f and the probability of a fault combination is:

$$\Pr[\tilde{\mathcal{F}}] = \prod_{f \in \tilde{\mathcal{F}}} q_f \cdot \prod_{f' \in \mathcal{F} \setminus \tilde{\mathcal{F}}} q_{f'}. \quad (10)$$

For each fault, we can further write:

$$q_f = \Pr[\tau \cap g] = \Pr[\tau | g] \Pr[g], \quad (11)$$

where $\Pr[g]$ is the probability that a gate $g \in \mathcal{G}$ is targeted and then a specific fault transformation τ occurs with probability $\Pr[\tau | g]$.

Furthermore, VERICA^+ allows a different number of fault transformations per gate. Let T_g be the set of all possible fault transformations of a specific gate g , which are considered to be equally likely to occur in VERICA^+ (Assumption 1). Hence, the probability of a particular fault transformation τ applied to the gate g is:

$$\Pr[\tau | g] = |T_g|^{-1},$$

which we substitute into Equation 11 to have $q_f = \Pr[g] \cdot |T_g|^{-1}$ for VERICA^+ . Moreover, the latter also assumes that up to one fault transformation τ is injected at a gate location, per verification analysis (Assumption 2). As a result, we can rewrite Equation 10 for VERICA^+ by

$$\Pr[\tilde{\mathcal{F}}] = \prod_{(g, \tau) \in \tilde{\mathcal{F}}} \frac{\Pr[g]}{|T_g|} \cdot \prod_{(g', \tau') \in \mathcal{F} \setminus \tilde{\mathcal{F}}} (1 - \Pr[g']), \quad (12)$$

where we consider the product of probabilities to inject a fault transformation into some gates times the product of probabilities to not inject any fault into the remaining gates. Therefore, to compute the probability of a fault combination, VERICA^+ simply iterates over the gates in the circuit and differentiates the faulted gates from the non-faulted ones.

Appendix D Extension of IronMask Verification

In this section, we commence by revisiting the core principles of **IronMask**. Next, we expound on the extension of **IronMask**'s random probing security verification for C(N)LR gadgets, defined to depict faulty circuits. Following this, we introduce our novel Python script designed to evaluate the random fault security.

Finally, we comment on the limitation of our tool regarding general models and its application to the verification of combined models and compositional notions. Henceforth, we simply denote the extended version of `IronMask` as `IronMask+`, encompassing the evolutions outlined in this section.

Main Principles of `IronMask`. In [11], the authors introduce an exact verification procedure of the random probing security of masked gadgets. Their procedure is implemented in a tool referred to as `IronMask`.⁹ This verification tool supports any masking gadgets with linear randomness (i.e., LR-gadgets) and also the so-called NLR-gadgets that are quadratic gadgets which might include non-linear randomness (e.g., by refreshing their inputs). Specifically, an ℓ -to- m NLR-gadget is a gadget G with ℓ s -share inputs and m s -share outputs such that:

$$G : (\mathbb{K}^s)^\ell \rightarrow (\mathbb{K}^s)^m \\ (x_1, \dots, x_\ell) \mapsto (y_1, \dots, y_m) = R_{\ell+1}(F(R_1(x_1, r_1), \dots, R_\ell(x_\ell, r_\ell)), r_{\ell+1})$$

where F is any arithmetic circuit which computes a homogeneous multi-linear form, the R_i are linear arithmetic circuits and the r_i are vectors of random values uniformly drawn from a finite field $\mathbb{K}$.

The (N)LR-gadgets are sufficient for capturing most basic operations of masked implementations. However, to achieve combined security, additional intermediate correction blocks, such as majority votes, may be required. While the primary purpose of these correction blocks is to provide fault resistance, they often introduce highly non-linear behavior as a side effect.

Furthermore, when considering faults, they can alter the initial gadget structure. In this work, we focus solely on set or reset faults. Specifically, a gate g might be faulted to output either the constant one or the constant zero, i.e., $\tau_{\text{set}}(g) = \text{one}$, $\tau_{\text{reset}}(g) = \text{zero}$ (see Section 3.2). Nevertheless, these two types of faults do not affect the (N)LR format of the gadgets between the correction blocks.

Following the additional constraints of combined security resistance and the presence of faults, our objective is twofold:

- (i) Integrating the verification of more intricate gadgets designed to satisfy a property of combined composability. These gadgets consist of $n = 2k + 1$ replications of an original (N)LR-gadget with intermediate correction blocks, where all the gates may be impacted by set or reset faults. We will refer to these new gadgets as C(N)LR gadgets in the following (with the additional 'C' letter to signify the presence of intermediate correction blocks).
- (ii) Not only verifying the random probing composability of input gadgets but also considering faults to verify the random fault security, the known-fault random combined security and their compositional variants.

⁹ <https://github.com/CryptoExperts/IronMask>

Extended Verification of Random Probing Security. Below, we introduce the new $C(N)$ LR gadgets, which serve as inputs for our extension tool, *IronMask*⁺. Subsequently, we elaborate on their verification process.

New $C(N)$ LR-Gadgets. As previously outlined, *IronMask* takes the implementation of (N)LR-gadgets as input. Our extension continues to support these gadgets while also accommodating more intricate ones that we refer to as $C(N)$ LR-gadgets. These $C(N)$ LR-gadgets are constructed by parallel replication of (N)LR-gadgets, incorporating intermediate non-linear correction blocks (e.g., implementing the majority function) and whose gates may be modified by set and reset faults.

An example is provided in Figure 5 with the (1,1)-CINI gadget from [25], implementing a masked multiplication with two masked inputs $a = (a_0, a_1) \in \mathbb{F}_2^2$ and $b = (b_0, b_1) \in \mathbb{F}_2^2$, $k = 1$ (i.e., 3 copies) and two correction blocks referred to as C_0 and C_1 . Copies of the same variable x , that are defined for fault resistance, are denoted x^0 , x^1 , and x^2 . For the sake of clarity, we do not represent copy operations that are mandatory to replicate any variable that is to be used in more than one gate.

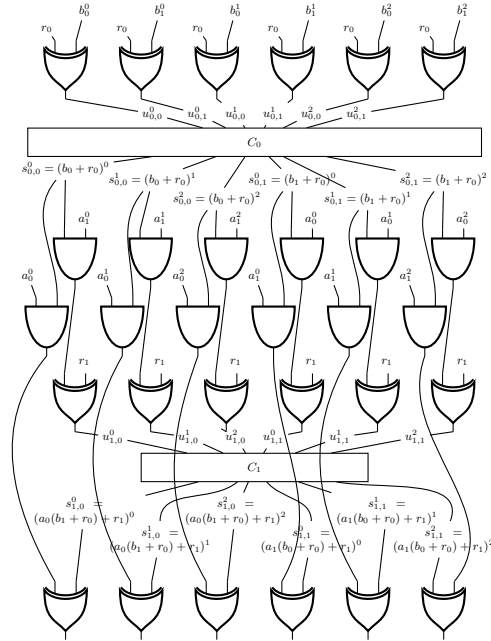


Fig. 5. CNLR gadget which implements a masked multiplication with 2-shared inputs, 3 replications and 2 intermediate correction blocks.

Verification of $C(N)LR$ -Gadgets. To date, no technique has been able to efficiently and accurately verify the (random) probing security of highly non-linear circuits. The computation of exact probe distributions quickly becomes prohibitively inefficient, while techniques relying on the analysis of symbolic expressions of variables prove incomplete in the presence of non-linear randomness (with the exception of specialized scenarios like the one developed in **IronMask**). To approach completeness for a $C(N)LR$ -gadget G , we propose a novel and efficient verification methodology in three steps. This methodology operates under the assumption that each output of a correction block remains correct provided that no fault occurs along its corresponding path within the correction block. Our method comes in three steps described below.

Step 1. Each wire in G is labeled with its symbolic expression made of the outputs of the correction blocks (starting from the last one), the random values, and the gadget inputs. Namely, we introduce symbolic variables $s_{i,j}^\ell$ to represent copies of the outputs of each correction block C_i , where i is the correction block index, j the share index and ℓ the replication index. Then, all the subsequent variables are directly expressed according to these $s_{i,j}^\ell$ that are not anymore further developed with their symbolic expression. For instance, the first output of the gadget represented in Figure 5 would be labeled $a_0^0 \cdot s_{0,0}^0 \oplus s_{1,0}^0$, with $s_{0,0}^0$ the first copy of the first output of the first correction block C_0 and $s_{1,0}^0$ the first copy of the first output of the second correction block C_1 .

Step 2. The gadget G is modified with the following step that is repeated until nothing can be further replaced (at most as many times as the number of correction blocks): the symbolic expression of each wire which depends on at least one output copy $s_{i,j}^\ell$ of a correction block, which is not impacted by a fault injected inside the correction block, is modified to replace $s_{i,j}^\ell$ by its correct symbolic expression (i.e., an hypothetical input of the correction block for which no fault had occurred in the path). In our example, if there is no fault on the path producing $s_{1,0}^0$ in C_1 , the symbolic expression of the first output of G would become $a_0^0 \cdot s_{0,0}^0 \oplus (a_0 \cdot s_{0,1} \oplus r_1)$.

Step 3. All the wires in the updated gadget G' are still labeled with symbolic expression made of the outputs of the correction blocks, the random values, and the gadget inputs. At this step, we can apply **IronMask**'s original verification on each tuple of variables with slight modifications.

Recall that **IronMask**'s original verification operates within two steps. The first step basically performs a Gaussian elimination on the probes that are linear combinations of sub-products and random values that do not appear in the sub-products. Given the structure of $(N)LR$ gadgets, all the random values that appear in these linear combinations are eliminated at this point. The second step then manipulates probes that are either the remaining sub-products or linear combinations of input shares and other random values. After a factorization step, described in [11], to linearize the probe expressions, a second Gaussian elimination is finally performed.

In `IronMask+`, to consider both the output correction blocks or the intermediate variable of correction blocks (that are impacted by faults, from the first steps), they need to be replaced by the corresponding inputs of their correction block during the two verification steps. Specifically, the Gaussian elimination is performed progressively for each probe in the considered tuple. Once the probe is treated, each output of correction block or intermediate variable in its expression is replaced by the corresponding $2k + 1$ inputs of its correction blocks. Similarly, each intermediate variables of a correction block is replaced by the corresponding $2k + 1$ inputs of its correction blocks. They are then treated like regular probes in the Gaussian elimination. The exact same scenario happens in the second step of `IronMask+`'s verification during the second Gaussian elimination procedure. A pseudo-code of `IronMask+` for the verification of a tuple in the random probing model is given in Algorithm 5, with blue highlights on the added replacements. The only differences with the original `IronMask`'s verification lies in the calls to the new function `replace_correction_outputs`, available in Algorithm 6. Functions `factorize` and `input_shares` are not detailed here since they are explained in [11] and are not useful in this extension's description.

Algorithm 5: IronMask⁺ Verification of a Tuple

```

1: Inputs: a tuple of probes  $\mathbf{w} = (w_1, \dots, w_k)$ 
2: Outputs: inputs shares required for the simulation input_shares
3:  $L = []$  {** List of symbolic expressions}
4:  $R = []$  {** List of random masks for each symbolic expression}
5: {** First Step **}
6: for  $i = 1$  to  $k$  do
7:   for  $j = 1$  to  $\text{length}(L)$  do
8:     if  $R[j]$  is in the symbolic expression of  $w_i$  then
9:        $w_i \leftarrow w_i + L[j]$ 
10:    end if
11:  end for
12:  Append  $w_i$  to  $L$ 
13:  if there is any remaining additive random  $r$  in the expression of  $w_i$  then
14:    Append  $r$  to  $R$ 
15:  else
16:    Append 0 to  $R$ 
17:  end if
18:  call replace_correction_outputs( $w_i$ ,  $L$ ,  $R$ )
19: end for
20: {** Second Step **}
21:  $L' = []$ 
22:  $R' = []$ 
23: for  $i = 1$  to  $\text{length}(L)$  do
24:   if  $R[i] = 0$  then
25:      $\text{fact} \leftarrow \text{factorize}(L[i])$  {** factorized expressions of  $L[i]$ }
26:     for each  $w$  in  $\text{fact}$  do
27:       for  $j = 1$  to  $\text{len}(L')$  do
28:         if  $R'[j]$  is in the symbolic expression of  $w$  then
29:            $w \leftarrow w + L'[j]$ 
30:         end if
31:       end for
32:       Append  $w$  to  $L'$ 
33:       if there is any remaining additive random  $r$  in the expression of  $w$  then
34:         Append  $r$  to  $R'$ 
35:       else
36:         Append 0 to  $R'$ 
37:       end if
38:       call replace_correction_outputs( $w$ ,  $L'$ ,  $R'$ )
39:     end for
40:   end if
41: end for
42:  $\text{input\_shares} \leftarrow \text{get\_input\_shares}(L', R')$  {** input shares from  $L'$ }

```

Algorithm 6: Function `replace_correction_outputs`

```

1: Inputs and Outputs: probe  $w$ , lists  $L$  and  $R$ 
2: if  $R[-1] \neq 0$  then
3:   return
4: end if
5: for each correction block output  $s_i$  in the expression of  $w$  do
6:    $w_i \leftarrow s_i$ 
7:   for  $j = 1$  to  $\text{len}(L)$  do
8:     if  $R[j]$  is in the expression of  $w_i$  then
9:        $w_i \leftarrow w_i + L[j]$ 
10:    end if
11:  end for
12:  Append  $w_i$  to  $L$ 
13:  if there is any remaining additive random  $r$  in the expression of  $w_i$  then
14:    Append  $r$  to  $R$ 
15:  else
16:    Append 0 to  $R$ 
17:  end if
18:  call replace_correction_outputs( $w_i$ ,  $L$ ,  $R$ )
19: end for

```

While `IronMask` provides complete verification without false positives, our new methodology may yield artificial attack paths due to the third item. The first two points of our methodology simply substitute symbolic expressions with simplified expressions of equivalent value to aid in computation of correction steps. However, the third item addresses non-linear terms not conforming to `IronMask`'s format due to specific faults. In such cases, intermediate variables are replaced by all their dependencies, which is a conservative approach. By granting attackers access to sets of variables rather than functions of these variables, our methodology enhances their potential reach. Put differently, if our extended tool `IronMask`⁺ determines a tuple as not being a failure, then indeed it is not. Conversely, if it indicates a failure path, there is a possibility of false positives in certain scenarios.

Remark 2. One could easily extend our verification procedure to gadgets with a different format (i.e., not covered by our (C)(N)LR gadgets), or whose authorized faults might impact the non-linearity, by proceeding as in the third point. Namely, as soon as the symbolic expressions of variables within a targeted tuple are not eligible to a complete verification, they can trivially be replaced by all their dependencies for the rest of the verification. Although not tight, this method benefits from capturing all the possible attack paths.

Verification of Random Fault Security. `IronMask`⁺ is completed with a Python script to generate all the faulty scenarios for a circuit that can be corrected. It takes as input a circuit description, a single type of faults (*set* or *reset*),

a number of faults and a number of tolerated faults (usually linked to the number of replications $2k+1$). The script then evaluates copies of the circuit outputs in SageMath using symbolic computation (Boolean polynomial ring) based on randoms and input shares (where the copy of the input share is considered as the input share, i.e., a_0^0 evaluates to a_0). Then, for a combination of faults, the circuit is re-evaluated in SageMath, and the outputs are compared: if, for any of the output shares, there are strictly more than k faulty copies, it constitutes a faulty scenario that cannot be corrected. For the types of faults, we have either *set* or *reset*, and if the fault is on a copy of an input share, it is treated as a new symbolic variable, i.e., a_0^0 with a fault becomes a new variable in the Boolean polynomial ring, instead of evaluating to a_0 when not faulty. However, for faults on the randoms, the outputs are not compared to the original circuit (golden circuit) but rather to the circuit with the same faults on the randoms (as justified for combined analysis in [40]).

Verification of General and Combined Models. We briefly discuss the possible extension of *IronMask*⁺ to general models and its application to verify combined security and compositional notions.

General Models. We should note that we do not verify the general random probing security in *IronMask*⁺ for various probabilities. While this is not particularly challenging to implement, it is significantly less efficient, as *IronMask*⁺ only computes the cardinals of the leaking tuples based on their size (formerly referred to as $|\mathcal{W}_{\#i}^\infty|$ in Equation 2). Similarly, our Python script to verify the random fault security currently applies for faults with the same probability for efficiency reasons. Adapting to the general random fault model would simply require to associate a specific probability to each faulty circuit.

Combined Security. The new verification of the random fault security with the extension to C(N)LR gadgets makes it possible to verify the known-fault random combined security directly from both advantages, as detailed in Section 5.1. The attacker is computed following Equation 5 from the result of Equation 4 with identical fault probabilities.

Compositional Notions. As in previous versions of *IronMask*, the compositional notions are all based on the verification of the previously detailed properties (random probing security, random fault security and known-fault random combined security). As depicted in Appendix B, the main difference lies in exploring all the input and output behaviors.

Appendix E Extended Evaluation Results

We provide additional evaluation results for *VERICA*⁺ in Table 4 and for *IronMask*⁺ in Table 5. Interestingly, for the $(1, 1)$ -CPC₁ gadget the bounds for μ are

not consistent when keeping q constant. This is because we determine the maximum $\gamma_{\min}$ and report the respective bounds for ϵ and μ . Since the ϵ changes with p the resulting maximum in $\gamma_{\min}$ can change as well, meaning that we report different bounds for μ .

Table 4. RCC_{KF} analysis of three gadgets with different fault and leakage probabilities on VERICA^+ . Here, all faults and wire leakages happen with the same indicated probability.

Design	#Loc.		Model			Probabilities			Time	
	FIA	SCA	τ	q	p	$\mu_{\min}/\mu_{\max}$	$\epsilon_{\min}/\epsilon_{\max}$	$\gamma_{\min}/\gamma_{\max}$	t	
$(1,1)\text{-CPC}_1$ [25]	174	98	<i>set</i>	2^{-10}	2^{-10}	5.13e-2/5.14e-2	8.19e-2/8.21e-2	1.29e-1/1.29e-1	53.3 min	
			<i>set</i>	2^{-10}	2^{-15}	5.31e-2/5.33e-2	2.57e-3/2.57e-3	5.56e-2/5.57e-2	53.3 min	
			<i>set</i>	2^{-15}	2^{-10}	1.52e-3/1.52e-3	8.43e-2/8.44e-2	8.57e-2/8.58e-2	52.6 min	
			<i>set</i>	2^{-15}	2^{-15}	1.65e-3/1.65e-3	2.65e-3/2.65e-3	4.29e-3/4.29e-3	52.6 min	
			<i>reset</i>	2^{-10}	2^{-10}	6.78e-2/6.79e-2	8.17e-2/8.19e-2	1.44e-1/1.44e-1	40.9 min	
			<i>reset</i>	2^{-10}	2^{-15}	6.78e-2/6.79e-2	2.66e-3/2.66e-3	7.03e-2/7.04e-2	41.3 min	
			<i>reset</i>	2^{-15}	2^{-10}	2.04e-3/2.04e-3	8.42e-2/8.44e-2	8.61e-2/8.63e-2	40.9 min	
			<i>reset</i>	2^{-15}	2^{-15}	2.19e-3/2.19e-3	2.65e-3/2.65e-3	4.84e-3/4.84e-3	40.8 min	
			<i>flip</i>	2^{-10}	2^{-10}	6.96e-2/6.97e-2	8.18e-2/8.20e-2	1.46e-1/1.46e-1	38.8 min	
			<i>flip</i>	2^{-10}	2^{-15}	6.96e-2/6.97e-2	2.66e-3/2.66e-3	7.20e-2/7.22e-2	38.5 min	
			<i>flip</i>	2^{-15}	2^{-10}	2.10e-3/2.10e-3	8.43e-2/8.44e-2	8.62e-2/8.64e-2	38.6 min	
			<i>flip</i>	2^{-15}	2^{-15}	2.26e-3/2.26e-3	2.65e-3/2.65e-3	4.90e-3/4.90e-3	38.5 min	
$(1,1)\text{-HPC}_1^C$ [27]	180	104	<i>set</i>	2^{-10}	2^{-10}	5.68e-2/5.69e-2	1.61e-1/1.61e-1	2.09e-1/2.09e-1	46.6 min	
			<i>set</i>	2^{-10}	2^{-15}	5.68e-2/5.69e-2	5.48e-3/5.48e-3	6.20e-2/6.21e-2	47.0 min	
			<i>set</i>	2^{-15}	2^{-10}	1.83e-3/1.83e-3	1.61e-1/1.61e-1	1.63e-1/1.63e-1	46.5 min	
			<i>set</i>	2^{-15}	2^{-15}	1.83e-3/1.83e-3	5.48e-3/5.48e-3	7.30e-3/7.30e-3	46.4 min	
			<i>reset</i>	2^{-10}	2^{-10}	6.41e-2/6.43e-2	1.61e-1/1.61e-1	2.15e-1/2.15e-1	40.5 min	
			<i>reset</i>	2^{-10}	2^{-15}	6.41e-2/6.43e-2	5.48e-3/5.48e-3	6.93e-2/6.94e-2	40.5 min	
			<i>reset</i>	2^{-15}	2^{-10}	2.07e-3/2.07e-3	1.61e-1/1.61e-1	1.63e-1/1.63e-1	41.0 min	
			<i>reset</i>	2^{-15}	2^{-15}	2.07e-3/2.07e-3	5.48e-3/5.48e-3	7.54e-3/7.54e-3	40.6 min	
			<i>flip</i>	2^{-10}	2^{-10}	6.41e-2/6.42e-2	1.61e-1/1.61e-1	2.15e-1/2.15e-1	39.1 min	
			<i>flip</i>	2^{-10}	2^{-15}	6.41e-2/6.42e-2	5.48e-3/5.48e-3	6.92e-2/6.93e-2	39.1 min	
			<i>flip</i>	2^{-15}	2^{-10}	2.07e-3/2.07e-3	1.61e-1/1.61e-1	1.63e-1/1.63e-1	39.0 min	
			<i>flip</i>	2^{-15}	2^{-15}	2.07e-3/2.07e-3	5.48e-3/5.48e-3	7.54e-3/7.54e-3	39.0 min	
$(1,1)\text{-SININA}$ [40,19]	198	122	<i>set</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	46.0 min	
			<i>set</i>	2^{-10}	2^{-15}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	46.2 min	
			<i>set</i>	2^{-15}	2^{-10}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	46.1 min	
			<i>set</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	46.1 min	
			<i>reset</i>	2^{-10}	2^{-10}	9.99e-1/9.99e-1	2.99e-4/4.52e-2	9.99e-1/9.99e-1	48.2 min	
			<i>reset</i>	2^{-10}	2^{-15}	9.99e-1/9.99e-1	8.99e-6/1.34e-3	9.99e-1/9.99e-1	47.7 min	
			<i>reset</i>	2^{-15}	2^{-10}	1.00e-0/1.00e-0	8.11e-3/4.52e-2	1.00e-0/1.00e-0	47.6 min	
			<i>reset</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	2.44e-4/1.34e-3	1.00e-0/1.00e-0	48.1 min	
			<i>flip</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	45.0 min	
			<i>flip</i>	2^{-10}	2^{-15}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	45.0 min	
			<i>flip</i>	2^{-15}	2^{-10}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	44.9 min	
			<i>flip</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	44.9 min	

Appendix F Correctness and Comparison Between Tools

In this section, we aim to compare the results of both tools for two main purposes: verifying their correctness and comparing their verification time on concrete ex-

Table 5. RCC_{KF} analysis of three gadgets with different fault and leakage probabilities on IronMask^+ . Here, all faults and wire leakages happen with the same indicated probability.

Design	#Loc.		Model			Probabilities			Time	
	FIA	SCA	τ	q	p	$\mu_{\min}/\mu_{\max}$	$\epsilon_{\min}/\epsilon_{\max}$	$\gamma_{\min}/\gamma_{\max}$	t	
(1, 1)-CPC ₁ [25]	182	316	<i>set</i>	2^{-10}	2^{-10}	8.89e-2/8.95e-2	1.62e-1/1.66e-1	2.37e-1/2.41e-1	21h 42min	38s
			<i>set</i>	2^{-10}	2^{-15}	8.89e-2/8.95e-2	8.47e-3/8.48e-3	9.66e-2/9.73e-2	21h 42min	19s
			<i>set</i>	2^{-15}	2^{-10}	2.56e-3/2.56e-3	1.61e-1/1.65e-1	1.63e-1/1.67e-1	21h 41min	46s
			<i>set</i>	2^{-15}	2^{-15}	2.93e-3/2.93e-3	5.66e-3/5.66e-3	8.57e-3/8.57e-3	21h 41min	48s
			<i>reset</i>	2^{-10}	2^{-10}	8.89e-2/8.96e-2	1.62e-1/1.65e-1	2.36e-1/2.40e-1	20h 33min	24s
			<i>reset</i>	2^{-10}	2^{-15}	8.89e-2/8.96e-2	7.50e-3/7.51e-3	9.58e-2/9.64e-2	20h 33min	21s
			<i>reset</i>	2^{-15}	2^{-10}	2.59e-3/2.59e-3	1.61e-1/1.65e-1	1.63e-1/1.67e-1	20h 33min	35s
			<i>reset</i>	2^{-15}	2^{-15}	2.93e-3/2.93e-3	5.63e-3/5.63e-3	8.54e-3/8.54e-3	20h 33min	15s
(1, 1)-HPC ₁ [27]	188	322	<i>set</i>	2^{-10}	2^{-10}	3.68e-2/3.76e-2	9.93e-1/9.97e-1	9.93e-1/9.97e-1	23h 54min	38s
			<i>set</i>	2^{-10}	2^{-15}	3.68e-2/3.76e-2	9.96e-1/9.97e-1	9.96e-1/9.97e-1	23h 55min	04s
			<i>set</i>	2^{-15}	2^{-10}	1.16e-3/1.16e-3	9.96e-1/1.00e-0	9.96e-1/1.00e-0	23h 54min	58s
			<i>set</i>	2^{-15}	2^{-15}	1.16e-3/1.16e-3	1.00e-0/1.00e-0	1.00e-0/1.00e-0	23h 55min	18s
			<i>reset</i>	2^{-10}	2^{-10}	8.53e-2/8.60e-2	9.95e-1/1.00e-0	9.96e-1/1.00e-0	23h 32min	20s
			<i>reset</i>	2^{-10}	2^{-15}	8.17e-2/8.24e-2	9.99e-1/1.00e-0	9.99e-1/1.00e-0	23h 32min	26s
			<i>reset</i>	2^{-15}	2^{-10}	2.80e-3/2.80e-3	9.96e-1/1.00e-0	9.96e-1/1.00e-0	23h 33min	27s
			<i>reset</i>	2^{-15}	2^{-15}	2.68e-3/2.68e-3	1.00e-0/1.00e-0	1.00e-1/1.00e-0	23h 33min	10s
(1, 1)-SININA [40,19]	230	364	<i>set</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	3h 12min	53s
			<i>set</i>	2^{-10}	2^{-15}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	3h 12min	17s
			<i>set</i>	2^{-15}	2^{-10}	9.99e-1/1.00e-0	0.00e-0/0.00e-0	9.99e-1/1.00e-0	3h 25min	34s
			<i>set</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	0.00e-0/0.00e-0	1.00e-0/1.00e-0	3h 33min	52s
			<i>reset</i>	2^{-10}	2^{-10}	9.99e-1/1.00e-0	1.18e-4/1.09e-1	9.99e-1/1.00e-0	3h 21min	59s
			<i>reset</i>	2^{-10}	2^{-15}	9.99e-1/1.00e-0	3.75e-6/3.30e-3	9.99e-1/1.00e-0	3h 18min	32s
			<i>reset</i>	2^{-15}	2^{-10}	1.00e-0/1.00e-0	3.84e-3/1.09e-1	1.00e-0/1.00e-0	3h 13min	28s
			<i>reset</i>	2^{-15}	2^{-15}	1.00e-0/1.00e-0	1.22e-4/3.30e-3	1.00e-0/1.00e-0	3h 13min	49s

amples. We arbitrarily choose one of the gadgets previously tested, namely (1, 1)-CPC₁ [25], with *set* and *reset* faults and probabilities $q = 2^{-10}$ and $p = 2^{-10}$. For this specific test, unlike the previous sections, we consider the same design and structure in both tools to have a fair comparison. In particular, VERICA⁺ usually considers one copy of a wire per usage, while IronMask^+ simulates the repeated usage via special copy operations. Consequently, repeated usage of intermediate variables leads to a higher number of wires in IronMask^+ . To ensure a fair comparison, we artificially eliminate the copies in both tools, assuming that each wire has precisely one copy in the circuit. The results of this comparison are depicted in Table 6.

Table 6. Comparison of verification results on (1, 1)-CPC₁ [25] with probabilities $q = 2^{-10}$ and $p = 2^{-10}$ for 170 FIA and 176 SCA locations.

Tools	Fault Model	Probabilities			CPU Time
		$\mu_{\min}/\mu_{\max}$	$\epsilon_{\min}/\epsilon_{\max}$	$\gamma_{\inf}/\gamma_{\sup}$	
VERICA ⁺	<i>set</i>	8.89e-2/8.96e-2	8.77e-2/8.85e-2	1.69e-1/1.70e-1	57d 21h 45min 21s
IronMask^+		8.89e-2/8.95e-2	9.93e-2/1.00e-1	1.79e-1/1.81e-1	21h 17min 42s
VERICA ⁺	<i>reset</i>	8.89e-2/8.96e-2	8.76e-2/8.84e-2	1.69e-1/1.70e-1	63d 7h 0min 16s
IronMask^+		8.89e-2/8.96e-2	9.84e-2/9.92e-2	1.79e-1/1.80e-1	20h 44min 20s

We can see that the bounds $\mu_{\min}$ and $\mu_{\max}$ are identical for both tools. This consistency arises from the fact that copies are not taken into account for faults, as previously explained. Therefore, the presence or absence of copies has no impact on the values of $\mu_{\min}$ and $\mu_{\max}$. The alignment of these values across both tools instills confidence in the correctness of the algorithms, considering that each tool employs completely different strategies.

However, there are slight differences in $\epsilon_{\min}$ and $\epsilon_{\max}$ between the two tools. **VERICA⁺** employs an optimization based on the binomial tree traversal that enables it to achieve tighter lower bounds on the failure probability while maintaining the same upper bound for both tools. Consequently, $\epsilon_{\min}$ tends to be higher with **VERICA⁺** in both scenarios, resulting in tighter values. Regarding the upper bound on the failure probability, $\epsilon_{\max}$, they exhibit very close values, with a small discrepancy where the values computed with **IronMask⁺** are slightly higher. This divergence stems from the computation method for the upper bound, which leverages the previously computed lower bound and assumes that all non-tested tuples of probes result in failures. Since the lower bound computed in **IronMask⁺** is lower, with fewer coefficients actually computed and the remaining ones replaced by zero, the upper bound also shows a slight increase, with the uncomputed coefficients replaced by their upper bounds, which are binomial coefficients. In essence, while the upper bound of **VERICA⁺** is tighter, the discrepancy is minimal given the exponentially large number of non-tested tuples.

Finally, these results illustrate the increased efficiency of **IronMask⁺**, which evaluates symbolic variables using efficient rules, compared to **VERICA⁺**, which relies on an exhaustive approach to check the probing distribution. Thus, the tools offer two different valuable trade-offs in terms of accuracy versus efficiency.

Appendix G Acronyms

CA Combined Analysis

CDF Cumulative Distribution Function

CINI Combined-Isolating Non-Interference

C-(S)NI Combined (Strong) Non-Interference

CDF Cumulative Distribution Function

DAG Directed Acyclic Graph

DOM Domain-Oriented Masking

EM electromagnetic

- FIA** Fault Injection Analysis
- FINI** Fault-Isolating Non-Interference
- F-(S)NI** Fault (Strong) Non-Interference
- ISW** Ishai-Sahai-Wagner
- RCS_{KF}** Known-Fault Random Combined Secure
- RCC_{KF}** Known-Fault Random Combined Composability
- PINI** Probe-Isolating Non-Interference
- PDN** Power Distribution Network
- RPC** Random Probing Composability
- GRPC** General Random Probing Composability
- RFC** Random Fault Composability
- RRFC** Restricted Random Fault Composability
- RPCUF** Random Probing Composition under Faults
- SCA** Side-Channel Analysis
- (S)NA** (Strong) Non-Accumulation
- (S)NI** (Strong) Non-Interference
- (S)NINA** (Strong) Non-Interference Non-Accumulation
- RCS_{UF}** Unknown-Fault Random Combined Secure

Appendix F

New Techniques for Random Probing Security and Application to Raccoon Signature Scheme

Hereafter is appended the full version of our paper [\[BRR25\]](#), joint work with Matthieu Rivain, and Mélissa Rossi, published at **EUROCRYPT 2025**.

New Techniques for Random Probing Security and Application to Raccoon Signature Scheme

Sonia Belaïd, Matthieu Rivain, and Mélissa Rossi*

CryptoExperts, Paris, France
first.last@cryptoexperts.com

Abstract. The random probing model formalizes a leakage scenario where each wire in a circuit leaks with probability p . This model holds practical relevance due to its reduction to the noisy leakage model, which is widely regarded as the appropriate formalization for power and electromagnetic side-channel attacks.

In this paper, we present new techniques for designing efficient masking schemes that achieve tighter random probing security with lower complexity. First, we introduce the notion of *cardinal random probing composability* (Cardinal-RPC), offering a new trade-off between complexity and security for composing masking gadgets. Next, we propose a novel refresh technique based on a simple iterative process: randomly selecting and updating two shares with fresh randomness. While not perfectly secure in the standard probing model, this method achieves arbitrary cardinal-RPC security, making it a versatile tool for constructing random-probing secure circuits. Using this refresh, we develop additional basic gadgets (e.g., linear multiplication, addition, and copy) that satisfy the cardinal-RPC notion. Despite the increased complexity, the gains in security significantly outweigh the overhead, with the number of iterations offering useful flexibility.

To showcase our techniques, we apply them to lattice-based signatures. Specifically, we introduce a new random-probing composable gadget for sampling small noise, a key component in various post-quantum algorithms. To assess security in this context, we generalize the random probing security model to address auxiliary inputs and public outputs. We apply our findings to Raccoon, a masking-friendly signature scheme originally designed for standard probing security. We prove the secure composition of our new gadgets for key generation and signature computation, and show that our masking scheme achieves a superior security-performance tradeoff compared to previous approaches based on random probing expansion. To our knowledge, this is the first fully secure instantiation of a post-quantum algorithm in the random probing model.

Keywords: Post-quantum signature · Masking countermeasure · Random Probing Model · Raccoon Signature Scheme

1 Introduction

Most widely used cryptographic algorithms are considered secure against black-box attacks, where the adversary’s knowledge is restricted to certain inputs and outputs. However, as discovered in the late 1990s, their implementations on physical devices can be vulnerable to more sophisticated side-channel attacks. These attacks leverage physical emanations from the device, such as execution time, temperature, power consumption, or electromagnetic emissions during the algorithm’s execution. One of the most prevalent techniques to defend against side-channel attacks is masking, introduced independently in 1999 by Chari, Jutla, Rao, and Rohatgi [CJRR99], and by Goubin and Patarin [GP99]. In essence, the approach involves splitting each sensitive variable in an implementation into n shares, with $n - 1$ generated uniformly at random and the final share derived from the original value and the previous ones. Masking ensures that an adversary cannot recover the secret without all the shares.

* This work was done while the author was working at Agence Nationale de Sécurité des Systèmes d’Informations, Paris, France.

The probing model. To prove the security of masking schemes, it is essential to model the side-channel leakage an attacker could exploit, which leads to the concept of leakage models. One of the most well-known models is the d -probing model, introduced by Ishai, Sahai, and Wagner in [ISW03], which states that an attacker can gain access to the exact values of d intermediate variables, where d is a security parameter, but nothing more. The underlying intuition is that while all processed data may leak, combining noisy variables requires an exponential number of measurements as their number increases [CJRR99].

Most masking approaches consist in carefully composing small gadgets that are individually proven secure. This often necessitates using *refresh gadgets*, which functionally act as the identity function and help reduce share dependencies by introducing additional randomness. However, masking schemes secure in the d -probing model can become prohibitively expensive as the number of shares increases. While the complexity of linear operations scales linearly with the number of shares, the main bottleneck arises from non-linear operations, such as multiplying shared variables in $\mathbb{Z}_q$, which is common in post-quantum schemes. These operations significantly affect overall efficiency, complicating the balance between security and performance as the number of shares grows.

Masking-friendly post-quantum schemes. The use of masking in post-quantum NIST standards comes with a significant performance cost. While the slowdown remains manageable for the Crystals-Kyber key encapsulation mechanism [SAB⁺22], the impact on performance is much more severe for the Crystals-Dilithium signature scheme [LDK⁺22], as highlighted in [ABC⁺23,CGTZ23]. For other schemes, such as Falcon [PFH⁺22], designing efficient and provable masking in the d -probing model presents even greater challenges. To tackle the security-performance challenge, recent efforts have focused on designing masking-friendly schemes that reconsider the unmasked design to limit the use of non-linear operations. These schemes instead primarily rely on gadgets that can be efficiently masked and provably secure within the d -probing model. A first attempt to modify Falcon’s sampler in this purpose [EFG⁺22] was invalidated in [Pre23]. A more successful approach has emerged with the design of Raccoon [dEK⁺23], a masking-friendly analog to Crystals-Dilithium. While Raccoon produces larger signature sizes, it offers a significant speedup (by orders of magnitude for masked implementation). Similarly, a masking-friendly version of Falcon has been introduced in [EEN⁺24]. Both designs abandon the use of complex Gaussian distributions in favor of a simpler noise distribution based on small uniform additions.

The stronger random probing model. Although these schemes have made progress in reconciling security and efficiency, the d -probing model, on which they rely, has raised concerns about its practical relevance [BCPZ16]. The noisy leakage model, introduced by Prouff and Rivain [RP10] and inspired by [CJRR99], better reflects real-world devices by assuming all data leaks with noise. In 2014, Duc, Dziembowski, and Faust demonstrated that masking schemes that are secure under the conceptually-simpler *random probing model* are also secure against in the noisy leakage model [DDF14].

In this model, each wire leaks with a constant probability p , reflecting side-channel noise in practice. A circuit is secure in this model if the leakage can be simulated without knowledge of the secret, with a negligible failure probability, as originally formalized in [Ajt11]. This model enables provable security bounds against attackers, similar to traditional black-box security proofs. Furthermore, the random probing model not only addresses probing attacks but also captures horizontal attacks [BCPZ16], which exploit repeated manipulations of variables. Overall, this model provides significantly stronger theoretical security guarantees compared to the standard probing model. Several attempts have been made to design masking schemes secure in the random probing model [ISW03,Ajt11,ADF16,GJR18,AIS18,BCP⁺20]. However, most of these schemes are either impractical or fail to tolerate a constant leakage probability p . A notable breakthrough was made by Ananth, Ishai, and Sahai [AIS18], who introduced an expansion strategy built upon secure multi-party computation protocols. This approach was later refined by Belaïd et al. [BCP⁺20,BRT21,BRTV21], resulting in simpler *random probing expandable* gadgets that achieve arbitrary levels of random probing security for a fixed leakage probability p . Cassiers et al. further developed a tighter composition framework applied to the AES S-box in [CFOS21], though it struggles to scale to larger circuits and numbers of shares. In parallel, Berti, Faust, and Orlt [BFO23] introduced a generic compiler based on leakage diagrams, by employing a refresh gadget derived from Dziembowski et al. [DFZ19]. While promising, this approach requires specific

gadgets with carefully crafted composition properties to achieve efficient secure composition, making its generalization challenging. Jahandideh, Mennink, and Batina [JMB24] proposed an algebraic approach to evaluate random probing security, but it relies on estimations and assumes the adversary targets a single secret, limiting its applicability. As a result, extending these solutions to general secure circuit constructions remains non-trivial. All in all, no concrete instantiations exist for post-quantum schemes, which are expected to become widely adopted in the near future.

Our contributions. The primary goal of this paper is to make random probing security more practical for real-world implementations. We introduce new techniques which we apply to the use case of lattice-based signatures. Specifically, we focus our study on Raccoon [dEK⁺23], whose masking-friendly structure makes it an ideal starting point. Our work addresses all the key steps necessary to achieve this goal, ranging from extending security definitions to designing elementary gadgets and providing a comprehensive composition proof. More specifically, the contributions of this paper are as follows:

1. We first extend existing random-probing security notions, which already include various flavors (*e.g.* for composition and expansion), by adding two key features:
 - We extend the random-probing security framework to handle auxiliary inputs in the circuit. Such inputs are not masked and must hence tolerate some amount of leakage. In Raccoon’s masking proof, auxiliary inputs (a.k.a. “unshared inputs”) capture the small uniform randomness used for sampling noise. Additionally, we consider the possibility to expose public outputs, such as the public key in the key generation algorithm and the signature in the signing algorithm, as proposed in [BBE⁺18] in the context of standard probing security. The extended random-probing security notions are presented in Section 3.1.
 - We generalize the notion of random probing composability (RPC). Instead of a fixed threshold on the number of input and output shares, we consider all possible cardinalities of leaked output shares and required input shares for the simulation. This refined notion, termed *cardinal-RPC*, is detailed in Section 3.2. We also combine the concepts of auxiliary inputs/public outputs with cardinal-RPC in Section 3.3.
2. A crucial step in designing random probing secure gadgets is defining a refreshing gadget that randomizes the shares before performing the target operation. One of our key contributions is the design of a new random probing-friendly refresh gadget. The concept is simple: start with an encoding of zero and, in each iteration, add and subtract a random value to two randomly selected shares. After a certain number of iterations, the input to be refreshed is share-wise added to the result. As the number of iterations increase, the “quality” of the refreshing improves until a threshold is reached. In Section 4.1, we quantify this quality by introducing the new intermediary notion of *random probing partitioned* (RPP). We use this to prove that our new refresh is cardinal-RPC, with a tunable security advantage depending on the number of iterations. Taking a step forward, we integrate our new refresh design and composability definition into the necessary basic gadgets: addition, copy and linear multiplication (*i.e.* multiplication by a public value). We prove the cardinal-RPC security of these gadgets. These designs and proofs are detailed in Section 4.2.
3. In Section 5, we introduce the noise generation gadget essential for Raccoon. This design leverages new elementary gadgets to create an algorithm that combines small uniform random samples for generating Raccoon’s secrets. We employ two approaches for designing the secure gadget: (1) using *random probing expandable* (RPE) gadgets from [BCP⁺20, BRT21, BRTV21] and (2) using our new cardinal-RPC composition framework and gadgets. This section demonstrates a significant enhancement in complexity and randomness consumption with our composition proof and new gadgets.
4. Finally, in Section 6, we combine all concepts and gadgets to present a random-probing-secure instantiation of Raccoon, using parameters from the original scheme [dEK⁺23]. We evaluate both scenarios: (1) using expansion and (2) applying our general composition proof. Our detailed complexity analysis demonstrates that our new gadgets significantly enhance efficiency while achieving high levels of security. For example, for the Raccoon-128-16 instance with a leakage probability of 2^{-24} , we achieve 128-bit probing security with a randomness consumption that is between 10 and 20 times lower than with the

expansion strategy of [BCP⁺20] (the number of multiplications remaining similar in both approaches, while the number of additions scales with the randomness consumption).

All security and complexity results for our new proposal can be reproduced using the Python script available at <https://github.com/CryptoExperts/EC25-random-probing-Raccoon>.

2 Preliminaries

2.1 Notations

A probabilistic polynomial time algorithm (PPT) runs in time polynomial in the security parameter. Along the paper, $\mathbb{K}$ shall denote a finite field or a finite ring.¹ Sequences indexes are starting at 1. By convention, when the maximum index is 0, the sequence is empty, for example $x_1, \dots, x_0 = \emptyset$. For some tuple $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{K}^n$ and for some set $I \subseteq [n]$, the tuple $(x_i)_{i \in I}$ is denoted $\mathbf{x}|_I$. For an integer n , we denote $\mathcal{P}(n)$ the set of partitions of n , which is defined as:

$$\mathcal{P}(n) = \{(n_1, \dots, n_\ell) \mid \ell \in \mathbb{N} \wedge n_1 \geq n_2 \geq \dots \geq n_\ell \wedge n_1 + \dots + n_\ell = n\}.$$

Distributions. Two probability distributions, D_1 and D_2 , are ε -close, denoted as $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε , namely

$$\text{SD}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon,$$

where $p_{D_1}(\cdot)$ and $p_{D_2}(\cdot)$ represent the probability mass functions of D_1 and D_2 , respectively. For two random variables X_1 and X_2 , we write $X_1 \stackrel{\text{id}}{=} X_2$ when $\varepsilon = 0$ i.e. when X_1 and X_2 are identically distributed.

Envelopes. In the proofs throughout this paper, exact probabilities are often challenging to compute directly, so we instead rely on upper bounds. The notion of probability envelope will be heavily used in this paper. Specifically, for a random variable X following a discrete distribution $\mathcal{D}$, we define $\mathcal{E}$ as an upper envelope of the distribution. That is, for any value x , the probability that X takes the value x (i.e. $\mathbb{P}(X = x)$) is bounded above by $\mathcal{E}(x)$, written as $X \lesssim \mathcal{E}$. As we will consider probability distributions conditioned to different disjoint events, we will often consider collections of probability envelopes with sometimes complex indices. For instance, we consider collections $\mathcal{E} = (\mathcal{E}_{(j_1, \dots, j_m)})$ indexed by tuples $(j_1, \dots, j_m) \in [0, n]^m$. In other words, each possible tuple value gives rise to an envelope associated with a distribution indexed by the same tuple.

Sharing. We use the general notation of $\llbracket x \rrbracket = (x_1, \dots, x_n) \in \mathbb{K}^n$ for a sharing of a secret value x .

2.2 Linear Sharing, Circuits, and Gadgets

In the following, the n -linear decoding mapping, denoted by LinDec , refers to the function $\mathbb{K}^n \rightarrow \mathbb{K}$ defined as

$$\text{LinDec} : (x_1, \dots, x_n) \mapsto x_1 + \dots + x_n,$$

for any $n \in \mathbb{N}$ and $(x_1, \dots, x_n) \in \mathbb{K}^n$. We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $(\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \in (\mathbb{K}^n)^\ell$ the n -linear decoding mapping acts as

$$\text{LinDec} : (\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \mapsto (\text{LinDec}(\llbracket x_1 \rrbracket), \dots, \text{LinDec}(\llbracket x_\ell \rrbracket)).$$

¹ Along the paper, $\mathbb{K}$ denotes the base structure (ring or field) of the arithmetic circuits. We use the notation $\mathbb{K}$ as those circuits are defined over a field most of the time. But in the context of our application to Raccoon, $\mathbb{K}$ shall be the ring $\mathbb{Z}_q$ with q a non-prime integer.

Definition 1 (Linear Sharing from [BCP⁺20]). Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an n -sharing of x is a random vector $\llbracket x \rrbracket \in \mathbb{K}^n$ such that $\text{LinDec}(\llbracket x \rrbracket) = x$. It is said to be uniform if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $\llbracket x \rrbracket_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. An n -linear encoding is a probabilistic algorithm LinEnc which on input a tuple $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{K}^\ell$ outputs a tuple $\llbracket \mathbf{x} \rrbracket = (\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \in (\mathbb{K}^n)^\ell$ such that $\llbracket x_i \rrbracket$ is a uniform n -sharing of x_i for every $i \in [\ell]$.

An *arithmetic circuit* on a finite field (or finite ring) $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations on $\mathbb{K}$. The circuit is built from a base set of gates $\mathbb{B} = \{g : \mathbb{K}^\ell \rightarrow \mathbb{K}^m\}$, such as addition gates $(x_1, x_2) \mapsto x_1 + x_2$, multiplication gates $(x_1, x_2) \mapsto x_1 \cdot x_2$, and copy gates $x \mapsto (x, x)$. A *randomized arithmetic circuit* includes a random gate that outputs a fresh uniform random value from $\mathbb{K}$. For a randomized circuit C , when we write $C(\rho, \cdot)$, this refers to the corresponding de-randomized circuit with the random tape $\rho \in \{0, 1\}^*$ as input.

In the following, we shall call an $(n\text{-share}, \ell\text{-to-}m)$ *gadget*, a randomized arithmetic circuit that maps an input $\llbracket \mathbf{y} \rrbracket \in (\mathbb{K}^n)^\ell$ to an output $\llbracket \mathbf{y} \rrbracket \in (\mathbb{K}^m)^m$ such that $\mathbf{x} = \text{LinDec}(\llbracket \mathbf{x} \rrbracket) \in \mathbb{K}^\ell$ and $\mathbf{y} = \text{LinDec}(\llbracket \mathbf{y} \rrbracket) \in \mathbb{K}^m$ satisfy $\mathbf{y} = g(\mathbf{x})$ for some function g . We recall the definition of a circuit compiler in Appendix A (introduced in [BCP⁺20]).

2.3 Random Probing Security

Let $p \in [0, 1]$ be some constant leakage probability parameter, a.k.a. the *leakage rate*. In the p -random probing model, an evaluation of a circuit C leaks the value carried by each wire with a probability p , all the wire leakage events being mutually independent.

As in [BCP⁺20], we formally define the random-probing leakage of a circuit from the two following algorithms:

- The *leaking-wires sampler* takes as input a randomized arithmetic circuit C and a probability $p \in [0, 1]$, and outputs a set $\mathcal{W}$, denoted as

$$\mathcal{W} \leftarrow \text{LeakingWires}(C, p) ,$$

where $\mathcal{W}$ is drawn by including each wire label from the circuit C with probability p to $\mathcal{W}$ (where all the probabilities are mutually independent).

- The *assign-wires sampler* takes as input a deterministic arithmetic circuit C , a fixed random tape $\rho \in \{0, 1\}^*$, a set of wire labels $\mathcal{W}$ (subset of all the wire labels of C), and a shared input $\llbracket x \rrbracket$, and it outputs a $|\mathcal{W}|$ -tuple $\mathbf{w} \in \mathbb{K}^{|\mathcal{W}|}$, denoted as

$$\mathbf{w} \leftarrow \text{AssignWires}(C, \rho, \mathcal{W}, \llbracket x \rrbracket) ,$$

where $\mathbf{w}$ corresponds to the exact assignments of the wires with label in $\mathcal{W}$ for an evaluation of $C(\rho, \llbracket x \rrbracket)$.

Note that $\mathbf{w}$ is deterministically fixed from $C, \rho, \mathcal{W}$ and $\llbracket x \rrbracket$. Let us now induce a distribution for $\mathbf{w}$ as a random probing leakage with fixed leaked wires.

Definition 2 (Random Probing Leakage for Fixed Leaked Wires). Let C be a deterministic arithmetic circuit, p be a probability, $\mathcal{W}$ be a set of wire labels and $\llbracket x \rrbracket$ be a shared input of C . Let $|\rho|$ be the size of the random tape for C and R be a uniform random variable on $\{0, 1\}^{|\rho|}$. We define the leakage for $\mathcal{W}$ as the following random variable

$$\mathcal{L}_{\mathcal{W}}(C, \llbracket x \rrbracket) := \text{AssignWires}(C, R, \mathcal{W}, \llbracket x \rrbracket) .$$

Following [BCRT23], we shall say that a pair of vectors $(\llbracket x \rrbracket, \llbracket y \rrbracket) \in (\mathbb{K}^n)^2$ is *admissible* for a randomized 1-to-1 circuit C if there exists a random tape $\rho_{\llbracket x \rrbracket, \llbracket y \rrbracket}$ such that $\llbracket y \rrbracket = C(\rho_{\llbracket x \rrbracket, \llbracket y \rrbracket}, \llbracket x \rrbracket)$.

Definition 3 (Induced Random Probing Leakage for Fixed Leaked Wires). Let C be a deterministic arithmetic circuit, p be a probability and $\mathcal{W}$ be a set of wire labels. Let $(\llbracket x \rrbracket, \llbracket y \rrbracket)$ be an admissible pair of vectors for C and $R_{\llbracket x \rrbracket, \llbracket y \rrbracket}$ be a uniform random variable on the set of random tapes ρ such that $\llbracket y \rrbracket = C(\rho, \llbracket x \rrbracket)$. We define the leakage for $\mathcal{W}$ as the following random variable

$$\mathcal{L}_{\mathcal{W}}(C, \llbracket x \rrbracket, \llbracket y \rrbracket) := \text{AssignWires}(C, R_{\llbracket x \rrbracket, \llbracket y \rrbracket}, \mathcal{W}, \llbracket x \rrbracket) .$$

Let us now introduce the formal definition of the so-called random probing security (RPS for short).

Definition 4 (Random Probing Security). *A randomized arithmetic circuit C is (p, ϵ) -random probing secure with respect to encoding Enc if there exists a PPT simulator² Sim such that for every input x , the distribution of $\text{Sim}(C)$ is ϵ -close to the distribution of $\mathcal{L}_{\mathcal{W}}(C, \llbracket x \rrbracket)$ where*

$$\begin{aligned} \mathcal{W} &\leftarrow \text{LeakingWires}(C, p) && \text{Drawing the leaking wires.} \\ \llbracket x \rrbracket &\leftarrow \text{Enc}(x). && \text{Drawing an encoding of the input.} \end{aligned}$$

Informally, while the random probing allows to consider that a large amount of wires can leak at the same time, the leakage itself is balanced by the probability of such an event. For example, if $\mathcal{W}$ corresponds to the labels of all the shares of the input x , the simulator will not be able to draw a correct sample without knowing x . However, this event happens with a probability p^n (n being the number of shares) which becomes negligible as n grows.

3 Extensions of the Random Probing Security

In this section we introduce the notion of *random probing security with auxiliary inputs and public outputs* (RPS-AI-O) as well as associated notions for the secure composition of gadgets. We also introduce the notion of *cardinal random probing composability* (cardinal-RPC) which provides a new trade-off between complexity and tightness for the composition compared to existing notions. We finally provide general composition results in this framework.

3.1 Random Probing Security with Auxiliary Inputs and Public Outputs

The notion of random probing security with auxiliary inputs and public outputs is a generalization of the RPS notion (Definition 4). It considers a randomized arithmetic circuit with shared inputs and outputs (as in the standard RPS notion) with two extra powers for the simulator.

1. First, the circuit admits additional auxiliary inputs which are not in shared form and for which partial information is potentially leaked. This partial information is captured by letting the simulator requesting a set of coordinates for each auxiliary input. The cardinality of these sets follow a list of distributions which is a parameter of the achieved notion.
2. Secondly, additional (unmasked) outputs are fully given to the simulator.

We introduce these new features in order to relax the RPS and achieve better complexity-security trade-offs for cryptosystems using random nonces that can be partly leaked without compromising the security. A typical example is the masked lattice-based signature scheme Raccoon. In this signature, some random nonces can be exposed to a probing attacker without harming the security. In security proofs, these random nonces are treated not as shares but as additional inputs [dKPR24]. This is formalized in the following definition.

² In concrete instantiations, the simulator Sim starts with generating a set of leaking wires through, as explicated in other definitions. Specifically, Sim will be decomposed with another PPT algorithm Sim^* as follows

$$\begin{aligned} \text{Sim}(C) : \quad & \mathcal{W} \leftarrow \text{LeakingWires}(C, p) \\ & \text{out} \leftarrow \text{Sim}^*(C, \mathcal{W}). \end{aligned}$$

Experiment 1: RPS-AI-O

$(I_1, \dots, I_k) \leftarrow \text{Sim}_1(C)$ $\text{out} \leftarrow \text{Sim}_2(\mathbf{a}_1 _{I_1}, \dots, \mathbf{a}_k _{I_k}, \mathbf{z})$ $\text{return } (I_1, \dots, I_k, \text{out})$	The simulator chooses indexes for the auxiliary inputs. The simulator is given the values of the auxiliary inputs. Next, it returns simulated values for the leaking wires.

Definition 5 (Random Probing Security with Auxiliary Inputs and Public Outputs). Let $n, \ell, m, k, d, \alpha \in \mathbb{N}$. Let C be a randomized arithmetic circuit with the following input/output partition³

$$\begin{array}{ccc}
 \text{masked inputs} & \text{auxiliary inputs} & \text{masked outputs} \quad \text{public outputs} \\
 C : \underbrace{(\mathbb{K}^n)^\ell}_{\text{masked inputs}} \times \underbrace{(\mathbb{K}^\alpha)^k}_{\text{auxiliary inputs}} & \rightarrow & \underbrace{(\mathbb{K}^n)^m}_{\text{masked outputs}} \times \underbrace{\mathbb{K}^d}_{\text{public outputs}} \\
 ([\mathbf{x}], \mathbf{a}_1, \dots, \mathbf{a}_k) & \mapsto & ([\mathbf{y}], \mathbf{z}).
 \end{array}$$

Let $\mathcal{E} = \mathcal{E}_1, \dots, \mathcal{E}_k$ be a set of probability distribution envelopes over the discrete set $[0, \alpha]$.

The circuit C is $(p, \epsilon, \mathcal{E})$ -random probing secure with auxiliary inputs and public outputs ($(p, \epsilon, \mathcal{E})$ -RPS-AI-O) with respect to encoding Enc if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that, for every admissible pair $(([\mathbf{x}], \mathbf{a}_1, \dots, \mathbf{a}_k), ([\mathbf{y}], \mathbf{z}))$, the outputs of Experiment 1 are such that

1. $|I_1| \lesssim \mathcal{E}_1, \dots, |I_k| \lesssim \mathcal{E}_k$, and
2. $\text{out} \approx_\epsilon \underbrace{\mathcal{L}_W(C, ([\mathbf{x}], \mathbf{a}_1, \dots, \mathbf{a}_k), ([\mathbf{y}], \mathbf{z}))}_{\text{Induced random probing leakage}}$ where $W \leftarrow \text{LeakingWires}(C, p)$ and $[\mathbf{x}] \leftarrow \text{Enc}(\mathbf{x})$.

3.2 Cardinal Random Probing Composability

The notion of *random probing composability* (RPC) was initially introduced in [BCP⁺20]. In essence, a circuit composed of individual gadgets that satisfy the RPC property will itself be RPC, and, as a result, random probing secure. Specifically, a gadget is (t, p, ϵ) -RPC with some integer $t < n$ and $p, \epsilon \in [0, 1]$ if, given t output shares, the probability that more than t shares of each input are required to simulate the leakage of the gadget – where each wire leaks with probability p – along with the t output shares, is upper bounded by ϵ . Based on this property, a circuit C composed of (t, p, ϵ_i) -RPC gadgets $(G_i)_{1 \leq i \leq |C|}$ will itself be (t, p, ϵ) -RPC with $\epsilon = 1 - \prod_{1 \leq i \leq |C|} (1 - \epsilon_i)$. The formal RPC definition and the latter composition theorem are formally recalled in Appendix A.

In this paper, we shall refer to the original RPC notion as *threshold-RPC* because it is defined with respect to a threshold t on the tolerated leakage on the input and output sharings. An alternative notion, based on so-called *probe distribution tables* (PDT) and targeting tighter composition, was put forward in [CFOS21]. Instead of fixing a threshold t for both the number of output shares (to be simulated) and the number of input shares (required for the simulations), the PDT of a gadget considers each specific pair of sets of input shares and output shares. Specifically, each cell (I, J) in the table represents the probability that a particular set of input shares I (represented by the row) is required to simulate the leakage of a gadget – with each wire leaking with probability p – along with a corresponding set of output shares J (represented by the column). Based on such PDT for each gadget, the authors of [CFOS21] define several composition rules to upper bound the random probing security of the global circuit.

Similarly, we shall refer to the PDT notion as *general-RPC* because it is the most general RPC notion in the sense that it encompasses the original (threshold-)RPC notion as well as the cardinal-RPC notion that

³ To be more generic, one can consider different dimensions for the auxiliary inputs i.e. $\mathbb{K}^{\alpha_1}, \dots, \mathbb{K}^{\alpha_k}$ instead of $(\mathbb{K}^\alpha)^k$. The definition can be straightforwardly adapted.

Experiment 2: Cardinal-RPC

$\mathcal{W} \leftarrow \text{LeakingWires}(G, p)$ $I_1, \dots, I_\ell \leftarrow \text{Sim}_1(G, \mathcal{W}, J_1, \dots, J_m)$ $out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket_{I_1}, \dots, \llbracket x_\ell \rrbracket_{I_\ell})$ $\text{return } (I_1, \dots, I_\ell, out_1, out_2)$	<i>Drawing the leaking wires.</i> <i>The simulator is given the indexes of the leaking outputs. It chooses indexes for the inputs.</i> <i>The simulator is given the shares at desired indexes and returns simulated values for the leaking wires (out_1) and outputs (out_2)</i>

we propose here. The definition of general-RPC with our formalism is given in Appendix A. This notion provides tighter composition than the threshold-RPC notion, which can be viewed as a special case of the former. Specifically, the t -threshold-RPC security parameter ϵ of a gadget can be easily recalculated from its PDT. For each column with an output set of cardinality at most t , we compute the sum of the rows corresponding to input sets with cardinality strictly greater than t . The largest of these sums across all eligible columns determines the threshold-RPC security parameter ϵ .

The cardinal-RPC notion provides a practical trade-off between the threshold RPC notion and the general RPC notion. In essence, rather than using a fixed threshold t and for the number of shares required for each output and input, the cardinal-RPC notion considers the possible cardinalities for each of them. This makes it simpler than the general RPC notion, which considers all possible sets of shares for each input and output – a number that grows exponentially with the number of shares. On the other hand, the combinations of cardinalities only grow polynomially with the number of shares.

In the following definition, we use a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. It means that there are $(n+1)^m$ functions $\mathcal{E}_{j_1, \dots, j_m}$ such that

$$\forall j_1, \dots, j_m, i_1, \dots, i_\ell \in [0, n]^{m+\ell}, \quad \mathcal{E}_{j_1, \dots, j_m}(i_1, \dots, i_\ell) \in [0, 1].$$

Definition 6 (Cardinal Random Probing Composability). Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}$ represent a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E})$ -cardinal random probing composable ($(p, \mathcal{E})$ -cardinal-RPC) for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \dots, J_m)$ where $J_1 \subseteq [n], \dots, J_m \subseteq [n]$, the outputs of Experiment 2 are such that

1. $(|I_1|, \dots, |I_\ell|) \lesssim \mathcal{E}_{|J_1|, \dots, |J_m|}$, and
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, \llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{y} \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket_{J_1}, \dots, \llbracket y_m \rrbracket_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(G, p)$ and $\llbracket \mathbf{y} \rrbracket \leftarrow G(\llbracket \mathbf{x} \rrbracket)$.

We emphasize the broadness of Definition 6: every gadget can be cardinal RPC for some probability and envelope. Hence, it only makes sense to claim that a gadget is $(p, \mathcal{E})$ -cardinal-RPC for some explicit p and $\mathcal{E}$.

The security advantage for the threshold-RPC property can be derived from the distribution envelope of the cardinal-RPC property in a similar way that it is derived from the PDT of the general RPC notion as explained above. This implication is formally stated in Lemma 1 hereafter.

Although cardinal-RPC provides a looser bound compared to general-RPC, the two notions become equivalent in the case of a uniformly ℓ -to- m cardinal-RPC gadget. A gadget is uniformly cardinal-RPC when, for any $(t_1, \dots, t_\ell) \in [0, n]^\ell$, all input shares of cardinality $(t_1, \dots, t_\ell)$ – necessary for simulating

Experiment 3: Cardinal-RPC-AI-O

$(\llbracket \mathbf{y} \rrbracket, \mathbf{z}) \leftarrow G(\llbracket \mathbf{x} \rrbracket, \mathbf{a})$ $\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$ $(I_1, \dots, I_\ell), (L_1, \dots, L_k) \leftarrow \text{Sim}_1(G, \mathcal{W}, \mathbf{z}, J_1, \dots, J_m)$	<i>Drawing the public outputs.</i> <i>Drawing the leaking wires.</i> <i>The simulator is given the indexes of the leaking outputs. It chooses indexes for the shared and auxiliary inputs.</i>
$out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket _{I_1}, \dots, \llbracket x_\ell \rrbracket _{I_\ell}, \mathbf{a}_1 _{L_1}, \dots, \mathbf{a}_k _{L_k})$	<i>The simulator is given the shares at desired indexes and returns simulated values for the leaking wires and outputs.</i>
$\text{return } (out_1, out_2, (I_1, \dots, I_\ell), (L_1, \dots, L_k))$	

both the leakage and the specified output – occur with equal probability. Furthermore, this equal probability distribution also applies to outputs of the same cardinality, ensuring symmetry in the simulation. The formal definition is given in Appendix A.

3.3 Cardinal-RPC with Auxiliary Inputs and Public Outputs

We now extend the previously introduced notion of cardinal-RPC to gadgets with auxiliary inputs and public outputs. The following definition is naturally obtained from cardinal-RPC (Definition 6) and RPS-AI-O (Definition 5).

Definition 7 (Cardinal RPC with Auxiliary Inputs and Public Outputs).

Let $n, \ell, m, k, \alpha, d \in \mathbb{N}$. Let $\mathcal{E}$ represent a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. Let $\mathcal{E}'$ represent a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, \alpha]^k$. Let G be a gadget with the following input/output partition

$$G : \overbrace{(\mathbb{K}^n)^\ell}^{\text{masked inputs}} \times \overbrace{(\mathbb{K}^\alpha)^k}^{\text{auxiliary inputs}} \rightarrow \overbrace{(\mathbb{K}^n)^m}^{\text{masked outputs}} \times \overbrace{\mathbb{K}^d}^{\text{public outputs}}$$

$$(\llbracket \mathbf{x} \rrbracket, \mathbf{a}_1, \dots, \mathbf{a}_k) \mapsto (\llbracket \mathbf{y} \rrbracket, \mathbf{z}).$$

The gadget G is $(p, \mathcal{E}, \mathcal{E}')$ -cardinal random probing composable with auxiliary inputs and public outputs $((p, \mathcal{E}, \mathcal{E}')$ -cardinal-RPC-AI-O) for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$, auxiliary input $\mathbf{a}_1, \dots, \mathbf{a}_k \in (\mathbb{K}^\alpha)^k$, and for every set collection $(J_1, \dots, J_m)$ where $J_1 \subseteq [n], \dots, J_m \subseteq [n]$, the outputs of Experiment 3 are such that

1. $(|I_1|, \dots, |I_\ell|) \lesssim \mathcal{E}_{|J_1|, \dots, |J_m|}$,
2. $(|L_1|, \dots, |L_k|) \lesssim \mathcal{E}'_{|J_1|, \dots, |J_m|}$, and
- 3.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, (\llbracket \mathbf{x} \rrbracket, \mathbf{a}), (\llbracket \mathbf{y} \rrbracket, \mathbf{z}))}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right)$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$ and $(\llbracket \mathbf{y} \rrbracket, \mathbf{z}) \leftarrow G(\llbracket \mathbf{x} \rrbracket, \mathbf{a})$.

Similarly, one can define threshold-RPC-AI-O from the original threshold-RPC definition. See Definition 17 in appendix for details.

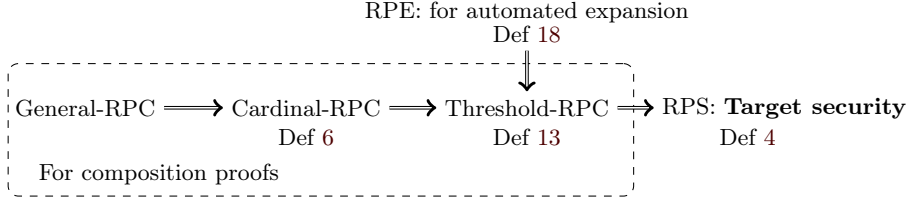


Fig. 1: Schematic implications between security notions

3.4 Summary of the notions

There exist many implications between the presented security notions. The main ones are shown in the schematic graph of Figure 1.

Lemma 1 (Cardinal-RPC $\implies$ Threshold-RPC). *Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}$ represent a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. If an n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E})$ -cardinal-RPC for some $p \in [0, 1]$, then it is (t, p, ϵ) -threshold-RPC for the same probability p and for any integer $t \in [0, n - 1]$, with*

$$\epsilon \leq \max_{(J_1, \dots, J_m) \in \mathbf{J}(t)} \left(\sum_{(I_1, I_2, \dots, I_\ell) \in \mathbf{I}(t)} \mathcal{E}_{|J_1|, \dots, |J_m|}(|I_1|, \dots, |I_\ell|) \right)$$

with $\mathbf{I}(t) = [(I_1, I_2, \dots, I_\ell) \in [0, n]^\ell \text{ such that } \exists i \in [1, \ell], |I_i| > t]$,

and $\mathbf{J}(t) = [(J_1, J_2, \dots, J_m) \in [0, n]^m \text{ such that } |J_1| \leq t, |J_2| \leq t, \dots, |J_m| \leq t]$.

Naturally, an n -share cardinal-RPC gadget is also cardinal-RPC-AI-O with auxiliary inputs of cardinality that is a multiple of n . This implication is established in Lemma 2 and follows directly from the definitions and the application of the law of total probability.

Lemma 2 (Cardinal-RPC-O $\implies$ Cardinal-RPC-AI-O). *Let $n, \ell, m, k, d \in \mathbb{N}$. Let $\mathcal{E}$ represent a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, n]^{\ell+k}$. If a gadget $G : (\mathbb{K}^n)^{\ell+k} \rightarrow (\mathbb{K}^n)^m \times \mathbb{K}^d$ is $(p, \mathcal{E})$ -cardinal-RPC-O, then it is also $(p, \mathcal{E}^x, \mathcal{E}^a)$ -cardinal-RPC-AI-O where $\mathcal{E}^x$ and $\mathcal{E}^a$ are defined, for all $j_1, \dots, j_m$, for all $i_1, \dots, i_\ell \in [0, n]^\ell$, for all $i'_1, \dots, i'_k \in [0, n]^k$ as*

$$\mathcal{E}_{j_1, \dots, j_m}^x(i_1, \dots, i_\ell) = \sum_{(t_1, \dots, t_k) \in [0, n]^k} \mathcal{E}_{j_1, \dots, j_m}(i_1, \dots, i_\ell, t_1, \dots, t_k),$$

$$\mathcal{E}_{j_1, \dots, j_m}^a(i'_1, \dots, i'_k) = \sum_{(t_1, \dots, t_\ell) \in [0, n]^\ell} \mathcal{E}_{j_1, \dots, j_m}(t_1, \dots, t_\ell, i'_1, \dots, i'_k).$$

While the cardinal-RPC-AI-O property is introduced for compositional purposes, it is important to highlight that a cardinal-RPC-AI-O circuit is also threshold-RPC-AI-O (following similar arguments to those of Lemma 1) and RPS-AI-O, as formalized in Lemma 3.

Lemma 3 (Cardinal-RPC-AI-O $\implies$ RPS-AI-O). *Following Definition 7, let $n, \ell, m, k, \alpha \in \mathbb{N}$. Let G be a gadget with the following input/output partition $(\mathbb{K}^n)^\ell \times (\mathbb{K}^\alpha)^k \rightarrow (\mathbb{K}^n)^m \times \mathbb{K}^d$. Let $\mathcal{E}$ be a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. Let $\mathcal{E}'$ be a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, \alpha]^k$. If the gadget G is $(p, \mathcal{E}, \mathcal{E}')$ -cardinal-RPC-AI-O, then it is also $(p, \epsilon, \mathcal{E}^*)$ -RPS-AI with*

$$\epsilon \geq 1 - \sum_{(n_1, n_2, \dots, n_\ell) \in [0, n-1]^\ell} \mathcal{E}_{(0, \dots, 0)}(n_1, n_2, \dots, n_\ell) \quad \text{and} \quad (\mathcal{E}_1^*, \dots, \mathcal{E}_k^*) = \mathcal{E}'_{(0, \dots, 0)}.$$

Proof. Let us assume that there exists a simulator $(\text{Sim}_1, \text{Sim}_2)$ for Experiment 3 and build a simulator $(\text{Sim}_1^*, \text{Sim}_2^*)$ for Experiment 1. These simulators $(\text{Sim}_1^*, \text{Sim}_2^*)$ are defined as follows.

$\text{Sim}_1^*(G, \mathcal{W}, \mathbf{z}) :$ $(I_1, \dots, I_\ell), (L_1, \dots, L_k) \leftarrow \text{Sim}_1(C, \mathcal{W}, \mathbf{z}, \emptyset, \dots, \emptyset)$
 keep $(|I_1|, \dots, |I_\ell|)$ in the internal state
 return $(L_1, \dots, L_k)$

$\text{Sim}_2^*(\mathbf{a}_1|_{I_1}, \dots, \mathbf{a}_k|_{I_k}) :$ get $(|I_1|, \dots, |I_\ell|)$ from the internal state
 for $i \in [1, \ell]$:
 $r_i \leftarrow (\$, \dots, \$) \in [1, n]^{|I_i|}$
 $\text{out}_1, \text{out}_2 \leftarrow \text{Sim}_2(r_1, \dots, r_\ell, \mathbf{a}_1|_{I_1}, \dots, \mathbf{a}_k|_{I_k})$
 return out_2

The simulators $(\text{Sim}_1^*, \text{Sim}_2^*)$ will succeed if the cardinals of $\mathbf{I}$ are all in $[0, n-1]$. Indeed, in that case, the shares can be simulated as uniform random values without knowledge of the inputs. The distinguishing advantage ϵ is then

$$\epsilon \geq 1 - \sum_{(n_1, n_2, \dots, n_\ell) \in [0, n-1]^\ell} \mathcal{E}_{(0, \dots, 0)}(n_1, n_2, \dots, n_\ell).$$

The probability envelopes are an application of Condition 2 of Definition 7 with $\mathbf{J} = (\emptyset, \dots, \emptyset)$. □

Property 1 (RPx-AI-O $\implies$ RPx). Let RPx be a security notion $\in \{\text{RPS}, \text{threshold-RPC}, \text{cardinal-RPC}\}$ (see Definitions 4, 13 and 6 respectively). The security RPx is a particular case of RPx-AI-O (see Definitions 5, 17 and 7 respectively) where $d = k = 0$.

Definition 8 (RPx-AI and RPx-O). Let RPx be a security notion $\in \{\text{RPS}, \text{RPC}, \text{threshold-RPC}, \text{cardinal-RPC}\}$. We denote by RPx-AI (resp. RPx-O) the particularly case of RPx-AI-O where $d = 0$ (resp. $k = 0$).

3.5 Composition Results

In the following, to construct cardinal-RPC-AI-O secure circuits, we will begin by considering the auxiliary inputs as regular shared inputs of base cardinal-RPC gadgets, using a padding that we will define later. Next, we will compose these cardinal-RPC gadgets using Lemma 4 and variants.

Lemma 4. Let $n \in \mathbb{N}$ and $p \in [0, 1]$. Let $G_1 : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ be a $(p, \mathcal{E}^1)$ -cardinal-RPC gadget for $\mathcal{E}^1$ a collection of $n+1$ probability envelopes over $[0, n]^2$, and let $G_2 : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ be a $(p, \mathcal{E}^2)$ -cardinal-RPC gadget for $\mathcal{E}^2$ a collection of $n+1$ probability envelopes over $[0, n]^2$. The sequential composition $G_3 : (\mathbb{K}^n)^3 \rightarrow \mathbb{K}^n$ depicted in Figure 2 is $(p, \mathcal{E}^3)$ -cardinal-RPC where $\mathcal{E}^3$ is a collection of $n+1$ probability envelopes over $[0, n]^3$ verifying $\forall \mathbf{t}_{in} = (t_{in}^1, t_{in}^2, t_{in}^3) \in [0, n]^3$ and $\forall t_{out} \in [0, n]$,

$$\mathcal{E}_{t_{out}}^3(\mathbf{t}_{in}) = \sum_{i=0}^n \mathcal{E}_i^1(t_{in}^1, t_{in}^2) \cdot \mathcal{E}_{t_{out}}^2(t_{in}^3, i).$$

Proof. Let us assume that there exists simulators $(\text{Sim}_1^1, \text{Sim}_2^1)$ for Experiment 2 for G_1 and $(\text{Sim}_1^2, \text{Sim}_2^2)$ for G_2 . We aim at building a simulator $(\text{Sim}_1^*, \text{Sim}_2^*)$ for Experiment 2 for G_3 .

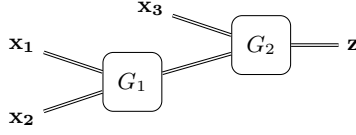


Fig. 2: Composition between $G_1 : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ and $G_2 : (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$.

Let us define $(\text{Sim}_1^*, \text{Sim}_2^*)$ as follows.

$\text{Sim}_1^*(G, \mathcal{W}, J) :$ Separate $\mathcal{W}$ in $\mathcal{W}_1, \mathcal{W}_2$ for leaking wires in G_1 and G_2
 $(I_1^2, I_2^2) \leftarrow \text{Sim}_1^2(C, \mathcal{W}_2, J)$
 $(I_1^1, I_2^1) \leftarrow \text{Sim}_1^1(C, \mathcal{W}_1, I_2^2)$
 return (I_1^1, I_2^1, I_1^2) .

$\text{Sim}_2^*(\llbracket x_1 \rrbracket|_{I_1}, \llbracket x_2 \rrbracket|_{I_2}, \llbracket x_3 \rrbracket|_{I_3}) :$ $out_1^1, out_2^1 \leftarrow \text{Sim}_2^1(\llbracket x_1 \rrbracket|_{I_1}, \llbracket x_2 \rrbracket|_{I_2})$
 $out_1^2, out_2^2 \leftarrow \text{Sim}_2^2(\llbracket x_3 \rrbracket|_{I_3}, out_2^1)$
 return $(out_1^1, out_2^1, out_2^2)$

Let us compute the distribution envelope. We denote by $I_{i,j} = |I_j^i|$ and $O = |J|$ where I_j^i and J are defined in Sim_1^* . By definition, $\forall \mathbf{t}_{\text{in}} = (t_{\text{in}}^1, t_{\text{in}}^2, t_{\text{in}}^3) \in [0, n]^3$, $\mathcal{E}_{t_{\text{out}}}^3(\mathbf{t}_{\text{in}})$ is an upper bound on the following probability

$$p_c = \mathbb{P}(I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2 \wedge I_{2,1} = t_{\text{in}}^3 | O = t_{\text{out}}).$$

From the law of total probability and the Bayes formula, we have

$$\begin{aligned} p_c &= \sum_{i=0}^n \mathbb{P}((I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2 \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i) | O = t_{\text{out}}) \\ &= \sum_{i=0}^n \mathbb{P}((I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2) | (O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i)) \\ &\quad \cdot \mathbb{P}((I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i) | O = t_{\text{out}}) \end{aligned}$$

We now introduce the random variable $\mathcal{Y}$ which represents the subset of output shares from G_1 that are required for the subsequent simulation:

$$\begin{aligned} p_c &= \sum_{i=0}^n \sum_{Y \subseteq [0, n]} \mathbb{P}((I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2) | (O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i \wedge \mathcal{Y} = Y)) \\ &\quad \cdot \mathbb{P}((\mathcal{Y} = Y) | O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i) \\ &\quad \cdot \mathbb{P}((I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i) | O = t_{\text{out}}) \end{aligned}$$

The events $I_{1,1} = t_{\text{in}}^1$ and $I_{1,2} = t_{\text{in}}^2$ in the first term depend only on the output set $\mathcal{Y}$ (and its cardinality) and not on the subsequent leakage anymore, hence we can simplify the expression:

$$\begin{aligned} p_c &= \sum_{i=0}^n \sum_{\substack{Y \subseteq [0, n] \\ |\mathcal{Y}|=i}} \mathbb{P}((I_{1,1} = t_{\text{in}}^1 \wedge I_{1,2} = t_{\text{in}}^2) | (\mathcal{Y} = Y)) \\ &\quad \cdot \mathbb{P}((\mathcal{Y} = Y) | O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i) \\ &\quad \cdot \mathbb{P}((I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i) | O = t_{\text{out}}) \end{aligned}$$

Algorithm 1 RPZeroEnc $^\gamma$

Output: $\llbracket z \rrbracket = (z_1, \dots, z_n) \in \mathbb{K}^n$ such that $z_1 + \dots + z_n = 0$

1. $\llbracket z \rrbracket = (0, 0, \dots, 0)$ $\{n \text{ zeros}\}$
2. **for** $i = 1$ to γ **do**
3. Select two distinct indices $i_1, i_2 \in [1, n]$ uniformly at random
4. $r \xleftarrow{\$} \mathbb{K}$
5. $z_{i_1} \leftarrow z_{i_1} + r$
6. $z_{i_2} \leftarrow z_{i_2} - r$
7. **end for**
8. **return** $\llbracket z \rrbracket$

Algorithm 2 RPRefresh $^\gamma(\llbracket a \rrbracket)$

Input: $\llbracket a \rrbracket = (a_1, \dots, a_n) \in \mathbb{K}^n$

Output: $\llbracket s \rrbracket = (s_1, \dots, s_n) \in \mathbb{K}^n$ such that $s_1 + \dots + s_n = a_1 + \dots + a_n$

1. $\llbracket z \rrbracket \leftarrow \text{RPZeroEnc}^\gamma()$
2. **for** $i = 1$ to n **do**
3. $s_i \leftarrow a_i + z_i$
4. **end for**
5. **return** $\llbracket s \rrbracket$

To obtain an upper bound, we use the worst-case scenario for the first term, which corresponds to the envelope function of G_1 , and we bound the third term using the envelope function of G_2 , denoted by $\mathcal{E}_{t_{\text{out}}}^2(t_{\text{in}}^3, i)$:

$$p_c \leq \sum_{i=0}^n \mathcal{E}_i^1(t_{\text{in}}^1, t_{\text{in}}^2) \cdot \mathcal{E}_{t_{\text{out}}}^2(t_{\text{in}}^3, i) \cdot \sum_{\substack{Y \subseteq [0, n] \\ |Y|=i}} \mathbb{P}((\mathcal{Y} = Y) \mid O = t_{\text{out}} \wedge I_{2,1} = t_{\text{in}}^3 \wedge I_{2,2} = i)$$

The last summation term equals one, which concludes the proof. $\square$

4 New Random Probing Composable Gadgets

Now that we have defined our new security notions, we introduce cardinal-RPC *base* gadgets, which will be used to construct random probing secure circuits. We begin with our new refresh algorithm in Section 4.1, which is based on a novel zero encoding and will serve as the foundation for our elementary linear operations – addition, multiplication by a constant, and copy – detailed in Section 4.2

4.1 A New Versatile Cardinal-RPC Refresh Gadget

Definition 9 (ZeroEncoding). Let $n \in \mathbb{N}$. A zero-encoding algorithm, denoted **ZeroEnc**, is a probabilistic algorithm sampling an n -sharing of 0. In other words, $\llbracket z \rrbracket \leftarrow \text{ZeroEnc}()$ such that $\sum z_i = 0$ (with probability 1).

We here introduce our new refresh gadget, referred to as **RPRefresh**, for *random-pair refresh*, which is described in Algorithm 2. This gadget is based on the general construction which consists in refreshing the input sharing by the addition of a sharing of 0. The novelty of our refresh gadget lies in the underlying gadget **RPZeroEnc** to generate a sharing of 0, which is detailed in Algorithm 1. This gadget is parameterized by a number γ of iterations, which influences both its complexity and its security (up to a certain limit).

The process begins with the initialization of a constant all-0 sharing. During each iteration, two share indices are generated uniformly at random⁴ and a fresh random value is respectively added to and subtracted from the corresponding shares. After all iterations have been completed, the sharing of zero produced by **RPZeroEnc** is added to the input value in **RPRefresh**, share by share.

While its conceptual idea is simple, this refreshing technique never outputs a perfectly fresh sharing of zero because of the probabilistic nature of the refreshed indexes. In a first step, let us discuss the quality of the output while leaving probing security apart.

⁴ We deviate here from the classical circuit model by introducing the generation of random indices. However, this can be interpreted as sampling a specific circuit (instantiated with our uniform indices) that we fully disclose to the attacker.

Definition 10 (Partition ZeroEncoding). For $P = (n_1, \dots, n_\ell) \in \mathcal{P}(n)$ an integer partition, we define $\text{PartitionZeroEnc}(P)$ as a particular type of distribution producing a sharing $\llbracket z \rrbracket = \llbracket z_1 \rrbracket \parallel \dots \parallel \llbracket z_\ell \rrbracket$ where $\llbracket z_i \rrbracket$ is a fresh n_i -sharing of 0 for every $i \in [1, \ell]$. In other words,

$$\text{PartitionZeroEnc}(n_1, \dots, n_\ell) = \underbrace{z_1, \dots, z_{n_1}}_{\sum z_i = 0}, \underbrace{z_{n_1+1}, \dots, z_{n_1+n_2}}_{\sum z_i = 0}, \dots, \underbrace{z_{n-n_\ell+1}, \dots, z_n}_{\sum z_i = 0}.$$

The two extreme cases in Definition 10 are (worst case) for $P = (1, \dots, 1)$ where $\text{PartitionZeroEnc}(1, \dots, 1) = (0, \dots, 0)$ (the poorest output quality), and (ideal case) for the singleton $P = (n)$ where $\text{PartitionZeroEnc}(n)$ is a uniform sharing of 0 (the best output quality).

Lemma 5. Let $n, \gamma \in \mathbb{N}$. Let ρ be a random tape. There exists a partition $P_{\gamma, \rho} \in \mathcal{P}(n)$, a permutation function $\sigma_{\gamma, \rho} : [1, n] \rightarrow [1, n]$ and another random tape $\rho'_{\gamma, \rho}$ such that

$$\text{RPZeroEnc}^\gamma(\rho) = \sigma_{\gamma, \rho}(\text{PartitionZeroEnc}(P_{\gamma, \rho}, \rho'_{\gamma, \rho}))$$

where RPZeroEnc is depicted in Algorithm 1.

This lemma follows from the definition of RPZeroEnc . Let $\gamma \in \mathbb{N}$ and the random tape ρ be fixed parameters. Here is an intuition of an iterative construction of the partition, denoted P , and the permutation, denoted σ . Before the first iteration, the variable $\llbracket z \rrbracket = (0, \dots, 0)$ is exactly the output of

$$\text{RPZeroEnc}(\underbrace{1, 1, \dots, 1}_{n \text{ times}}).$$

Suppose now that the generated indices in the first iteration are $(i_1, i_2) = (1, 2)$, then we have $\llbracket z \rrbracket = (r_1, -r_1, \dots, 0)$ for $r_1 \leftarrow \$$ which is identically distributed to

$$\text{RPZeroEnc}(2, \underbrace{1, \dots, 1}_{n-2 \text{ times}}).$$

If the generated indices were different from $(1, 2)$, we would have a distribution identical to $\sigma(\text{PartitionZeroEnc}(2, 1, \dots, 1))$ for the same partition but with some coordinate permutation σ . Now for the second iteration, let $r_2 \leftarrow \$$,

$$\begin{aligned} \text{if } (i_1, i_2) &= (1, 2), \llbracket z \rrbracket = (r_1 + r_2, r_1 - r_2, 0 \dots, 0), \\ \text{if } (i_1, i_2) &= (2, 3), \llbracket z \rrbracket = (r_1, -r_1 + r_2, r_2, 0 \dots, 0), \\ \text{if } (i_1, i_2) &= (3, 4), \llbracket z \rrbracket = (r_1, -r_1, r_2, -r_2, 0 \dots, 0). \end{aligned}$$

In these cases, $\llbracket z \rrbracket$ will be statistically indistinguishable to

$$\begin{aligned} &\text{RPZeroEnc}(2, \underbrace{1, \dots, 1}_{n-2 \text{ times}}) \text{ if } (i_1, i_2) = (1, 2), \\ &\text{RPZeroEnc}(3, \underbrace{1, \dots, 1}_{n-3 \text{ times}}) \text{ if } (i_1, i_2) = (2, 3), \\ &\text{RPZeroEnc}(2, 2, \underbrace{1, \dots, 1}_{n-4 \text{ times}}) \text{ if } (i_1, i_2) = (3, 4). \end{aligned}$$

All the cases are captured with a coordinate permutation in a similar way. On and on, one can construct a partition iteratively depending on the randomness of RPZeroEnc . The permutation σ captures the possibility of RPZeroEnc shuffling the order of the refreshed indices. As the number of iterations accumulates, the distribution of $\llbracket z \rrbracket$ becomes increasingly less partitioned, approaching the ideal partition $P = (n)$. This iterative construction is formalized in a more general case (in the presence of leakage) in Lemma 6.

Experiment 4: Random Probing Partition

$\mathcal{W} \leftarrow \text{LeakingWires}(\text{ZeroEnc}, p)$ $\rho \leftarrow \$$ $\llbracket z \rrbracket \leftarrow \text{ZeroEnc}(\rho)$ $\text{leak} \leftarrow \text{AssignWires}(\text{ZeroEnc}, \rho, \mathcal{W}, \emptyset)$ $\llbracket w \rrbracket \leftarrow \text{Sim}(\mathcal{W}, \text{leak})$ $(P, \sigma) := \phi(\mathcal{W})$ $\llbracket u \rrbracket \leftarrow \sigma(\text{PartitionZeroEnc}(P))$ $\text{return } (P, \llbracket z \rrbracket, \llbracket w \rrbracket, \llbracket u \rrbracket)$	<i>Drawing the leaking wires.</i> <i>Drawing a random tape.</i> <i>Drawing $\llbracket z \rrbracket$ an encoding of 0.</i> <i>Computing the leaked values.</i> <i>The leakages are given to the simulator. It outputs an encoding of 0 whose randomness is "weakened" due to the leakage.</i> <i>Computing a partition and a permutation corresponding to the leaking wires.</i> <i>Drawing an encoding of 0 corresponding to the "remaining" randomness to compensate the randomness that was given in the leakage.</i>

Let us now address the random probing security of RPZeroEnc . We will show below in Theorem 1 that RPZeroEnc is cardinal-RPC (we recall that this security notion was presented in Definition 6). To be able to establish the cardinal-RPC property, one needs to take into account the possible leaked iterations in the quality of the refreshing. Naturally, the more leaked randomness, the less qualitative the refreshing is. We introduce the intermediary notion of *random probing partitioned* (RPP) to characterize this quality with respect to the number of iterations and the leakage probability. We later prove the cardinal-RPC security of the refresh (Theorem 1) from the RPP security of the zero-encoding gadget.

Definition 11 (Random-Probing-Partition). Let $n \in \mathbb{N}$ and $\mathcal{P}(n)$ be the set of partitions of n . Let $\mathcal{E}$ be a distribution envelope over $\mathcal{P}(n)$. An n -share zero-encoding gadget $\text{ZeroEnc} : \emptyset \rightarrow \mathbb{K}^n$ is $(p, \mathcal{E})$ -random-probing-partitioned (RPP) for some $p \in [0, 1]$ if there exists a deterministic algorithm ϕ and a PPT simulator Sim such that the outputs of the random experiment 4 satisfy:

1. $P \lesssim \mathcal{E}$, and
2. $\llbracket z \rrbracket$ and $\llbracket w \rrbracket + \llbracket u \rrbracket$ are identically distributed.

Intuition of Definition 11. Note that Definition 11 is different from other security notions introduced in this paper. It does not ensure that the random probing leakage can be simulated. The simulator is instead given the leakage and the leaking wires. The idea of Definition 11 is to show that the output of ZeroEnc is such that

$$\llbracket z \rrbracket \approx \underbrace{\llbracket w \rrbracket}_{\text{leaked 0-encoding}} + \underbrace{\llbracket u \rrbracket}_{\text{fresh partitioned 0-encoding}}.$$

The sharing $\llbracket u \rrbracket$ hence captures the “remaining” randomness of the output sharing $\llbracket z \rrbracket$ given the internal leakage of the gadget. Let us further stress that $\llbracket u \rrbracket$ follows the random distribution $\sigma(\text{PartitionZeroEnc}(P))$ which randomness is highly related to the partition P . The envelope $\mathcal{E}$ upper-bounds the distribution of the partition P defined by $(P, \sigma) = \phi(\text{LeakingWires}(\text{ZeroEnc}, p))$.

The rationale for characterizing RPZeroEnc under the RPP notion is the following. Assume that, after a number of non-leaking iterations, we get a distribution $\llbracket z \rrbracket = \sigma(\text{PartitionZeroEnc}(P))$ for some partition P and coordinate permutation σ . If, during a subsequent iteration, the value of the random r leaks, the output $\llbracket z' \rrbracket$ of this iteration can be expressed as: $\llbracket z' \rrbracket = \llbracket z \rrbracket + \llbracket w \rrbracket$ where $\llbracket w \rrbracket$ is a vector with one coordinate set to r , one to $-r$, and the rest to 0. Given the leakage of r , one can perfectly simulate this $\llbracket w \rrbracket$. This gives the

intuition behind the RPP property of **RPZeroEnc**: non-leaking iterations contribute to reducing the partition structure of $\llbracket u \rrbracket = \sigma(\text{PartitionZeroEnc}(P))$, while leaking iterations result in updates to the $\llbracket w \rrbracket$ component, accumulating the leaked randomness. We note that the leakage can also concern z_{i_1} or z_{i_2} which results in an update of both $\llbracket u \rrbracket$ (or (P, σ)) and $\llbracket w \rrbracket$.

Before formally establishing the RPP security of **RPZeroEnc** in the next lemma, we introduce the concept of *transition set* of a partition. Assume the output of an iteration corresponds to a partition P . The transition set $\mathcal{T}_j(P)$ contains all partitions P' that can be obtained after an iteration with $i_1 = i_2 = j$ while the transition set $\mathcal{T}_{j_1, j_2}(P)$ contains all partitions P' that can be obtained when $i_1 = j_1 \neq i_2 = j_2$. These transition sets are identified in Table 1 and Table 2, considering the different possible scenarios in terms of leakage. In these tables, expressions such as $(z_{i_1} \oplus z_{i_2}) \wedge \bar{r}$ in the leakage column should be interpreted as a scenario where either z_{i_1} or z_{i_2} leaks (but not both) and r does not leak.

Lemma 6 (RPZeroEnc is RPP). *Let n, γ in $\mathbb{N}$ and $p \in [0, 1]$. The n -share zero-encoding gadget RPZeroEnc^γ is $(p, \mathcal{E}^\gamma)$ -RPP for some distribution envelope $\mathcal{E}^\gamma$ defined over $\mathcal{P}(n)$ constructively as follows. For $\gamma = 0$, we simply have $\mathcal{E}^0(\mathbf{1}) = 1$ and $\mathcal{E}^0(P) = 0$ for all $P \in \mathcal{P}(n) \setminus \{\mathbf{1}\}$, where $\mathbf{1} = (1, 1, \dots, 1)$. Then, for all $\gamma > 0$, we define $\mathcal{E}^{\gamma+1}$ from $\mathcal{E}^\gamma$ as follows:*

1. for all $P \in \mathcal{P}(n)$, we initiate $\mathcal{E}^{\gamma+1}(P)$ to 0,
2. for all $P \in \mathcal{P}(n)$, for all $j \in \{1, \dots, |P|\}$, for all $P' \in \mathcal{T}_j(P)$ from Table 1,

$$\mathcal{E}^{\gamma+1}(P') = \mathcal{E}^{\gamma+1}(P') + \mathbb{P}(P \rightarrow P') \cdot \mathcal{E}^\gamma(P),$$

3. for all $P \in \mathcal{P}(n)$, for all $j_1, j_2 \in \{1, \dots, |P|\}$ with $j_1 < j_2$, for all $P' \in \mathcal{T}_{j_1, j_2}(P)$ from Table 2,

$$\mathcal{E}^{\gamma+1}(P') = \mathcal{E}^{\gamma+1}(P') + \mathbb{P}(P \rightarrow P') \cdot \mathcal{E}^\gamma(P).$$

Proof. The **RPZeroEnc** algorithm begins with a shared vector initialized to $(0, \dots, 0)$. Since all shares are constant and known, and based on Definition 11, the only possible partition with strictly positive probability is the vector of ones.

At iteration $\gamma + 1$, two indices i_1 and i_2 are drawn uniformly at random from $\{1, \dots, n\}$. These indices can either belong to the same part n_j of a partition $P = (n_1, n_2, \dots, n_k)$ with probability

$$p_s = \frac{n_j \cdot (n_j - 1)}{n(n - 1)}$$

or to two different parts n_{j_1} and n_{j_2} with probability

$$p_d = 2 \cdot \frac{n_{j_1} \cdot n_{j_2}}{n(n - 1)}.$$

Next, a random value r is drawn uniformly from $\mathbb{K}$ and added to the two elements z_{i_1} and z_{i_2} . If neither of these elements leaks and both indices belong to the same part n_j , then the partition remains unchanged. However, if all these three elements are leaking and both indices belong to the same part n_j , then z_{i_1} and z_{i_2} are excluded from their zero encodings, forming two new independent parts. This scenario corresponds to the first row of Table 1 and occurs with probability

$$p_s \cdot p^2 \cdot (1 - (1 - p)^3),$$

where z_{i_1} and z_{i_2} each leak with probability p , and the value r , being manipulated three times, leaks with probability $1 - (1 - p)^3$.

All possible leakage scenarios, whether the indices are drawn from the same part or from different parts, are detailed in Tables 1 and 2. This demonstrates the construction of our envelope. $\square$

Table 1: Intermediate probabilities for Lemma 6, starting from a partition $P = (n_1, \dots, n_k)$ and when the drawn indices i_1 and i_2 belong to the same subset of cardinal n_j where $p_s := \frac{n_j \cdot (n_j - 1)}{n(n-1)}$.

Leakage	New partition $P' \in \mathcal{T}_j(P)$	Probability $\mathbb{P}(P \rightarrow P')$
$z_{i_1} \wedge z_{i_2} \wedge r$	$(1, 1, n_1, \dots, n_{j-1}, n_j - 2, n_{j+1}, \dots, n_k)$	$p_s \cdot p^2 \cdot (1 - (1-p)^3)$
$z_{i_1} \wedge z_{i_2} \wedge \bar{r}$	$(2, n_1, \dots, n_{j-1}, n_j - 2, n_{j+1}, \dots, n_k)$	$p_s \cdot p^2 \cdot (1-p)^3$
$(z_{i_1} \oplus z_{i_2}) \wedge r$	$(1, n_1, \dots, n_{j-1}, n_j - 1, n_{j+1}, \dots, n_k)$	$2 \cdot p_s \cdot (p - p^2) \cdot (1 - (1-p)^3)$
$((z_{i_1} \oplus z_{i_2}) \wedge \bar{r})$ $\vee (\bar{z}_{i_1} \wedge \bar{z}_{i_2})$	$(n_1, \dots, n_k)$	$2 \cdot p_s \cdot p \cdot (1-p)^4$ $+ p_s \cdot (1-p)^2$

Table 2: Intermediate probabilities for Lemma 6, starting from a partition $P = (n_1, \dots, n_k)$ and when the drawn indices i_1 and i_2 belong to two blocks of cardinals n_{j_1} and n_{j_2} , where $p_d := 2 \cdot \frac{n_{j_1} \cdot n_{j_2}}{n(n-1)}$.

Leakage	New partition $P' \in \mathcal{T}_{j_1, j_2}(P)$	Probability $\mathbb{P}(P \rightarrow P')$
$z_{i_1} \wedge z_{i_2} \wedge r$	$(1, 1, n_1, \dots, n_{j_1-1}, n_{j_1} - 1, n_{j_1+1}, \dots, n_{j_2-1}, n_{j_2} - 1, n_{j_2+1}, \dots, n_k)$	$p_d \cdot p^2 \cdot (1 - (1-p)^3)$
$z_{i_1} \wedge z_{i_2} \wedge \bar{r}$	$(2, n_1, \dots, n_{j_1-1}, n_{j_1} - 1, n_{j_1+1}, \dots, n_{j_2-1}, n_{j_2} - 1, n_{j_2+1}, \dots, n_k)$	$p_d \cdot p^2 \cdot (1-p)^3$
$z_{i_1} \wedge \bar{z}_{i_2} \wedge r$	$(1, n_1, \dots, n_{j_1-1}, n_{j_1} - 1, n_{j_1+1}, \dots, n_k)$	$p_d \cdot p \cdot (1-p) \cdot (1 - (1-p)^3)$
$\bar{z}_{i_1} \wedge z_{i_2} \wedge r$	$(1, n_1, \dots, n_{j_2-1}, n_{j_2} - 1, n_{j_2+1}, \dots, n_k)$	$p_d \cdot p \cdot (1-p) \cdot (1 - (1-p)^3)$
$z_{i_1} \wedge \bar{z}_{i_2} \wedge \bar{r}$	$(n_1, \dots, n_{j_1-1}, n_{j_1} - 1, n_{j_1+1}, \dots, n_{j_2-1}, n_{j_2} + 1, n_{j_2+1}, \dots, n_k)$	$p_d \cdot p \cdot (1-p)^4$
$\bar{z}_{i_1} \wedge z_{i_2} \wedge \bar{r}$	$(n_1, \dots, n_{j_1-1}, n_{j_1} + 1, n_{j_1+1}, \dots, n_{j_2-1}, n_{j_2} - 1, n_{j_2+1}, \dots, n_k)$	$p_d \cdot p \cdot (1-p)^4$
$\bar{z}_{i_1} \wedge \bar{z}_{i_2} \wedge r$	$(n_1, \dots, n_k)$	$p_d \cdot (1-p)^2 \cdot (1 - (1-p)^3)$
$\bar{z}_{i_1} \wedge \bar{z}_{i_2} \wedge \bar{r}$	$(n_1, \dots, n_{j_1-1}, n_{j_1+1}, \dots, n_{j_2-1}, n_{j_1} + n_{j_2}, n_{j_2+1}, \dots, n_k)$	$p_d \cdot (1-p)^5$

In order to simplify the security analysis of RPRefresh, we now introduce an artificial extension of RPZeroEnc, called RPZeroEnc₊, in which all the output shares are individually manipulated after γ iterations. In terms of leakage, this means that all the output shares of RPZeroEnc can now leak with probability p , in addition to the internal leakage already accounted for in Lemma 6.

Lemma 7 (RPZeroEnc₊ is RPP). *Let n, γ in $\mathbb{N}$ and $p \in [0, 1]$. The n -share zero-encoding gadget RPZeroEnc₊ ^{n, γ} is $(p, \mathcal{E}^\gamma)$ -RPP for some distribution envelope $\mathcal{E}^\gamma$ defined over $\mathcal{P}(n)$ constructively as follows. The distribution envelope $\mathcal{E}^\gamma$ is initialized to 0 for all the possible partitions. Then, for all k , for all partition $P = (n_1, \dots, n_k)$ in $\mathcal{P}(n)$, for all $(d_1, \dots, d_k)$ such that $d_1 \leq n_1, \dots, d_k \leq n_k$,*

$$\mathcal{E}^\gamma(P') = \mathcal{E}^\gamma(P) + \mathcal{E}_{\text{RPZeroEnc}}^\gamma(P) \cdot \prod_{i=1}^k \binom{n_i}{d_i} \cdot p^{d_i} \cdot (1-p)^{n_i-d_i}$$

with $\mathcal{E}_{\text{RPZeroEnc}}^\gamma$ the distribution envelope for the RPP property of RPZeroEnc given in Lemma 6 and $P' = (1, \dots, 1, n_1 - d_1, \dots, n_k - d_k)$.

Proof. Let us consider a partition $P = (n_1, \dots, n_k)$ for some $k \in [1, n]$. We recall that the elements n_i for $i \in [1, k]$ represent the cardinals of shares of $\llbracket z \rrbracket$ which form a perfect zero-encoding given the probed wires.

We begin by computing the intermediate probability p_{int} that for each part n_i of P , exactly $d_i (\leq n_i)$ shares of $\llbracket z \rrbracket$ leak. Since each share of $\llbracket z \rrbracket$ leaks independently with probability p , we apply the binomial distribution to obtain:

$$p_{\text{int}} = \prod_{i=1}^k \binom{n_i}{d_i} \cdot p^{d_i} \cdot (1-p)^{n_i-d_i}.$$

As a result, from partition P , we derive a new partition $P' = (1, \dots, 1, n_1 - d_1, \dots, n_k - d_k)$ with probability p_{int} .

Since the probability of obtaining P at the output of RPZeroEnc^γ is independent of p_{int} , the joint probability is simply the product of these two probabilities.

Finally, the probability of obtaining P' at the output of $\text{RPZeroEnc}_+^{n,\gamma}$ is the sum of the probabilities to have a partition P at the output of RPZeroEnc^γ ($\mathcal{E}_{\text{RPZeroEnc}}^\gamma(P)$) and a final leakage that transform P into P' . $\square$

We now introduce the major statement of this section.

Theorem 1 (RPRefresh is Cardinal-RPC). *Let $n, \gamma \in \mathbb{N}$, $p \in [0, 1]$. The n -share gadget RPRefresh^γ is $(p, (\mathcal{E}_t)_{t \in [0, n]})$ -cardinal-RPC with $(\mathcal{E}_t)_{t \in [0, n]}$ defined over $[0, n]$ as follows. For all $t_{\text{in}}, t_{\text{out}} \in [0, n]$,*

$$\mathcal{E}_{t_{\text{out}}}(t_{\text{in}}) = \sum_{i=0}^{t_{\text{in}}} q(i, t_{\text{out}}) \cdot \binom{n-i}{t_{\text{in}}-i} \cdot p^{t_{\text{in}}-i} \cdot (1-p)^{n-t_{\text{in}}}$$

where $q(i, t_{\text{out}})$ is defined constructively as follows:

1. For all $i, t_{\text{out}} \in [0, n]$, initialize $q(i, t_{\text{out}}) \leftarrow 0$.
2. For every $P = (n_1, \dots, n_k) \in \mathcal{P}(n)$, for every $d_1 \in [0, n_1]$, $\dots$, $d_k \in [0, n_k]$, update

$$q(i, t_{\text{out}}) \leftarrow q(i, t_{\text{out}}) + \mathcal{E}_{\text{RPZeroEnc}_+}^\gamma(P) \cdot \frac{\binom{n_1}{d_1} \cdots \binom{n_k}{d_k}}{\binom{n}{t_{\text{out}}}}$$

for $t_{\text{out}} = d_1 + \dots + d_k$, $i = n_1 \cdot \mathbb{1}_{d_1=n_1} + \dots + n_k \cdot \mathbb{1}_{d_k=n_k}$ and $\mathcal{E}_{\text{RPZeroEnc}_+}^\gamma$ taken from Lemma 7.

Intuitively, $q(i, t_{\text{out}})$ upper bounds the probability that i input shares are required to simulate the leakage on the internal variables of RPRefresh (without considering the input shares) along with the given t_{out} output shares.

Proof. From Lemmas 6 and 7, we have the probabilities associated to each partition of zero encoding cardinals at the output of $\text{RPZeroEnc}_+^\gamma$. In the cardinal-RPC property, the attacker is been given t_{out} output shares. When the indices of some of these output shares recover all the indices of a part of the partition at the output of $\text{RPZeroEnc}_+^\gamma$, then the attacker obtains the corresponding input shares. Conversely, if the indices of the output shares given to the attacker do not entirely recover the indices of a part of the partition at the output of $\text{RPZeroEnc}_+^\gamma$, the corresponding input shares are perfectly masked and hence not required for the simulation. We can compute the related probability with the distribution of a multihypergeometric law. Namely, for a partition $P = (n_1, \dots, n_k)$ for some $k \in [1, n]$, given t_{out} output shares, the probability that i input shares are required to simulate the leakage on the internal variables of RPRefresh (without considering the leakage on the input shares) is given by

$$q(i, t_{\text{out}}) = \sum_{\substack{P=(n_1, \dots, n_k) \\ P \in \mathcal{P}(n)}} \mathcal{E}_{\text{RPZeroEnc}_+}^\gamma(P) \sum_{\substack{(d_1, \dots, d_k) \\ d_1 + \dots + d_k = t_{\text{out}} \\ n_1 \cdot \mathbb{1}_{d_1=n_1} + \dots + n_k \cdot \mathbb{1}_{d_k=n_k} = i}} \frac{\binom{n_1}{d_1} \cdots \binom{n_k}{d_k}}{\binom{n}{t_{\text{out}}}}$$

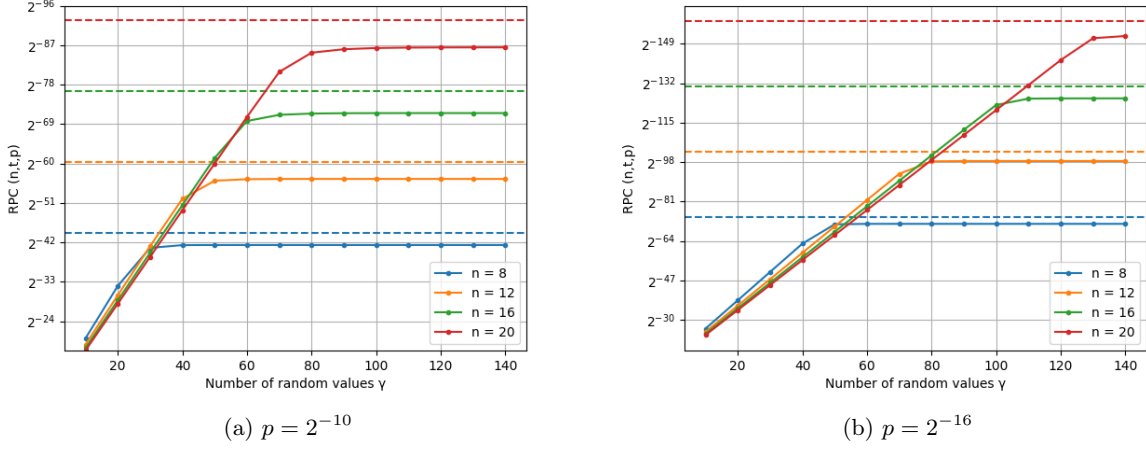


Fig.3: Computed Threshold-RPC-AI security advantage ε of RPRrefresh with fixed $(p, n, \frac{n}{2})$ for $n \in \{8, 12, 16, 20\}$ and $p \in \{p = 2^{-10}, p = 2^{-16}\}$ as a function of the number of iterations γ . The dotted lines corresponds to the maximum achievable security advantage.

in which each d_i represent the number of given output shares whose indices belong to the i^{th} encoding. When $i \leq t_{in}$, this probability is to be multiplied by the probability to recover the remaining input shares according to their leakage. Namely, the probability to recover j input shares among $n - i$ is given by

$$\binom{n-i}{j} \cdot p^j \cdot (1-p)^{n-i-j},$$

which concludes the proof. $\square$

Complexity of the envelope computation. The above results provide explicit formulas to compute the envelopes characterizing the cardinal-RPC security of our refresh. Following Lemma 6, evaluating those formulas has a complexity $\mathcal{O}(\gamma \cdot |\mathcal{P}(n)| \cdot n^2)$ by iterating over the γ iterations of the refresh, the $|\mathcal{P}(n)|$ partitions of n , and the different pairs of elements within a partition (whose number is lower than n^2). According to the Hardy-Ramanujan asymptotic formula, the number of integer partitions of n can be approximated as

$$|\mathcal{P}(n)| \approx \frac{1}{4n\sqrt{3}} \exp\left(\pi\sqrt{\frac{2n}{3}}\right)$$

which is subexponential in n . For instance, $|\mathcal{P}(8)| = 22$, $|\mathcal{P}(16)| = 231$, $|\mathcal{P}(32)| = 8349$, $|\mathcal{P}(64)| < 2^{21}$. In practice, using our (non-optimized) Python implementation of this computation results in, *e.g.*, less than 40 milliseconds for $(n, \gamma) = (8, 50)$, around 9 seconds for $(n, \gamma) = (16, 100)$, around 3 minutes for $(n, \gamma) = (20, 120)$ on a regular laptop.

Figure 3 illustrates the threshold-RPC security advantage of RPRrefresh for various numbers of shares n , with $t = \frac{n}{2}$, and for two leakage probabilities, $p = 2^{-10}$ (Figure 3a) and $p = 2^{-16}$ (Figure 3b), as a function of the number of random values γ . Those results have been obtained and validated by two independent implementations (from two different developers) of the envelope evaluation. We observe that, after a sufficient number of random values, the security advantage stabilizes, reaching a steady state. This steady state is logically lower bounded by the probability of observing $t + 1$ input shares from a simple random probing leakage of these shares (i.e., each share leaks with probability p), which is given by $\binom{n}{t+1} \cdot p^{t+1} \cdot (1-p)^{n-t-1}$. In the figure, this probability is represented by dotted lines in colors corresponding to n .

4.2 Simple Sharewise Cardinal-RPC Gadgets

With RPreFresh now defined, we proceed to construct a set of foundational gadgets that will serve as building blocks for many common schemes. The proposed constructions follow a very natural blueprint for defining a sharewise (linear multiplication, addition, or copy) gadget from a refresh gadget, called Refresh, such as already proposed in [BRT21, BRTV21]. Our main contribution here is to prove the cardinal-RPC security of these gadgets from that of the underlying refresh.

Linear Multiplication Gadget. The linear multiplication gadget was defined in [BRTV21] with a linear multiplication preceding the refreshing. For consistency with the other simple gadgets, we define it with a refreshing of the shares preceding the linear multiplication with the constant: $G_{\text{cmult}}(\llbracket x \rrbracket, c) = \text{Refresh}(\llbracket x \rrbracket) \cdot c$.

Algorithm 3 $G_{\text{cmult}}(\llbracket x \rrbracket, c)$ adapted from [BRTV21]

Input: $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n)$, constant $c \in \mathbb{K}$

Output: $\llbracket z \rrbracket = (z_1, z_2, \dots, z_n)$ such that $z_1 + z_2 + \dots + z_n = c \cdot (x_1 + x_2 + \dots + x_n)$

1. $\llbracket z \rrbracket \leftarrow \text{Refresh}(\llbracket x \rrbracket)$
 2. **for** $i = 1$ to n **do**
 3. $z_i \leftarrow z_i \cdot c$
 4. **end for**
 5. **return** $\llbracket z \rrbracket$
-

Lemma 8 (G_{cmult} is Cardinal-RPC). *Let $n \in \mathbb{N}$. Let $\text{Refresh}: \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t \in [0, n]})$ -cardinal-RPC for some collection of envelopes $(\mathcal{E}_t^R)_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, G_{cmult} is $(p, (\mathcal{E}_t^{\text{cmult}})_{t \in [0, n]})$ -cardinal-RPC where, for all $t_{\text{out}} \in [0, n]$, $\mathcal{E}_{t_{\text{out}}}^{\text{cmult}}$ is defined, for all $t_{\text{in}} \in [0, n]$, as:*

$$\mathcal{E}_{t_{\text{out}}}^{\text{cmult}}(t_{\text{in}}) = \sum_{i=0}^{n-t_{\text{out}}} \binom{n-t_{\text{out}}}{i} \cdot p^i \cdot (1-p)^{n-t_{\text{out}}-i} \cdot \mathcal{E}_{t_{\text{out}}+i}^R(t_{\text{in}}).$$

Proof. The only additional wires that can potentially leak in G_{cmult} compared to Refresh are the output shares of Refresh. The probability that t_{in} input shares are required to simulate the leakage and the final output shares is therefore obtained by computing the product of the leakage probability of i such wires with the probability of Refresh, considering an output composed of the global output plus these extra leaking wires. The probability that i output shares of Refresh leak, out of the $n - t_{\text{out}}^{\text{cmult}}$ output shares that remain unknown to the attacker, is given by

$$\binom{n-t_{\text{out}}}{i} \cdot p^i \cdot (1-p)^{n-t_{\text{out}}-i},$$

which completes the proof. □

Addition Gadget. G_{add} is the simple addition gadget from [BRT21]. It performs $G_{\text{add}}(\llbracket x \rrbracket, \llbracket y \rrbracket) = \text{Refresh}(\llbracket x \rrbracket) + \text{Refresh}(\llbracket y \rrbracket)$.

Algorithm 4 $G_{\text{add}}(\llbracket x \rrbracket, \llbracket y \rrbracket)$ from [BRT21]

Input: $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n)$ and $\llbracket y \rrbracket = (y_1, y_2, \dots, y_n)$

Output: $\llbracket z \rrbracket = (z_1, z_2, \dots, z_n)$ such that $z_1 + z_2 + \dots + z_n = (x_1 + y_1) + (x_2 + y_2) + \dots + (x_n + y_n)$

1. $\llbracket x' \rrbracket \leftarrow \text{Refresh}(\llbracket x \rrbracket)$
 2. $\llbracket y' \rrbracket \leftarrow \text{Refresh}(\llbracket y \rrbracket)$
 3. **for** $i = 1$ to n **do** $z_i \leftarrow x'_i + y'_i$
 4. **return** $\llbracket z \rrbracket$
-

Lemma 9 (G_{add} is Cardinal-RPC). Let $n \in \mathbb{N}$. Let $\text{Refresh}: \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t \in [0, n]})$ -cardinal-RPC for some collection envelopes $(\mathcal{E}_t^R)_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, G_{add} is $(p, (\mathcal{E}_t^{\text{add}})_{t \in [0, n]})$ -cardinal-RPC where, for all $t_{\text{out}} \in [0, n]$, $\mathcal{E}_{t_{\text{out}}}^{\text{add}}$ is defined, for all $\mathbf{t}_{\text{in}} = (t_{\text{in},1}, t_{\text{in},2}) \in [0, n]^2$, as

$$\begin{aligned} \mathcal{E}_{t_{\text{out}}}^{\text{add}}(\mathbf{t}_{\text{in}}) &= \sum_{i_1=0}^{n-t_{\text{out}}} \binom{n-t_{\text{out}}}{i_1} \cdot p^{i_1} \cdot (1-p)^{n-t_{\text{out}}-i_1} \cdot \mathcal{E}_{t_{\text{out}}+i_1}^R(t_{\text{in},1}) \\ &\quad \cdot \sum_{i_2=0}^{n-t_{\text{out}}} \binom{n-t_{\text{out}}}{i_2} \cdot p^{i_2} \cdot (1-p)^{n-t_{\text{out}}-i_2} \cdot \mathcal{E}_{t_{\text{out}}+i_2}^R(t_{\text{in},2}). \end{aligned}$$

Proof. We assume (this is the worst case) that all the t_{out} output shares given to the attacker require the corresponding $2 \cdot t_{\text{out}}$ output shares of the two instances of Refresh to be simulated. As for G_{cmult} , the output shares to simulate for each instance of Refresh are completed by the possible leakage at the input of the addition. In that case, the probabilities for both instances of Refresh to require $t_{\text{in},\alpha}$ input shares (for $\alpha \in \{1, 2\}$) are independent given the total number of output shares to simulate with the leakage and we obtain the above envelope. $\square$

In general, the addition gadget G_{add} can involve two distinct refresh gadgets, each with a different envelopes collection. Lemma 9 can be easily updated to account for this by getting use of the first envelopes collection in the first sum and the second envelopes collection in the second sum.

Copy Gadget. G_{copy} is the simple copy gadget from [BRT21]. It computes $G_{\text{copy}}(\llbracket x \rrbracket) = (\text{Refresh}(\llbracket x \rrbracket), \text{Refresh}(\llbracket x \rrbracket))$.

Algorithm 5 $G_{\text{copy}}(\llbracket x \rrbracket)$ from [BRT21]

Input: $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n)$

Output: $\llbracket z^1 \rrbracket = (z_1^1, z_2^1, \dots, z_n^1)$ and $\llbracket z^2 \rrbracket = (z_1^2, z_2^2, \dots, z_n^2)$ such that $z_1^1 + z_2^1 + \dots + z_n^1 = z_1^2 + z_2^2 + \dots + z_n^2 = x_1 + x_2 + \dots + x_n$

1. $\llbracket z^1 \rrbracket \leftarrow \text{Refresh}(\llbracket x \rrbracket)$
 2. $\llbracket z^2 \rrbracket \leftarrow \text{Refresh}(\llbracket x \rrbracket)$
 3. **return** $(\llbracket z^1 \rrbracket, \llbracket z^2 \rrbracket)$
-

Lemma 10 (G_{copy} is Cardinal-RPC). Let $n \in \mathbb{N}$. Let $\text{Refresh}: \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t \in [0, n]})$ -cardinal-RPC for some collection of envelopes $(\mathcal{E}_t^R)_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, G_{copy} is $(p, (\mathcal{E}_t^{\text{copy}})_{t \in [0, n]^2})$ -cardinal-RPC where, for all $t_{\text{out}} \in [0, n]^2$, $\mathcal{E}_{t_{\text{out}}}^{\text{copy}}$ is defined, for all $t_{\text{in}} \in [0, n]$, as:

$$\mathcal{E}_{t_{\text{out}}}^{\text{copy}}(t_{\text{in}}) = \sum_{0 \leq i \leq n} \mathcal{E}_{t_{\text{out},1}}^R(i) \cdot \mathcal{E}_{t_{\text{out},2}}^R(t_{\text{in}} - i).$$

Proof. From the property of Refresh , when t_{out}^R output shares are given to the attacker, the probability that t_{in}^R input shares are required for the simulation of both these output shares and the leakage is upper bounded by

$$\mathcal{E}_{t_{\text{out}}}^R(t_{\text{in}}^R).$$

In the context of G_{copy} , we have two instances of Refresh . The probability that t_{in} input shares are required for the simulation of the leakage and the output shares for G_{copy} is thus computed from the probability that one instance of Refresh requires i input shares for its own simulation and the second one $t_{\text{in}} - i$, hence the final result. $\square$

Algorithm 6 $G_{\text{decode}}(\llbracket x \rrbracket)$

Input: $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n)$ **Output:** $x = x_1 + x_2 + \dots + x_n$

1. $\llbracket t \rrbracket \leftarrow \text{Refresh}(\llbracket x \rrbracket)$
 2. $x \leftarrow \text{ShareSum}(\llbracket t \rrbracket)$
 3. **return** x
-

Algorithm 7 $\text{ShareSum}(\llbracket x \rrbracket)$

Input: $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n)$ **Output:** $x = x_1 + x_2 + \dots + x_n$

1. **if** $n = 2$ **then return** $x_1 + x_2$
 2. **if** $n = 3$ **then return** $\text{ShareSum}((x_1, x_2)) + x_3$
 3. $y \leftarrow \text{ShareSum}(x_1, x_2, \dots, x_{\lfloor \frac{n}{2} \rfloor})$
 4. $z \leftarrow \text{ShareSum}(x_{\lfloor \frac{n}{2} \rfloor + 1}, x_{\lfloor \frac{n}{2} \rfloor + 2}, \dots, x_n)$
 5. **return** $x = y + z$
-

The cardinal-RPC advantage of G_{copy} is tighter when the gadget is instantiated with uniformly cardinal-RPC refresh gadgets, as shown in Corollary 1⁵. Similar to the addition gadget, G_{copy} can also be instantiated with two distinct refresh gadgets, and the security results can be easily adapted accordingly.

Corollary 1. *Let $n \in \mathbb{N}$. Let $\text{Refresh}: \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share $(p, (\mathcal{E}_t^R)_{t \in [0, n]})$ -uniformly cardinal-RPC refresh gadget for some collection of envelopes $(\mathcal{E}_t^R)_{t \in [0, n]}$ and some probability $p \in [0, 1]$ (See Definition 16). Then, G_{copy} is $(p, (\mathcal{E}_t^{\text{copy}})_{t \in [0, n]^2})$ -cardinal-RPC where, for all $t_{\text{out}} \in [0, n]^2$, $\mathcal{E}_{t_{\text{out}}}^{\text{copy}}$ is defined as:*

$$\forall t_{\text{in}} \in [0, n], \quad \mathcal{E}_{t_{\text{out}}}^{\text{copy}}(t_{\text{in}}) = \sum_{\substack{0 \leq i, j \leq n \\ i, j \leq t_{\text{in}} \leq i+j}} \mathcal{E}_{t_{\text{out},1}}^R(i) \cdot \mathcal{E}_{t_{\text{out},2}}^R(j) \cdot \frac{\binom{i}{j-(t_{\text{in}}-i)} \cdot \binom{n-i}{t_{\text{in}}-i}}{\binom{n}{j}}.$$

Proof. We use the property of uniformly cardinal-RPC gadgets for which the sets of input shares required for the simulation are equiprobable for equal cardinals. Specifically, we can compute the probability that two sets of input shares of sizes i and j , required to simulate the leakage of both refresh gadgets and the outputs, represent t_{in} input shares. The number of common elements between both sets follows an hypergeometric law. Namely, the probability to have $i + j - t_{\text{in}}$ common elements is given by

$$\frac{\binom{i}{j-(t_{\text{in}}-i)} \cdot \binom{n-i}{t_{\text{in}}-i}}{\binom{n}{j}},$$

hence the final result. $\square$

4.3 Output Decoding Gadget

An output decoding gadget reconstructs values from their shares and outputs them in plain form. It is involved for variables that should be unmasked before being returned such as the public key in the key generation algorithm or the signature in the signing algorithm. In the probing model, it typically consists of a refresh gadget followed by a straightforward addition of all the shares. To our knowledge, it has not yet been analyzed within the random probing model. In this work, we propose combining a refresh gadget with a recursive addition of the shares (yielding a binary tree of additions), as presented in Algorithm 6.

Since the output of this gadget is intended to be revealed publicly, its security requirements differ from those of the previous gadgets. Specifically, in the context of composition, we are interested in the *cardinal RPC with public output* (Cardinal-RPC-O). This notion is a special case of the Cardinal-RPC-AI-O notion of Definition 7.

Theorem 2 (ShareSum is Cardinal-RPC-O). *Let $n \in \mathbb{N}$. ShareSum from Algorithm 7 is $(p, \mathcal{E})$ -cardinal-RPC-O where $\mathcal{E}$ is defined from the distribution $\mathcal{D}^{(n)}$ over $[0, n]^2$ as:*

⁵ Note that the refresh gadget instantiated in Section 4.1 does not achieve this uniform property. Hence, the tightness of Corollary 1 cannot be obtained in Section 6.

1. Sample $(r, s) \leftarrow \mathcal{D}^{(n)}$
2. Return s

The distribution $\mathcal{D}^{(n)}$ is recursively defined as follows. For every $n > 1$:

1. Sample $(r_1, s_1) \leftarrow \mathcal{D}^{(\lfloor n/2 \rfloor)}$
2. Sample $(r_2, s_2) \leftarrow \mathcal{D}^{(\lceil n/2 \rceil)}$
3. Return

$$(r, s) = \begin{cases} (n, \min(r_1 + r_2, s_1 + \lceil n/2 \rceil, s_2 + \lfloor n/2 \rfloor)) & \text{with proba. } p \\ (r_1 + r_2, \min(r_1 + r_2, s_1 + \lceil n/2 \rceil, s_2 + \lfloor n/2 \rfloor)) & \text{with proba. } 1 - p \end{cases}$$

And for $n = 1$:

$$\begin{cases} \text{Return } (1, 1) & \text{with proba. } p \\ \text{Return } (0, 0) & \text{with proba. } 1 - p \end{cases}$$

Proof. Consider the circuit defined by **ShareSum** which is represented as a binary tree, where the n leaves are the input shares of the circuits and the root the output. Each node leaks with probability p and we want to identify the number of input shares which are necessary for a perfect simulation of the leakage, given that the output x is provided to the simulator. By definition of **ShareSum**, we have the following invariant: an addition node in the tree with n' incoming leaves has two incoming sub-trees, the left one having $\lfloor n'/2 \rfloor$ leaves and the right one having $\lceil n'/2 \rceil$ leaves. The simulation strategy is illustrated on Figure 4.

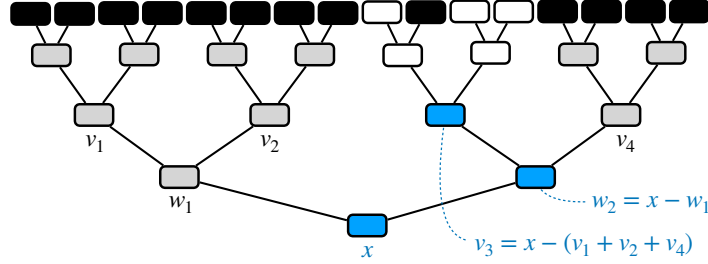


Fig. 4: Illustration of the simulation strategy for the **ShareSum** gadget. White nodes do not leak and hence do not require simulation. Black leaves either leak or are required for the simulation gray nodes. All the gray nodes can be simulated from the leaves (whether they leak or not). All the blue nodes are “simulated from the root”, meaning they are simulated from the output x plus some of the gray nodes. In this example, one needs $n - 3$ input shares for the simulation.

We say that a node can be “simulated from the root” whenever its parent node can be simulated from the root and its sibling node is the root of a sub-tree for which all the leaves are input of the simulation. In the example of Figure 4, w_2 and v_3 satisfy this definition. This make them simulatable from the root x and some gray nodes which are themselves simulatable from the given leaves. In practice, the pattern represented in Figure 4 can be used for the simulation with any blue path going from the root to one intermediate node (v_3 in the example). The sibling sub-trees of this path (gray nodes in the example) are all fully simulated from their leaves which are required to the simulation. The remaining sub-tree with highest blue node as root is simulated from the required subset of its leaves, depending on the leaking nodes.

To identify the minimal set of input shares required for the simulation one can define a recursive walk from the root to the leaves as follows. Let us denote $T = (R, T_1, T_2)$ any sub-tree with root R , left sub-tree T_1 and right sub-tree T_2 . If $T = (R, \emptyset, \emptyset)$, then R is a leaf. We denote $\text{leak}(R)$ the event that the node R leaks (which occurs with probability p) and $\text{leaves}(T)$ the set of leaves of a sub-tree T . We further denote b ,

a Boolean value indicating whether the root of the current sub-tree T can be simulated from the root (which depends on what happened before in the walk). We obtain the following recursive function:

NeededLeaves(T, b):

1. $(R, T_1, T_2) = T$
2. If $T_1 = T_2 = \emptyset$ (R is a leaf),
 - (a) If $\text{leak}(R)$ (R leaks), return $\{R\}$.
 - (b) If $\neg \text{leak}(R)$ (R does not leak), return $\emptyset$.
3. Let $\mathcal{L}_1 = \text{NeededLeaves}(T_1, 1) \cup \text{leaves}(T_2)$ (make T_2 a gray sub-tree, only possible if $b = 1$).
4. Let $\mathcal{L}_2 = \text{NeededLeaves}(T_2, 1) \cup \text{leaves}(T_1)$ (make T_1 a gray sub-tree, only possible if $b = 1$).
5. Let $\mathcal{L}_3 = \text{NeededLeaves}(T_1, 0) \cup \text{NeededLeaves}(T_2, 0)$ (identify leaves to simulate T_1 and T_2 sub-trees, only possible if R does not leak).
6. If $b = 0$ (R cannot be simulated from the root):
 - (a) If $\text{leak}(R)$, return $\text{leaves}(T)$ (all the leaves are necessary to simulate R).
 - (b) If $\neg \text{leak}(R)$, return $\mathcal{L}_3$ (leaves to simulate T_1 and T_2).
7. If $b = 1$ (R can be simulated from the root): return either $\mathcal{L}_1$, $\mathcal{L}_2$ or $\mathcal{L}_3$, the set with the smallest cardinality.

By calling **NeededLeaves**($T, 1$) on the full tree T , we obtain the smallest set of leaves that are necessary to simulate all the leaking nodes of the tree according to the simulation strategy exemplified in Figure 4. Consider T as the random variable representing a sub-tree where each node leaks with probability p . Define $\mathcal{R} = \text{NeededLeaves}(T, 0)$ and $\mathcal{S} = \text{NeededLeaves}(T, 1)$. We now argue that the pair $(|\mathcal{R}|, |\mathcal{S}|)$ follows the distribution $\mathcal{D}^{(n_T)}$ defined in Theorem 2, where n_T is the number of leaves of the sub-tree T . The base case is trivial: $(|\mathcal{R}|, |\mathcal{S}|) = (1, 1)$ with probability p (leaf leakage) and $(0, 0)$ otherwise. The general case follows from the description of **NeededLeaves**, observing that in the definition of $\mathcal{D}^{(n)}$, we have $(r_1, s_1) = (|\mathcal{R}_1|, |\mathcal{S}_1|)$ and $(r_2, s_2) = (|\mathcal{R}_2|, |\mathcal{S}_2|)$ where:

$$(\mathcal{R}_i, \mathcal{S}_i) \sim (\text{NeededLeaves}(T_i, 0), \text{NeededLeaves}(T_i, 1)) \quad \forall i \in \{1, 2\}.$$

□

From Lemma 4, G_{decode} is therefore naturally Cardinal-RPC-O. The distribution $\mathcal{D}^{(n)}$ in the above theorem can be efficiently computed through recursion, providing a precise evaluation of the Cardinal-RPC-O security for our decoding gadget.

5 Random Probing Secure Noise Generation

In this section, we provide a random-probing composable version of the noise generation gadget introduced in Raccoon [dEK⁺23]. In such a context, the base ring of the considered arithmetic circuit is $\mathbb{K} = \mathbb{Z}_q$ for some integer q . The principle of the noise generation gadget is to sample uniform random values over a subset of $\mathbb{Z}_q$ and to add them to the input sharing, thereby introducing the noise required in the underlying Module Learning with Errors (MLWE) instance. In our context, these random values are captured as auxiliary inputs and the noise generation gadget simply aims at summing them to the sharings while minimizing the leakage on these values. We evaluate the security and complexity of two constructions –either in chain or in tree structure– when this sum is instantiated with our new addition gadget introduced in Section 4.2.

5.1 Noise Generation Gadget

Algorithm 8 generically describes the gadget responsible for noise addition. Essentially, it relies on a parsing procedure **DivideAI**, depicted in Algorithm 9, which splits the auxiliary inputs into blocks of n elements from $\mathbb{K}$ and appends zeros to the last block, if necessary. Subsequently, a macro gadget, G_{sum} , is invoked to operate on these blocks of size n . This can be instantiated either with $G_{\text{sum-chain}}$ from Algorithm 10 or with $G_{\text{sum-tree}}$ from Algorithm 11. The input sharing is finally added to the output of G_{sum} to produce the final result.

As stated in Lemma 11, **AddNoiseTo** is immediately cardinal-RPC if both algorithms G_{sum} and G_{add} are cardinal-RPC, as indicated by the composition result in Lemma 4.

Algorithm 8 AddNoiseTo($\llbracket x \rrbracket, \mathbf{a}$)

Input: $\llbracket x \rrbracket = (x_1, x_2, \dots, x_n)$, auxiliary inputs $\mathbf{a} \in \mathbb{K}^\alpha$
Output: $\llbracket y \rrbracket = (y_1, y_2, \dots, y_n)$ such that $y = x + a_1 + a_2 + \dots + a_\alpha$

1. $(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{\lceil \frac{\alpha}{n} \rceil}) \leftarrow \text{DivideAl}(\mathbf{a}, n)$
2. $\mathbf{z} \leftarrow G_{\text{sum}}(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{\lceil \frac{\alpha}{n} \rceil})$
3. **return** $G_{\text{add}}(\llbracket x \rrbracket, \mathbf{z})$

Algorithm 9 DivideAl($\mathbf{a}, n$)

Input: Auxiliary inputs $\mathbf{a} \in \mathbb{K}^\alpha, n$
Output: $\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{\lceil \frac{\alpha}{n} \rceil} \in \mathbb{K}^n$

1. **for** $i = 1$ to $\lceil \frac{\alpha}{n} \rceil - 1$ **do**
2. $\mathbf{b}_i \leftarrow (a_{(i-1) \cdot n + 1}, \dots, a_{i \cdot n})$
3. **end for**
4. $\mathbf{b}_{\lceil \frac{\alpha}{n} \rceil} \leftarrow (a_{(\lceil \frac{\alpha}{n} \rceil - 1) \cdot n + 1}, \dots, a_\alpha, 0, \dots, 0)$
5. **return** $(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_{\lceil \frac{\alpha}{n} \rceil})$

Algorithm 10 $G_{\text{sum-chain}}(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_\ell)$

Input: $(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_\ell) \in (\mathbb{K}^n)^\ell$
Output: $\mathbf{z} \in \mathbb{K}^n$

1. $\mathbf{t} \leftarrow \mathbf{b}_1$
2. **for** $i = 2$ to ℓ **do**
3. $\mathbf{t} \leftarrow G_{\text{add}}(\mathbf{t}, \mathbf{b}_i)$
4. **end for**
5. **return** $\mathbf{t}$

Algorithm 11 $G_{\text{sum-tree}}(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_\ell)$

Input: $(\mathbf{b}_1, \mathbf{b}_2, \dots, \mathbf{b}_\ell) \in (\mathbb{K}^n)^\ell$
Output: $\mathbf{z} \in \mathbb{K}^n$

1. **if** $\ell = 1$ **return** $\mathbf{b}_1$
2. **return** $G_{\text{add}}(G_{\text{sum-tree}}(\mathbf{b}_1, \dots, \mathbf{b}_{\lfloor \ell/2 \rfloor}), G_{\text{sum-tree}}(\mathbf{b}_{\lfloor \ell/2 \rfloor + 1}, \mathbf{b}_2, \dots, \mathbf{b}_\ell))$

Lemma 11 (AddNoiseTo is Cardinal-RPC). Let $n, \alpha \in \mathbb{N}$ and $p \in [0, 1]$. Let $G_{\text{sum}}: (\mathbb{K}^n)^{\lceil \frac{\alpha}{n} \rceil - 1} \rightarrow \mathbb{K}^n$ be $(p, (\mathcal{E}_t^{G_{\text{sum}}})_{t \in [0, n]})$ -cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{G_{\text{sum}}})_{t \in [0, n]}$ defined over $[0, n]^{\lceil \frac{\alpha}{n} \rceil - 1}$ and $G_{\text{add}}: (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ be $(p, (\mathcal{E}_t^{G_{\text{add}}})_{t \in [0, n]})$ -cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{G_{\text{add}}})_{t \in [0, n]}$ defined over $[0, n]^2$. Then, AddNoiseTo, defined in Algorithm 8, is $(p, (\mathcal{E}_t^{ADN})_{t \in [0, n]})$ -cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{ADN})_{t \in [0, n]}$ defined over $[0, n] \times [0, \alpha]$ such that for all $t \in [0, n]$, for all $\mathbf{t}_{\text{in}} = (t_{\text{in}}^x, t_{\text{in},1}^a, \dots, t_{\text{in}, \lceil \frac{\alpha}{n} \rceil}^a) \in [0, n]^{\lceil \frac{\alpha}{n} \rceil + 1}$,

$$\mathcal{E}_t^{ADN}(t_{\text{in}}^x, t_{\text{in},1}^a + \dots + t_{\text{in}, \lceil \frac{\alpha}{n} \rceil}^a) = \sum_{i=0}^n \mathcal{E}_i^{G_{\text{sum}}}(t_{\text{in},1}^a, \dots, t_{\text{in}, \alpha}^a) \cdot \mathcal{E}_t^{G_{\text{add}}}(t_{\text{in}}^x, i).$$

In the following, we explore two structures for generating noise: a chain structure, illustrated in Figure 5a, and a binary tree structure, shown in Figure 5b. The respective algorithms for these structures are provided in Algorithms 10 and 11. Lemmas 12 and 13 establish their cardinal-RPC properties.

Lemma 12 is proven by recursively applying the composition result from Lemma 4. Similarly, Lemma 13 is established by applying the same composition result recursively on each branch of the tree, starting from the output nodes.

Lemma 12 ($G_{\text{sum-chain}}$ is Cardinal-RPC). Let $n, \ell \in \mathbb{N}$ and $p \in [0, 1]$. Let $G_{\text{add}}: (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ be an n -share $(p, (\mathcal{E}_t^{G_{\text{add}}})_{t \in [0, n]})$ -cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{G_{\text{add}}})_{t \in [0, n]}$ defined over $[0, n]^2$. Gadget $G_{\text{sum-chain}}: (\mathbb{K}^n)^\ell \rightarrow \mathbb{K}^n$, instantiated with G_{add} , is $(p, (\mathcal{E}_t^{SNC})_{t \in [0, n]})$ -cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{SNC})_{t \in [0, n]}$ defined over $[0, n]^\ell$ as follows:

$$\forall \mathbf{t}_{\text{in}} = (t_{\text{in}}^1, t_{\text{in}}^2, \dots, t_{\text{in}}^\ell) \in [0, n]^\ell,$$
$$\mathcal{E}_t^{SNC}(\mathbf{t}_{\text{in}}) = \sum_{i_3=0}^n \mathcal{E}_{i_3}^{G_{\text{add}}}(t_{\text{in}}^1, t_{\text{in}}^2) \cdot \dots \cdot \sum_{i_\ell=0}^n \mathcal{E}_{i_\ell}^{G_{\text{add}}}(i_{\ell-1}, t_{\text{in}}^{\ell-1}) \cdot \mathcal{E}_t^{G_{\text{add}}}(i_\ell, t_{\text{in}}^\ell).$$

Lemma 13 ($G_{\text{sum-tree}}$ is Cardinal-RPC). Let $n, \ell \in \mathbb{N}$ and $p \in [0, 1]$. Let $G_{\text{add}}: (\mathbb{K}^n)^2 \rightarrow \mathbb{K}^n$ be an n -share $(p, (\mathcal{E}_t^{G_{\text{add}}})_{t \in [0, n]})$ -cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{G_{\text{add}}})_{t \in [0, n]}$ defined over $[0, n]^2$. Gadget

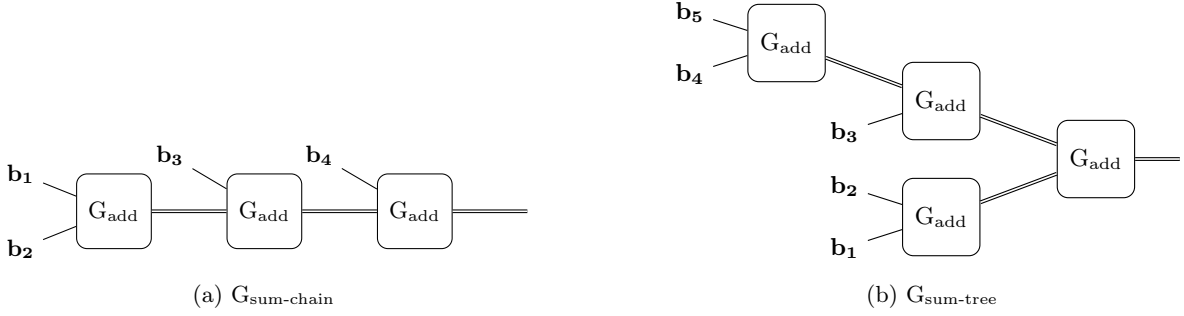


Fig. 5: Noise generation gadget.

$G_{\text{sum-tree}}: (\mathbb{K}^n)^\ell \rightarrow \mathbb{K}^n$, instantiated with G_{add} , is $(p, (\mathcal{E}_t^{SPT(\ell)})_{t \in [0, n]})$ -cardinal-RPC for some envelopes collection $(\mathcal{E}_t^{SPT(\ell)})_{t \in [0, n]}$ recursively defined over $[0, n]^\ell$ as follows. For all $\mathbf{t}_{in} = (t_{in}^1, t_{in}^2, \dots, t_{in}^\ell) \in [0, n]^\ell$,

$$\begin{aligned} \text{if } \ell = 3, \quad \mathcal{E}_t^{SPT(3)}(t_{in}^1, t_{in}^2, t_{in}^3) &= \sum_{i=0}^n \mathcal{E}_i^{G_{\text{add}}}(t_{in,1}, t_{in,2}) \cdot \mathcal{E}_t^{G_{\text{add}}}(i, t_{in,3}) \\ \text{otherwise, } \mathcal{E}_t^{SPT(\ell)}(\mathbf{t}_{in}) &= \sum_{i_1=0}^n \sum_{i_2=0}^n \mathcal{E}_{i_1}^{SPT(\lfloor \ell/2 \rfloor)}(t_{in}^1, \dots, t_{in}^{\lfloor \ell/2 \rfloor}) \cdot \mathcal{E}_{i_2}^{SPT(\lceil \ell/2 \rceil)}(t_{in}^{\lfloor \ell/2 \rfloor + 1}, \dots, t_{in}^\ell) \cdot \mathcal{E}_t^{G_{\text{add}}}(i_1, i_2) \end{aligned}$$

with the sub-tree $SPT(\alpha)$ being $(p, (\mathcal{E}_t^{SPT(\alpha)})_{t \in [0, n]})$ -cardinal-RPC for $3 \leq \alpha \leq n$ for some envelope $(\mathcal{E}_t^{SPT(\alpha)})_{t \in [0, n]}$ and with $SPT(2) = G_{\text{add}}$.

5.2 Instantiation from New Gadgets

In a first attempt, we can use the expansion strategy from [BCP⁺20] to construct a noise generation gadget based on a small addition gadget, as detailed in Appendix B.2. In this section, we illustrate how to leverage our new addition gadget G_{add} to achieve the same objective. Table 3 displays the threshold-RPC advantage of G_{add} from its cardinal-RPC envelopes (see Lemma 9) while fixing the leakage probability, the number of shares, and the value of t . The complexity is assessed in terms of the number of additions and random values required⁶. As anticipated, the complexities for the same leakage probability and security threshold are more favorable than those obtained through the expansion strategy.

Building on these findings, we can evaluate the RPC security of AddRepNoise for specific probabilities and sizes α of the auxiliary input. With cardinal-RPC gadgets, the composition is tighter, and the requirement for auxiliary inputs is considered globally. For example, with $\alpha = 75$ auxiliary inputs and a leakage probability $p = 2^{-24}$, one option is to combine three G_{add} gadgets, each with 25 shares, to set up AddNoiseTo. If we fix the number of iterations γ to 25 for each refresh, the resulting RPC advantage is upper bounded by 2^{-182} both for the chain and the tree structure, when 10 shares from the main input and 30 auxiliary inputs are required for the simulation. As for the complexity, the noise gadget requires 150 random values and implements 525 additions. This is to be compared with the 1200 randoms, 2475 additions and 1200 copies required by the AddNoiseTo gadget obtained from the expansion strategy (and which achieves an RPC advantage of 2^{-169}).

⁶ We omit the count of random values used for selecting indices in the refresh gadget, as they differ in nature from the random produced by the random gates of the circuit. Indeed, these random indices are from $[1, n]$ and not from $\mathbb{K}$. For the considered application (Raccoon), or other lattice-based schemes, elements from $[1, n]$ are expected to be (significantly) smaller than elements from $\mathbb{K}$. Moreover, the random index values in our refresh can be fully revealed to the adversary without compromising the security (which relies on their uniformity but not on their secrecy) unlike the random values of $\mathbb{K}$ produced by random gates which leak with probability p .

Table 3: RPC security advantage and complexity of G_{add} .

Leakage probability p	2^{-12}	2^{-16}	2^{-20}	2^{-24}
# Shares n and # leaking shares t	(25,13)	(20,10)	(16,8)	(12,6)
# Randoms	240	220	220	200
# Additions	555	500	488	436
RPC advantage	2^{-128}	2^{-128}	2^{-134}	2^{-129}

Remark 1. Our experiments show that the tree structure generally provides a slightly better security advantage compared to the chain structure, while maintaining the same single-thread complexity. We believe this is due to the higher average number of dependencies per input in the tree structure, and their balance. Furthermore, both structures offer the potential to explore different instantiations for the refresh gadgets – *i.e.* varying the number of iterations – based on their position, a direction we leave for future research.

6 Random Probing Secure Implementation of Raccoon

We recall the key generation and signature algorithms of Raccoon in Algorithms 12 and 14 from [dEK⁺23]. We refer to [dEK⁺23] for an explicit definition of the subroutines and their rationale. The parameters relevant to our work are presented in Table 4. Specifically, parameters n_R (denoted n in [dEK⁺23]) and q define the quotient ring $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^{n_R} + 1)$, while k and ℓ define the MLWE matrix dimensions. Like in the proof of Raccoon in the d -probing model [dKPR24], we do not consider that the noise samples (r_i) and (r'_i) are generated within `AddRepNoise` algorithm; instead, they are provided as auxiliary inputs. We also artificially de-correlate the number of shares, here denoted n , from the parameter d , fixed in the specifications.

6.1 Random Probing Secure Construction

We denote by G^{vec} the natural extension of a gadget G , originally operating on elements of $\mathbb{Z}_q$, to a gadget operating on elements of $\mathcal{R}_q$ and which applies the same operation component-wise, repeating it n_R times. We also introduce the following core gadgets:

- G_{cpmult} : This gadget multiplies a uniform public polynomial c_{poly}^i in $\mathcal{R}_q$ with a shared secret vector $\llbracket s^i \rrbracket$ in $(\mathcal{R}_q^n)$, for $i \in [1, \ell]$. It requires n_R^2 n -share linear multiplications on elements of $\mathbb{Z}_q$, $n_R(n_R - 1)$ n -share additions on elements of $\mathbb{Z}_q$ and $n_R(n_R - 1)$ n -share copies on elements of $\mathbb{Z}_q$.
- G_{cmmult} : This gadget performs a multiplication between a uniform public matrix A in $\mathcal{R}_q^{k \times \ell}$ with a shared secret vector $\llbracket s \rrbracket$ in $(\mathcal{R}_q^\ell)^n$. It requires $\ell k n_R^2$ n -share linear multiplications on elements of $\mathbb{Z}_q$, $n_R(n_R - 1)\ell k + n_R(\ell - 1)k$ n -share additions on elements of $\mathbb{Z}_q$ and $n_R(n_R - 1)\ell k + n_R(k - 1)\ell$ n -share copies on elements of $\mathbb{Z}_q$.

To obtain a random-probing secure construction, masked gadgets secure in the probing model can be seamlessly replaced by random-probing composable gadgets. The random-probing Raccoon equivalent is illustrated in Algorithms 13 and 15. Let us provide an intuition for the random probing version of the key generation. The original algorithm (Alg. 12) starts with a fresh zero-encoding (line 3) and adds the auxiliary inputs $(r_i)_{1 \leq i \leq \ell}$ with intermediate refreshes (inside `AddRepNoise` in line 4). In our design, the refresh gadgets are already included in the elementary gadgets, so we simply sum the auxiliary inputs $(r_i)_{1 \leq i \leq \ell}$ split in convenient packs using `DivideAI` and the G_{sum} gadget (see Section 5). Next, the resulting sharing $\llbracket s \rrbracket$ is copied: one copy is used for the public key and the other one for the secret key. After the multiplication step, we use `AddNoiseTo` (presented in Alg. 8) to incorporate the auxiliary inputs $(r'_i)_{1 \leq i \leq k}$ and finally decode and output the result. The same reasoning can be similarly applied to the signature algorithm. In a nutshell, compared to the original d -probing secure algorithms, most intermediate refresh gadgets can be removed and copy gadgets are added whenever a variable needs to be reused. We stress that *our random-probing Raccoon is functionally equivalent to the original d -probing version*.

Algorithm 12 d -Probing-KeyGen

Auxiliary Inputs: $(r_i)_{1 \leq i \leq \ell} \in (\{0, 1\}^{u_t})^{d \cdot \text{rep} \cdot \ell \cdot n_R}$
 $(r'_i)_{1 \leq i \leq k} \in (\{0, 1\}^{u_t})^{d \cdot \text{rep} \cdot k \cdot n_R}$ **Output:** Key pair (vk, sk)

1. $\text{seed} \leftarrow \{0, 1\}^\kappa$
 2. $A \leftarrow \text{ExpandA}(\text{seed})$
 3. $\llbracket s \rrbracket \leftarrow \ell \times \text{ZeroEncoding}(d)$
 4. $\llbracket s \rrbracket \leftarrow \text{AddRepNoise}(\llbracket s \rrbracket, (r_i)_{1 \leq i \leq \ell})$
 5. $\llbracket t \rrbracket \leftarrow A \cdot \llbracket s \rrbracket$
 6. $\llbracket t \rrbracket \leftarrow \text{AddRepNoise}(\llbracket t \rrbracket, (r'_i)_{1 \leq i \leq k})$
 7. $t \leftarrow \text{Decode}(\llbracket t \rrbracket)$
 8. **return** $(\text{vk} \leftarrow (\text{seed}, t), \text{sk} \leftarrow (t, \llbracket s \rrbracket))$
-

Algorithm 13 RP-KeyGen

Auxiliary Inputs: $(r_i)_{1 \leq i \leq \ell} \in (\{0, 1\}^{u_t})^{d \cdot \text{rep} \cdot \ell \cdot n_R}$
 $(r'_i)_{1 \leq i \leq k} \in (\{0, 1\}^{u_t})^{d \cdot \text{rep} \cdot k \cdot n_R}$ **Output:** Key pair (vk, sk)

1. $\text{seed} \leftarrow \{0, 1\}^\kappa$
 2. $A \leftarrow \text{ExpandA}(\text{seed})$
 3. $\mathbf{r}_b \leftarrow \text{DivideAI}^{\text{vec}}((r_i)_{1 \leq i \leq \ell})$
 4. $\llbracket s \rrbracket \leftarrow G_{\text{sum}}^{\text{vec}}(\mathbf{r}_b)$
 5. $(\llbracket s \rrbracket, \llbracket s_t \rrbracket) \leftarrow G_{\text{copy}}^{\text{vec}}(\llbracket s \rrbracket)$
 6. $\llbracket t \rrbracket \leftarrow G_{\text{cmmult}}(A, \llbracket s_t \rrbracket)$
 7. $\llbracket t \rrbracket \leftarrow \text{AddNoiseTo}^{\text{vec}}(\llbracket t \rrbracket, (r'_i)_{1 \leq i \leq k})$
 8. $t \leftarrow G_{\text{decode}}^{\text{vec}}(\llbracket t \rrbracket)$
 9. **return** $(\text{vk} \leftarrow (\text{seed}, t), \text{sk} \leftarrow (t, \llbracket s \rrbracket))$
-

Algorithm 14 d -Probing-Sign

Auxiliary Inputs: $(r_i)_{1 \leq i \leq \ell} \in (\{0, 1\}^{u_t})^{d \cdot \text{rep} \cdot \ell \cdot n_R}$
 $(r'_i)_{1 \leq i \leq k} \in (\{0, 1\}^{u_t})^{d \cdot \text{rep} \cdot k \cdot n_R}$ **Input:** $\text{sk} = (\text{vk}, \llbracket s \rrbracket)$, $\text{msg} \in \{0, 1\}^*$ **Output:** $\text{sig} = (c_{\text{hash}}, h, z)$

1. $(\text{vk}, \llbracket s \rrbracket) \leftarrow \text{sk}, (\text{seed}, t) \leftarrow \text{vk}$
 2. $\mu \leftarrow H(H(\text{vk}) \parallel \text{msg})$
 3. $A \leftarrow \text{ExpandA}(\text{seed})$
 4. $\llbracket r \rrbracket \leftarrow \ell \times \text{ZeroEncoding}(d)$
 5. $\llbracket r \rrbracket \leftarrow \text{AddRepNoise}(\llbracket r \rrbracket, (r_i)_{1 \leq i \leq \ell})$
 6. $\llbracket w \rrbracket \leftarrow A \cdot \llbracket r \rrbracket$
 7. $\llbracket w \rrbracket \leftarrow \text{AddRepNoise}(\llbracket w \rrbracket, (r'_i)_{1 \leq i \leq k})$
 8. $w \leftarrow \text{Decode}(\llbracket w \rrbracket)$
 9. $c_{\text{hash}} \leftarrow \text{ChalHash}(w, \mu)$
 10. $c_{\text{poly}} \leftarrow \text{ChalPoly}(c_{\text{hash}})$
 11. $\llbracket s \rrbracket \leftarrow \text{Refresh}(\llbracket s \rrbracket)$
 12. $\llbracket r \rrbracket \leftarrow \text{Refresh}(\llbracket r \rrbracket)$
 13. $\llbracket z \rrbracket \leftarrow c_{\text{poly}} \cdot \llbracket s \rrbracket + \llbracket r \rrbracket$
 14. $\llbracket z \rrbracket \leftarrow \text{Refresh}(\llbracket z \rrbracket)$
 15. $z \leftarrow \text{Decode}(\llbracket z \rrbracket)$
 16. $h \leftarrow w - A \cdot z - c_{\text{hash}} \cdot t$
 17. $\text{sig} \leftarrow (c_{\text{hash}}, h, z)$
 18. **if** $\text{CheckBounds}(\text{sig}) = \text{FAIL}$ **then**
 19. **goto** Line 4
 20. **end if**
 21. **return** sig
-

Algorithm 15 RP-Sign

Auxiliary Inputs: $(r_i)_{1 \leq i \leq \ell} \in (\{0, 1\}^{u_t})^{d \cdot \text{rep} \cdot \ell \cdot n_R}$
 $(r'_i)_{1 \leq i \leq k} \in (\{0, 1\}^{u_t})^{d \cdot \text{rep} \cdot k \cdot n_R}$ **Input:** $\text{sk} = (\text{vk}, \llbracket s \rrbracket)$, $\text{msg} \in \{0, 1\}^*$ **Output:** $\text{sig} = (c_{\text{hash}}, h, z)$

1. $(\text{vk}, \llbracket s \rrbracket) \leftarrow \text{sk}, (\text{seed}, t) \leftarrow \text{vk}$
 2. $\mu \leftarrow H(H(\text{vk}) \parallel \text{msg})$
 3. $A \leftarrow \text{ExpandA}(\text{seed})$
 4. $\mathbf{r}_b \leftarrow \text{DivideAI}^{\text{vec}}((r_i)_{1 \leq i \leq \ell})$
 5. $\llbracket r \rrbracket \leftarrow G_{\text{sum}}^{\text{vec}}(\mathbf{r}_b)$
 6. $(\llbracket r \rrbracket, \llbracket r_c \rrbracket) \leftarrow G_{\text{copy}}^{\text{vec}}(\llbracket r \rrbracket)$
 7. $\llbracket w \rrbracket \leftarrow G_{\text{cmmult}}(A, \llbracket r \rrbracket)$
 8. $\llbracket w \rrbracket \leftarrow \text{AddNoiseTo}^{\text{vec}}(\llbracket w \rrbracket, (r'_i)_{1 \leq i \leq k})$
 9. $w \leftarrow G_{\text{decode}}^{\text{vec}}(\llbracket w \rrbracket)$
 10. $c_{\text{hash}} \leftarrow \text{ChalHash}(w, \mu)$
 11. $c_{\text{poly}} \leftarrow \text{ChalPoly}(c_{\text{hash}})$
 12. $\llbracket z \rrbracket \leftarrow G_{\text{add}}^{\text{vec}}(G_{\text{cpmult}}(\llbracket s \rrbracket, c_{\text{poly}}), \llbracket r_c \rrbracket)$
 13. $z \leftarrow G_{\text{decode}}^{\text{vec}}(\llbracket z \rrbracket)$
 14. $h \leftarrow w - A \cdot z - c_{\text{hash}} \cdot t$
 15. $\text{sig} \leftarrow (c_{\text{hash}}, h, z)$
 16. **if** $\text{CheckBounds}(\text{sig}) = \text{FAIL}$ **then**
 17. **goto** Line 4
 18. **end if**
 19. **return** sig
-

Table 4: Some of Raccoon’s parameters (see [dEK⁺23] for the full description).

Param.	Raccoon-128-16
q	549824583172097
n_R	512
k	5
ℓ	4
d	16
rep	2

Table 5: Parameters introduced for the random probing security.

Param.	Description
p	Probability of leaking a wire (resulting from the noise level of the device).
ϵ	Threshold-RPC-AI-O advantage, indicates the security level (high for weak security, low for strong security).
n	Number of shares ($= d$ or $\geq d$ if expansion, often higher to reach better security levels).
t	Threshold RPC parameter (set to $n/2$).
γ	Number of iterations of the refresh algorithm, allowing optimizations of the security/performance trade-off.

Instantiation with our new gadgets. We now instantiate Algorithms 13 and 15 with our new cardinal-RPC(-AI-O) secure gadgets defined in Section 4.2. Figures 6 and 7 (in Appendix) visually illustrate the processes for Raccoon-128. We denote by G_{copies}^k the combination of copy gadgets arranged in a binary tree structure to produce k copies of the same sharing, and by G_{cpmult}^k the application of the G_{cpmult} gadget to perform a polynomial multiplication on $\mathcal{R}_q$ for k separate inputs.

With these new gadgets, our schemes can be instantiated in various ways, with multiple parameters subject to variation – namely, the probability p , the RPC threshold t , and the number of iterations γ for each refresh instance. All these new parameters are summarized in Table 5. We opt for aligning to the Raccoon-128-16 configuration, which operates with 16-sharings. The corresponding instantiation is detailed in Table 4, while the full set of parameters can be found in [dEK⁺23, Table 2]. The scheme is proven EUF-CMA secure even when up to $d - 1 = 15$ values of r_i and r'_i per polynomial coefficient are exposed to the attacker. In other words, the existing security model assumes that at most 15 out of 32 values of r_i and r'_i can be given to the attacker. Hence, at least 17 small uniforms per polynomial coefficient must remain secret. Note that this applies to each of the $n_R \times \ell$ (or $n_R \times k$ for r'_i) polynomial coefficients. Intuitively, the security then corresponds to the security of Raccoon with $\text{rep} = 1$ instead of 2, denoted “Small Raccoon” in [dKPR24]. We refer to [dKPR24, Security of Small Raccoon in Sec. 7] for more information.

The Raccoon-128-16 configuration is particularly well-suited because using 16-sharings requires two inputs per coefficient to incorporate the small uniforms. This results in a single addition gadget per coefficient in the first G_{sum} and two addition gadgets per coefficient in the **AddNoiseTo** gadget.

Table 6: Threshold-RPC-AI-O advantages and complexities of Raccoon’s key generation and signature when instantiated with our new gadgets. The values for γ (chosen to be different for block 1, block 2 and G_{cmult}) are fixed to minimize ϵ (which impacts the overall randomness consumption).

Key Generation					Signature				
p # additions # linear mult. # randoms	2^{-24}	2^{-20}	2^{-16}	2^{-12}	2^{-24}	2^{-20}	2^{-16}	2^{-12}	
	$\epsilon \leq 2^{-128}$				$\epsilon \leq 2^{-128}$				
	$2^{31.5}$	$2^{31.7}$	-	-	$2^{32.1}$	$2^{32.3}$	-	-	
	$2^{26.3}$	$2^{26.3}$	-	-	$2^{26.6}$	$2^{26.6}$	-	-	
# additions # linear mult. # randoms	$2^{30.3}$	$2^{30.5}$	-	-	$2^{30.8}$	$2^{31.1}$	-	-	
	$\epsilon \leq 2^{-80}$				$\epsilon \leq 2^{-80}$				
	$2^{30.7}$	$2^{30.9}$	$2^{31.1}$	-	$2^{31.3}$	$2^{31.4}$	$2^{31.6}$	-	
	$2^{26.3}$	$2^{26.3}$	$2^{26.3}$	-	$2^{26.6}$	$2^{26.6}$	$2^{26.6}$	-	
# additions # linear mult. # randoms	$2^{29.2}$	$2^{29.5}$	$2^{29.8}$	-	$2^{30.3}$	$2^{30.5}$	$2^{30.7}$	-	
	$\epsilon \leq 2^{-64}$				$\epsilon \leq 2^{-64}$				
	$2^{30.4}$	$2^{30.5}$	$2^{30.8}$	$2^{31.0}$	$2^{31.0}$	$2^{31.1}$	$2^{31.3}$	$2^{31.5}$	
	$2^{26.3}$	$2^{26.3}$	$2^{26.3}$	$2^{26.3}$	$2^{26.6}$	$2^{26.6}$	$2^{26.6}$	$2^{26.6}$	
# additions # linear mult. # randoms	$2^{28.8}$	$2^{29.0}$	$2^{29.3}$	$2^{29.6}$	$2^{29.5}$	$2^{29.7}$	$2^{30.0}$	$2^{30.2}$	

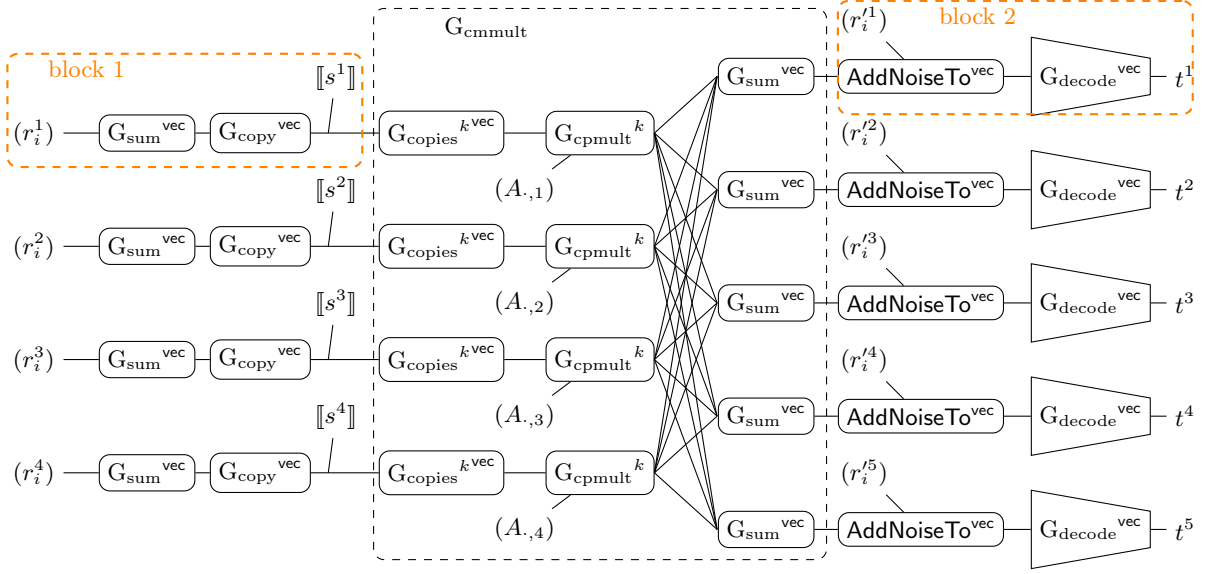


Fig. 6: RP-KeyGen for Raccoon-128 with $\ell = 4$ and $k = 5$

Theorems 3 establishes the threshold-RPC-AI-O security of Raccoon’s key generation and Raccoon’s signature processes as defined in Algorithms 13 and 15, following Definition 17. The proofs rely on the cardinal-RPC(-AI)(-O) envelopes of gadgets and their compositions, and is provided in Appendix C.

Theorem 3. *Let t' be the maximum number of leaking values of r_i and r'_i for each polynomial coefficient for both key generation and signature algorithm. The key generation and signature algorithms of Raccoon-128-16, displayed in Algorithms 13 and 15, are $(p, t = 8, t' = 15, \varepsilon)$ -threshold-RPC-AI-O⁷ with the pairs (p, ε) provided in Table 6.*

6.2 Comparison with the expansion technique and the probing secure implementation

Algorithms 13 and 15 can also be instantiated with the expanded gadgets from [BCP⁺20] (note that G_{decode} was not defined in [BCP⁺20]). Details on the expansion tools used to generate a random-probing secure Raccoon algorithm are provided in Appendix B.3. By applying new gadgets and leveraging composition proofs, we achieve significant improvements in both complexity and randomness consumption, as illustrated in Table 7.

Finally, to compare with the original probing-secure version of Raccoon, we calculate the probability of observing more than t variables in both algorithms, which gives a proven upper bound on its random probing security advantage. For $n = 16$ shares and a leakage probability of 2^{-24} , we obtain an advantage of almost 1 –meaning no proven security– for the key generation and for the signature. This highlights the benefit of our constructions to achieve random probing security compared to standard probing secure schemes.

7 Conclusion and future work

In conclusion, this work proposes new techniques towards stronger and tighter random probing security while providing all necessary components, from security definitions to concrete constructions, to show that random-probing security *is achievable* for a post-quantum scheme, namely Raccoon. We provide the first performance

⁷ Following Definition 17, we partition the auxiliary inputs into $n_r \times (\ell + k)$ vectors $\mathbf{a}_i$ containing 32 small uniform values. The threshold t' applies to each $\mathbf{a}_i$.

Table 7: Security and complexity of the key generation and the signature of Raccoon with 128 bits of (black-box) security in its original probing version and for both random probing secure scenarios with $p = 2^{-24}$.

	Key Generation			Signature		
	Original	Expansion	New Gadgets	Original	Expansion	New Gadgets
# shares	16	27	16	16	27	16
# additions	$2^{26.4}$	$2^{34.8}$	$2^{31.5}$	$2^{26.6}$	$2^{35.1}$	$2^{32.1}$
# linear mult.	$2^{26.3}$	$2^{27.1}$	$2^{26.3}$	$2^{26.6}$	$2^{27.4}$	$2^{26.6}$
# randoms	$2^{18.5}$	$2^{33.8}$	$2^{30.3}$	$2^{19.4}$	$2^{34.3}$	$2^{30.8}$
Security RPS/C	1	2^{-116}	2^{-132}	1	2^{-116}	2^{-130}

results for a random-probing secure post-quantum algorithm. We hope our design and notions will serve as building blocks for more efficient random-probing secure constructions and inspire “random-probing-friendly” designs.

In this work, we have focused on achieving random probing security for an existing version of Raccoon, namely Raccoon-128-16. It now remains to investigate variants with different parameters that would be better suited for random probing security. And, an additional relevant future work would be to refine the Small Raccoon security proof of [dKPR24] to directly combine black-box EUF-CMA security and random probing security and investigate various tradeoffs.

Another interesting avenue for future research could be to investigate if other recent techniques proposed in [DFZ19,BFO23,JMB24] could lead to different trade-offs for random probing secure versions of Raccoon. This would require extending the compilers proposed in these works to support composition with auxiliary input and/or public output, designing a secure decoding gadget within these frameworks, and comparing with our approach.

Acknowledgments. We warmly thank Katharina Boudgoust, Laurent Imbert, Loïc Masure, Laz Panard, Thomas Roche and Sara Sahraee for their meticulous and relevant feedback on this work. This work was co-funded by the European Union (ERC, AMaskZONE, 101077506) and Bpifrance (RESQUE project, France national quantum strategy). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

- ABC⁺23. Melissa Azouaoui, Olivier Bronchain, Gaëtan Cassiers, Clément Hoffmann, Yulia Kuzovkova, Joost Renes, Tobias Schneider, Markus Schönauer, François-Xavier Standaert, and Christine van Vredendaal. Protecting Dilithium against leakage revisited sensitivity analysis and improved implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):58–79, 2023.
- ADF16. Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EURO-CRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.
- AIS18. Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.
- Ajt11. Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 715–724, San Jose, CA, USA, June 6–8, 2011. ACM Press.

- BBE⁺18. Gilles Barthe, Sonia Belaïd, Thomas Espitau, Pierre-Alain Fouque, Benjamin Grégoire, Mélissa Rossi, and Mehdi Tibouchi. Masking the GLP lattice-based signature scheme at any order. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 354–384, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.
- BCP⁺20. Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 339–368, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.
- BCPZ16. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39, Santa Barbara, CA, USA, August 17–19, 2016. Springer Berlin Heidelberg, Germany.
- BCRT23. Sonia Belaïd, Gaëtan Cassiers, Matthieu Rivain, and Abdul Rahman Taleb. Unifying freedom and separation for tight probing-secure composition. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 440–472, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.
- BFO23. Francesco Berti, Sebastian Faust, and Maximilian Ortl. Provable secure parallel gadgets. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):420–459, 2023.
- BMRT22. Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. IronMask: Versatile verification of masking security. In *2022 IEEE Symposium on Security and Privacy*, pages 142–160, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.
- BRT21. Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 313–343, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.
- BRTV21. Sonia Belaïd, Matthieu Rivain, Abdul Rahman Taleb, and Damien Vergnaud. Dynamic random probing expansion with quasi linear asymptotic complexity. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 157–188, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.
- CFOS21. Gaëtan Cassiers, Sebastian Faust, Maximilian Ortl, and François-Xavier Standaert. Towards tight random probing security. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 185–214, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.
- CGTZ23. Jean-Sébastien Coron, François Gérard, Matthias Trannoy, and Rina Zeitoun. Improved gadgets for the high-order masking of Dilithium. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):110–145, 2023.
- CJRR99. Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO’99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer Berlin Heidelberg, Germany.
- DDF14. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.
- dEK⁺23. Rafael del Pino, Thomas Espitau, Shuichi Katsumata, Mary Maller, Fabrice Mouhartem, Thomas Prest, Mélissa Rossi, and Markku-Juhani Saarienen. Raccoon. Technical report, National Institute of Standards and Technology, 2023.
- DFZ19. Stefan Dziembowski, Sebastian Faust, and Karol Zebrowski. Simple refreshing in the noisy leakage model. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 315–344, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland.
- dKPR24. Rafaël del Pino, Shuichi Katsumata, Thomas Prest, and Mélissa Rossi. Raccoon: A masking-friendly signature proven in the probing model. In Leonid Reyzin and Douglas Stebila, editors, *Advances in*

- Cryptology – CRYPTO 2024, Part I*, volume 14920 of *Lecture Notes in Computer Science*, pages 409–444, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.
- EEN⁺24. Muhammed F. Esgin, Thomas Espitau, Guilhem Niot, Thomas Prest, Amin Sakzad, and Ron Steinfeld. sfPlover: Masking-friendly hash-and-sign lattice signatures. In Marc Joye and Gregor Leander, editors, *Advances in Cryptology – EUROCRYPT 2024, Part VII*, volume 14657 of *Lecture Notes in Computer Science*, pages 316–345, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.
- EFG⁺22. Thomas Espitau, Pierre-Alain Fouque, François Gérard, Mélissa Rossi, Akira Takahashi, Mehdi Tibouchi, Alexandre Wallet, and Yang Yu. Mitaka: A simpler, parallelizable, maskable variant of falcon. In Orr Dunkelman and Stefan Dziembowski, editors, *Advances in Cryptology – EUROCRYPT 2022, Part III*, volume 13277 of *Lecture Notes in Computer Science*, pages 222–253, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.
- GJR18. Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain. How to securely compute with noisy leakage in quasilinear complexity. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 547–574, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.
- GP99. Louis Goubin and Jacques Patarin. DES and differential power analysis (the “duplication” method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES’99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer Berlin Heidelberg, Germany.
- ISW03. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer Berlin Heidelberg, Germany.
- JMB24. Vahid Jahandideh, Bart Mennink, and Lejla Batina. An algebraic approach for evaluating random probing security with application to AES. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(4):657–689, 2024.
- LDK⁺22. Vadim Lyubashevsky, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Peter Schwabe, Gregor Seiler, Damien Stehlé, and Shi Bai. CRYSTALS-DILITHIUM. Technical report, National Institute of Standards and Technology, 2022.
- PFH⁺22. Thomas Prest, Pierre-Alain Fouque, Jeffrey Hoffstein, Paul Kirchner, Vadim Lyubashevsky, Thomas Pornin, Thomas Ricosset, Gregor Seiler, William Whyte, and Zhenfei Zhang. FALCON. Technical report, National Institute of Standards and Technology, 2022.
- Pre23. Thomas Prest. A key-recovery attack against mitaka in the t -probing model. In Alexandra Boldyreva and Vladimir Kolesnikov, editors, *PKC 2023: 26th International Conference on Theory and Practice of Public Key Cryptography, Part I*, volume 13940 of *Lecture Notes in Computer Science*, pages 205–220, Atlanta, GA, USA, May 7–10, 2023. Springer, Cham, Switzerland.
- RP10. Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427, Santa Barbara, CA, USA, August 17–20, 2010. Springer Berlin Heidelberg, Germany.
- SAB⁺22. Peter Schwabe, Roberto Avanzi, Joppe Bos, Léo Ducas, Eike Kiltz, Tancrede Lepoint, Vadim Lyubashevsky, John M. Schanck, Gregor Seiler, Damien Stehlé, and Jintai Ding. CRYSTALS-KYBER. Technical report, National Institute of Standards and Technology, 2022.

Experiment 5: Threshold-RPC

$\mathcal{W} \leftarrow \text{LeakingWires}(G, p)$ $I_1, \dots, I_\ell \leftarrow \text{Sim}_1(G, \mathcal{W}, J_1, \dots, J_m)$ $out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket_{I_1}, \dots, \llbracket x_\ell \rrbracket_{I_\ell})$ $\text{return } (I_1, \dots, I_\ell, out_1, out_2)$	<i>Drawing the leaking wires.</i> <i>The simulator is given the indexes of the leaking outputs. It chooses indexes for the inputs.</i> <i>The simulator is given the shares at desired indexes and returns simulated values for the leaking wires (out_1) and outputs (out_2)</i>

A Security definitions

Definition 12 (Circuit Compiler from [BCP⁺20]). A circuit compiler is a triplet of algorithms $(\text{CC}, \text{Enc}, \text{Dec})$ defined as follows:

- CC (circuit compilation) is a deterministic algorithm that takes as input an arithmetic circuit C and outputs a randomized arithmetic circuit $\hat{C}$,
- Enc (input encoding) is a probabilistic algorithm that maps an input $\mathbf{x} \in \mathbb{K}^\ell$ to an encoded input $\hat{\mathbf{x}} \in \mathbb{K}^{\ell'}$,
- Dec (output decoding) is a deterministic algorithm that maps an encoded output $\llbracket \mathbf{y} \rrbracket \in \mathbb{K}^{m'}$ to a plain output $\mathbf{y} \in \mathbb{K}^m$,

which satisfy the following properties:

- **Correctness:** For every arithmetic circuit C of input length ℓ , and for every $\mathbf{x} \in \mathbb{K}^\ell$, we have

$$\Pr(\text{Dec}(\hat{C}(\llbracket \mathbf{x} \rrbracket)) = C(\mathbf{x}) \mid \llbracket \mathbf{x} \rrbracket \leftarrow \text{Enc}(\mathbf{x})) = 1, \text{ where } \hat{C} = \text{CC}(C).$$

- **Efficiency:** For some security parameter $\kappa \in \mathbb{N}$, the running time of $\text{CC}(C)$ is $\text{poly}(\kappa, |C|)$, the running time of $\text{Enc}(\mathbf{x})$ is $\text{poly}(\kappa, |\mathbf{x}|)$ and the running time of $\text{Dec}(\llbracket \mathbf{y} \rrbracket)$ is $\text{poly}(\kappa, |\llbracket \mathbf{y} \rrbracket|)$, where $\text{poly}(\kappa, \ell) = O(\kappa^{e_1} \ell^{e_2})$ for some constants e_1, e_2 .

Definition 13 ((Threshold) Random Probing Composability from [BCP⁺20]). Let $n, \ell, m \in \mathbb{N}$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is (t, p, ϵ) -random probing composable (RPC) for some $t \in \mathbb{N}$ and $p, \epsilon \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every masked input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $J_1 \subseteq [n], \dots, J_m \subseteq [n]$ of cardinals $|J_1| \leq t, \dots, |J_m| \leq t$, the outputs of Experiment 5 are such that

1. $\Pr((|I_1| > t) \vee \dots \vee (|I_\ell| > t)) \leq \epsilon$
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, \llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{y} \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket_{J_1}, \dots, \llbracket y_m \rrbracket_{J_m})}_{\text{Output leakage}} \right)$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$ and $\llbracket \mathbf{y} \rrbracket \leftarrow G(\llbracket \mathbf{x} \rrbracket)$.

Definition 14. Let $f : \mathbb{R} \rightarrow \mathbb{R}$. The gadget G is (t, f) -RPC if it is $(t, p, f(p))$ -RPC for every $p \in [0, 1]$.

Theorem 4 (Composition from [BCP⁺20]). Let $t \in \mathbb{N}$, $p, \epsilon \in [0, 1]$, and CC be a standard circuit compiler with (t, p, ϵ) -RPC base gadgets. For every (randomized) arithmetic circuit C composed of $|C|$ gadgets, the compiled circuit $\text{CC}(C)$ is $(p, |C| \cdot \epsilon)$ -random probing secure. Equivalently, the standard circuit compiler CC is (p, ϵ) -random probing secure.

Experiment 6: General-RPC

$\mathcal{W} \leftarrow \text{LeakingWires}(G, p)$ $I_1, \dots, I_\ell \leftarrow \text{Sim}_1(G, \mathcal{W}, J_1, \dots, J_m)$ $out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket_{I_1}, \dots, \llbracket x_\ell \rrbracket_{I_\ell})$ $\text{return } (I_1, \dots, I_\ell, out_1, out_2)$	<i>Drawing the leaking wires.</i> <i>The simulator is given the indexes of the leaking outputs. It chooses indexes for the inputs.</i> <i>The simulator is given the shares at desired indexes and returns simulated values for the leaking wires (out_1) and outputs (out_2)</i>

Definition 15 (General Random Probing Composability from [CFOS21]). Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}$ represent a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathbb{K}^{|J_1|} \times \dots \times \mathbb{K}^{|J_m|}$, where $|J_i| \in [0, n]$ for every $i \in [1, m]$. These envelopes are defined over tuples of size ℓ of elements in $\mathbb{K}^i$ for $i \in [0, n]$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E})$ -general random probing composable (general-RPC for short) for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket x \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \dots, J_m)$, the outputs of Experiment 6 are such that

1. $(I_1, \dots, I_\ell) \lesssim \mathcal{E}_{J_1, \dots, J_m}$,
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, \llbracket x \rrbracket, \llbracket y \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket_{J_1}, \dots, \llbracket y_m \rrbracket_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(G, p)$ and $\llbracket y \rrbracket \leftarrow G(\llbracket x \rrbracket)$.

Definition 16 (Uniformly Cardinal Random Probing Composability). Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^u$ represent a collection of probability envelopes indexed by $(j_1, \dots, j_m) \subseteq [0, n]^m$ over $[0, n]^\ell$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is uniformly $(p, \mathcal{E}^u)$ -cardinal random probing composable (uniformly cardinal-RPC for short) for some $p \in [0, 1]$ if it is $(p, \mathcal{E}^g)$ -general-RPC and if for all $(J_1, \dots, J_m), (J'_1, \dots, J'_m) \in \mathbb{K}^{|J_1|} \times \dots \times \mathbb{K}^{|J_m|}$ such that

$$(|J_1|, \dots, |J_m|) = (|J'_1|, \dots, |J'_m|)$$

and for all $(I_1, \dots, I_\ell), (I'_1, \dots, I'_\ell) \in \mathbb{K}^{|I_1|} \times \dots \times \mathbb{K}^{|I_\ell|}$ such that

$$(|I_1|, \dots, |I_\ell|) = (|I'_1|, \dots, |I'_\ell|),$$

$$\text{we have } \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \mathcal{E}_{J'_1, \dots, J'_m}^g(I'_1, \dots, I'_\ell).$$

The collection of uniformly cardinal-RPC envelopes $\mathcal{E}^u$ is then naturally defined such that

$$\mathcal{E}_{|J_1|, \dots, |J_m|}^u(|I_1|, \dots, |I_\ell|) = \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell).$$

Definition 17 ((Threshold) Random Probing Composability with Auxiliary Inputs and Public Outputs).

Let $n, \ell, m, k, \alpha, d \in \mathbb{N}$. Let G be a gadget with the following input/output partition

$$G : \overbrace{(\mathbb{K}^n)^\ell}^{\text{masked inputs}} \times \overbrace{(\mathbb{K}^\alpha)^k}^{\text{auxiliary inputs}} \rightarrow \overbrace{(\mathbb{K}^n)^m}^{\text{masked outputs}} \times \overbrace{\mathbb{K}^d}^{\text{public outputs}}$$

$$(\llbracket x \rrbracket, \mathbf{a}_1, \dots, \mathbf{a}_k) \mapsto (\llbracket y \rrbracket, \mathbf{z}).$$

The gadget G is (p, t, t', ε) -threshold random probing composable with Auxiliary Inputs and Public Outputs for some $p \in [0, 1]$, $t, t' \in \mathbb{N}$ and $\varepsilon > 0$ if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such

Experiment 7: Threshold-RPC-AI-O

$(*, z) \leftarrow C(\text{Enc}(x), a)$	<i>Drawing the public outputs.</i>
$\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$	<i>Drawing the leaking wires.</i>
$(I_1, \dots, I_\ell), (L_1, \dots, L_k) \leftarrow \text{Sim}_1(C, \mathcal{W}, z, J_1, \dots, J_m)$	<i>The simulator is given the indexes of the leaking outputs. It chooses indexes for the shared and auxiliary inputs.</i>
$out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket _{I_1}, \dots, \llbracket x_\ell \rrbracket _{I_\ell}, a_1 _{L_1}, \dots, a_k _{L_k})$	<i>The simulator is given the shares at desired indexes and returns simulated values for the leaking wires and outputs.</i>
$\text{return } (out_1, out_2, (I_1, \dots, I_\ell), (L_1, \dots, L_k))$	

that for every shared input $\llbracket x \rrbracket \in (\mathbb{K}^n)^\ell$, auxiliary input $a_1, \dots, a_k \in (\mathbb{K}^\alpha)^k$, and for every set collection $(J_1, \dots, J_m)$ where $J_1 \subseteq [n], \dots, J_m \subseteq [n]$ of cardinals $|J_1| \leq t, \dots, |J_m| \leq t$, the outputs of Experiment 7 are such that

1. $\Pr(|I_1| > t) \vee \dots \vee (|I_\ell| > t) \leq \varepsilon$
2. $\Pr(|L_1| > t') \vee \dots \vee (|L_k| > t') \leq \varepsilon$
3. and

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, (\llbracket x \rrbracket, a), (\llbracket y \rrbracket, z))}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right)$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$ and $\llbracket y \rrbracket, z \leftarrow G(\llbracket x \rrbracket, a)$.

B Expansion Strategy

B.1 Random Probing Expandability

In [AIS18], an *expansion* approach was introduced to build a random-probing secure circuit compiler from a secure multi-party protocol. This approach was later revisited in [BCP⁺20] with the formalization of the notion of *expanding compiler* [BCP⁺20].

The principle of the expanding compiler is to recursively apply a base compiler, denoted CC and which simply consists in replacing each gate in the input circuit by the corresponding gadget. Assume we have n -share gadgets G_g for each gate g in a basis $\mathbb{B}$. The base compiler CC simply consists in replacing each gate g in these gadgets by G_g and by replacing each wire by n wires carrying a sharing of the value. We thus obtain n^2 -share gadgets by simply applying CC to each gadget: $G_g^{(2)} = \text{CC}(G_g)$. This process can be iterated an arbitrary number of times, say k , to an input circuit C :

$$C \xrightarrow{\text{CC}} \widehat{C}_1 \xrightarrow{\text{CC}} \dots \xrightarrow{\text{CC}} \widehat{C}_k.$$

The first output circuit $\widehat{C}_1$ is the original circuit in which each gate g is replaced by a base gadget G_g . The second output circuit $\widehat{C}_2$ is the original circuit C in which each gate is replaced by an n^2 -share gadget $G_g^{(2)}$. Equivalently, $\widehat{C}_2$ is the circuit $\widehat{C}_1$ in which each gate is replaced by a base gadget. In the end, the output

Experiment 8: RPE

$\mathcal{W} \leftarrow \text{LeakingWires}(G, p)$ $(I_1, I_2, J') \leftarrow \text{Sim}_1(G, \mathcal{W}, J)$ $out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket _{I_1}, \llbracket x_2 \rrbracket _{I_2})$ $\text{return } (I_1, I_2, J', out_1, out_2).$	<i>Drawing the leaking wires.</i> <i>The simulator is given the indexes of the leaking outputs (J). It chooses indexes for the inputs (I_1, I_2).</i> <i>The simulator is given the shares at desired indexes and returns simulated values for the leaking wires (out_1) and outputs (out_2).</i>

circuit $\widehat{C}_k$ is hence the original circuit C in which each gate has been replaced by a k -expanded gadget and each wire has been replaced by n^k wires carrying an (n^k) -linear sharing of the original wire.

The expanding compiler achieves random probing security if the base gadgets verify a property called *random probing expandability* [BCP⁺20]. We recall hereafter the original definition of the random probing expandability (RPE) property for 2-to-1 gadgets.

Definition 18 (Random Probing Expandability from [BCP⁺20]). *Let $f : \mathbb{R} \rightarrow \mathbb{R}$. An n -share 2-to-1 gadget $G : \mathbb{K}^n \times \mathbb{K}^n \rightarrow \mathbb{K}^n$ is (t, f) -random probing expandable ((t, f) -RPE for short) if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every admissible pair $((\llbracket x_1 \rrbracket, \llbracket x_2 \rrbracket), \llbracket y \rrbracket) \in (\mathbb{K}^n \times \mathbb{K}^n) \times \mathbb{K}^n$, for every set $J \subseteq [n]$ and for every $p \in [0, 1]$, the outputs of Experiment 8 are such that*

1. $\Pr(|I_1| > t) = \Pr(|I_2| > t) = f(p)$ and $\Pr((|I_1| > t) \wedge (|I_2| > t)) = f(p)^2$,
2. $J' \subseteq [n]$ is such that $J' = J$ if $|J| \leq t$ and $|J'| = n - 1$ otherwise,
3. and

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G, (\llbracket x_1 \rrbracket, \llbracket x_2 \rrbracket), \llbracket y \rrbracket))}_{\text{Induced Random probing leakage}}, \underbrace{\llbracket y \rrbracket|_{J'}}_{\text{Output leakage}} \right)$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$.

The first condition of Definition 18 defines two events, called "failure events" $\mathcal{F}_1 \equiv (|I_1| > t)$ and $\mathcal{F}_2 \equiv (|I_2| > t)$. In particular, condition 1 implies that $\mathcal{F}_1$ and $\mathcal{F}_2$ are mutually independent. The second condition may seem artificial as the simulator is able to modify the indexes of leaking outputs. This is actually necessary to capture some specific events (when $|J| \geq t$) in the proofs, we refer to [BCP⁺20] for more details.

Remark 2. The RPE notion can be simply extended to gadgets with 2 outputs: the Sim_1 simulator takes two sets $J_1 \subseteq [n]$ and $J_2 \subseteq [n]$ as input and produces two sets J'_1 and J'_2 satisfying the same property as J' in the above definition (w.r.t. J_1 and J_2). The Sim_2 simulator must then produce an output with the knowledge of $\llbracket y_1 \rrbracket|_{J'_1}$ and $\llbracket y_2 \rrbracket|_{J'_1}$ where $\llbracket y_1 \rrbracket$ and $\llbracket y_2 \rrbracket$ are the output sharings.

The RPE notion can also be simply extended to gadgets with a single input: the Sim_1 simulator produces a single set I so that the failure event $(|I| > t)$ occurs with probability ε (and the Sim_2 simulator is then simply given $\llbracket x \rrbracket|_I$ where $\llbracket x \rrbracket$ is the single input sharing). We refer the reader to [BCP⁺20] for the formal definitions of these variants.

B.2 Instantiation of Noise Generation from RPE

In this section, we rely on the expansion strategy from [BCP⁺20] (and recalled in Section B.1) to build a noise generation gadget from an addition gadget. Specifically, based on the analysis and instantiations

from [BRT21], we begin with the following small addition and copy gadgets of 3 and 5 shares, which we then expand to increase their original security level for various leakage probabilities:

$$\begin{aligned} G_{\text{add}}^3 : (\mathbb{K}^3)^2 &\rightarrow \mathbb{K}^3 \\ ((a_1, a_2, a_3), (b_1, b_2, b_3)) &\mapsto (c_1, c_2, c_3) \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow (r_1 + a_1) + (r_3 + b_1) \\ c_2 &\leftarrow (r_2 + a_2) + (r_4 + b_2) \\ c_3 &\leftarrow (r_1 + r_2 + a_3) + (r_3 + r_4 + b_3) \end{aligned}$$

$$\begin{aligned} G_{\text{copy}}^3 : \mathbb{K}^3 &\rightarrow (\mathbb{K}^3)^2 \\ (a_1, a_2, a_3) &\mapsto ((c_1, c_2, c_3), (d_1, d_2, d_3)) \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow r_1 + a_1 & d_1 &\leftarrow r_3 + a_1 \\ c_2 &\leftarrow r_2 + a_2 & d_2 &\leftarrow r_4 + a_2 \\ c_3 &\leftarrow r_1 + r_2 + a_3 & d_3 &\leftarrow r_3 + r_4 + a_3 \end{aligned}$$

and

$$\begin{aligned} G_{\text{add}}^5 : (\mathbb{K}^5)^2 &\rightarrow \mathbb{K}^5 \\ ((a_i)_{1 \leq i \leq 5}, (b_i)_{1 \leq i \leq 5}) &\mapsto (c_i)_{1 \leq i \leq 5} \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow (r_1 + r_2 + a_1) + (r_6 + r_7 + b_1) \\ c_2 &\leftarrow (r_2 + r_3 + a_2) + (r_7 + r_8 + b_2) \\ c_3 &\leftarrow (r_3 + r_4 + a_3) + (r_8 + r_9 + b_3) \\ c_4 &\leftarrow (r_4 + r_5 + a_4) + (r_9 + r_{10} + b_4) \\ c_5 &\leftarrow (r_5 + r_1 + a_5) + (r_{10} + r_6 + b_5) \end{aligned}$$

$$\begin{aligned} G_{\text{copy}}^5 : (\mathbb{K}^5)^2 &\rightarrow \mathbb{K}^5 \\ (a_i)_{1 \leq i \leq 5} &\mapsto ((c_i)_{1 \leq i \leq 5}, (d_i)_{1 \leq i \leq 5}) \end{aligned}$$

$$\begin{aligned} c_1 &\leftarrow r_1 + r_2 + a_1 & d_1 &\leftarrow r_6 + r_7 + a_1 \\ c_2 &\leftarrow r_2 + r_3 + a_2 & d_2 &\leftarrow r_7 + r_8 + a_2 \\ c_3 &\leftarrow r_3 + r_4 + a_3 & d_3 &\leftarrow r_8 + r_9 + a_3 \\ c_4 &\leftarrow r_4 + r_5 + a_4 & d_4 &\leftarrow r_9 + r_{10} + a_4 \\ c_5 &\leftarrow r_5 + r_1 + a_5 & d_5 &\leftarrow r_{10} + r_6 + a_5 \end{aligned}$$

Using **IronMask** from [BMRT22], we obtain the RPE advantage of both gadgets for $t = 1$ with 3 shares and $t = 2$ with 5 shares and for any probability p . For 3 shares, we have:

$$f_3(p) \approx 171p^2 + 36.4p^2\sqrt{p} + 72p^3 + 364.9p^3\sqrt{p} + 12648p^4 + \mathcal{O}(p^5)$$

with a maximum leakage probability (*i.e.* the maximum p for which $f(p) < p$) of $2^{-7.4}$, and for 5 shares, we have:

$$f_5(p) \approx 2870p^3 + 476.4p^3\sqrt{p} + 75p^4 + 9230.2p^5 + \mathcal{O}(p^6)$$

with a maximum leakage probability (*i.e.* the maximum p for which $f(p) < p$) of $2^{-5.77}$. Table 8 displays the levels of expansion k that are required for gadgets G_{add}^3 and G_{add}^5 and different probabilities to reach 128 bits of security, with the corresponding complexities. The latter are computed using the expanding compiler from [BCP⁺20].

As expected, the resulting complexities for the same leakage probability and security threshold are better for G_{add}^5 . While we could theoretically further increase the number of shares before the expansion, we are limited by the complexity of the automatic tools (here **IronMask**).

Table 8: Security and complexity of expansions of gadgets G_{add}^3 and G_{add}^5 .

	Gadget G_{add}^3					Gadget G_{add}^5				
leakage prob. p	2^{-8}	2^{-12}	2^{-16}	2^{-20}	2^{-24}	2^{-8}	2^{-12}	2^{-16}	2^{-20}	2^{-24}
expansion level k	8	5	4	4	3	4	3	3	2	2
# shares n	6561	243	81	81	27	625	125	125	25	25
# leaking shares t	2187	81	27	27	9	250	50	50	10	10
# randoms	854M	253k	17k	17k	1k	500k	14k	14k	400	400
# additions	1709M	506k	34k	34k	2k	1001k	29k	29k	825	825
# copies	854M	253k	17k	17k	1k	500k	14k	14k	400	400
advantage $f^k(p)$	2^{-152}	2^{-154}	2^{-145}	2^{-209}	2^{-140}	2^{-188}	2^{-175}	2^{-283}	2^{-134}	2^{-170}

Using the expansion method, the resulting G_{add} gadgets are only RPC for a fixed t once the level of expansion is determined (according to the probability and the target security level). Therefore, we rely on the composition theorem from [BCP⁺20] to compute the RPC security of `AddNoiseTo` for selected probabilities and size α of the auxiliary input. Specifically, the security advantage of `AddNoiseTo` is upper bounded by the product of the security advantage of G_{add} with the number of such gadgets. The whole gadget requires t (with its post-expansion value) regular input shares (of $\llbracket x \rrbracket$) and $t \cdot \lceil \frac{\alpha}{n} \rceil$ auxiliary inputs for the simulation of the leakage and t output shares. For instance, with $\alpha = 75$ auxiliary inputs and a leakage probability $p = 2^{-24}$, we need to combine 3 gadgets G_{add} to form a gadget `AddNoiseTo`. The resulting RPC advantage is upper bounded by 2^{-169} , and 10 shares from the main input and 30 auxiliary inputs are required for the simulation. As for the complexity, the noise gadget requires 1200 randoms and implements 2475 additions and 1200 copies. In this setting, the selection of $G_{\text{sum-chain}}$ or $G_{\text{sum-tree}}$ does not affect neither the security, nor the complexity (except when parallelization is activated).

B.3 Instantiation of Raccoon from RPE

To implement the key generation and the signature with the expansion strategy, we miss a linear multiplication gadget, G_{cmult} , for which we give instantiations below for 3 and 5 shares. We simply follow the structures of G_{add} and G_{copy} with a refresh of the inputs preceding the linear multiplication.

To implement key generation and signature using the expansion strategy, we need a linear multiplication gadget, G_{cmult} . Below, we provide specific instantiations for 3 and 5 shares. The design mirrors the structures of G_{add} and G_{copy} , with the inputs refreshed before performing the linear multiplication.

$$\begin{aligned}
 &G_{\text{cmult}}^3 : \mathbb{K}^3 \times \mathbb{K} \rightarrow \mathbb{K}^3 && G_{\text{cmult}}^5 : \mathbb{K}^5 \times \mathbb{K} \rightarrow \mathbb{K}^5 \\
 &((a_1, a_2, a_3), \alpha) \mapsto (c_1, c_2, c_3) && ((a_1, a_2, a_3, a_4, a_5), \alpha) \mapsto (c_1, c_2, c_3, c_4, c_5) \\
 &c_1 \leftarrow \alpha \cdot (r_1 + a_1) && c_1 \leftarrow \alpha \cdot (r_1 + r_2 + a_1) \\
 &c_2 \leftarrow \alpha \cdot (r_2 + a_2) && c_2 \leftarrow \alpha \cdot (r_2 + r_3 + a_2) \\
 &c_3 \leftarrow \alpha \cdot (r_1 + r_2 + a_3) && c_3 \leftarrow \alpha \cdot (r_3 + r_4 + a_3) \\
 &&& c_4 \leftarrow \alpha \cdot (r_4 + r_5 + a_4) \\
 &&& c_5 \leftarrow \alpha \cdot (r_5 + r_1 + a_5)
 \end{aligned}$$

We use `IronMask` from [BMRT22] to compute the new RPE advantages for G_{cmult} combined with G_{add} and G_{copy} . They are identical to those obtained for G_{add} without G_{cmult} in Section B.2.

In Table 8, we identify a single configuration where the number of shares n and the number of leaking shares t correspond to an existing version in Raccoon (version 16), specifically considering the total number

of small uniforms r_i and r'_i and the number of them that can leak. We denote by m_r the number of small uniforms that are added to each coefficient in $\mathbb{Z}_q$ and by m_s the number of small uniforms that cannot be revealed for each coefficient in $\mathbb{Z}_q$. Using our expansion strategy, the small uniforms must be captured inside a n -share variable, that is, there exists $\alpha \in \mathbb{N}^*$ such that

$$(\alpha - 1)n < m_r \leq \alpha n.$$

Then, we must ensure that the number of input shares that are required for the leakage simulation is lower than $m_r - m_s$. This constraint can be written as follows:

$$(\alpha - 1)t + \min(t, n - (\alpha n - m_r)) \leq m_r - m_s.$$

It can be observed that the only column in Table 8) satisfying this inequality corresponds to the use of gadget G_{add}^3 with $n = 27$ shares and an expansion level $k = 3$. For each coefficient in $\mathbb{Z}_q$, 27 small uniforms can be packed into a first sharing, while the remaining 5 small uniforms are packed into a second sharing, supplemented with 22 zeros. The RPC advantage of Raccoon's construction under this configuration then represents the probability that $t + 5 = 14$ small uniforms are required to simulate the leakage and t outputs, with $32 - 14 = 18$ small uniforms remaining secret, exceeding the expected 17 ones.

With these parameters in Raccoon-128, we count almost 16 million base gadgets for key generation and nearly 19 million base gadgets for the signature. Since no implementation is available for the decoding gadget using the expansion strategy, its impact on complexity and security is omitted. Consequently, the security advantage is upper-bounded by 2^{-116} in both cases, for a leakage probability of 2^{-24} . This is calculated by multiplying the atomic security advantage of a single gadget (*i.e.* 2^{-140} from Table 8) by the total number of gadgets. The complexity with this expansion strategy, including the number of logical gates and intermediate random values, is provided in Table 7 and can be compared to the original scheme with 16 shares.

C Proofs of Raccoon's Random Probing Security

C.1 Security Proof for Raccoon's Key Generation

Proof (Proof of Theorem 3). Due to the complexity of the key generation circuit, calculating a global cardinal-RPC-AI envelope is computationally impractical. Even before considering the internal wires, we already have $(\ell + k)n_R = 4.608$ n -share inputs and $\ell n_R = 2.048$ outputs, for which enumerating all possible values in $[0, n]$ would be too expensive. As a result, we opt to decompose the scheme into sub-components, each of which is individually proven to be cardinal-RPC-AI-O. For these sub-components, we calculate the corresponding threshold-RPC-AI-O, using a fixed threshold $t = \frac{n}{2}$. Finally, we sum the threshold-RPC-AI-O advantages of these blocks to determine the overall threshold-RPC-AI-O security of the complete scheme.

Specifically, we identify three different types of blocks:

- Block 1, illustrated on Figure 6, consists of G_{sum} and G_{copy} gadgets. We first compute the cardinal-RPC-AI envelope of the combination of both gadgets G_{sum} and G_{copy} , then the threshold-RPC-AI advantage for an attacker who gets $t = \frac{n}{2}$ shares of each output and when the auxiliary input r_i^j is constrained to have a cardinality of at most $t' = 15$ (per polynomial coefficient). The resulting threshold-RPC-AI advantage is then multiplied by $n_R = 512$ and $\ell = 4$ (following an extension of the composition property of [BCP⁺20]) to obtain an upper bound on the threshold-RPC-AI advantage for the left part of the circuit, whose outputs will be used as inputs for G_{cmmult} .
- Gadget G_{cmmult} is first split into three blocks. The $\ell n_R \mathbb{Z}_q$ n -share outputs of the left block are first each copied k times. We compute the cardinal-RPC envelope of the corresponding G_{copies}^k gadget from which we derive its threshold-RPC advantage with t output shares and when we restrict the cardinality of the input shares to t as well. Then each of the ℓk copies in $\mathcal{R}_q$ goes through a linear polynomial multiplication with gadget G_{cpmult} . Gadget G_{cpmult} consists in the generation of n_R copies of n_R elements in $\mathbb{Z}_q$. We are able to tightly compute the threshold-RPC advantage of the combination of G_{copy} gadgets

assembled in a tree to compute n_R copies based on the cardinal-RPC envelopes of each block and by restricting the cardinal of the output shares to t and the cardinal of the input shares to at most t . Then, n_R^2 linear multiplications in $\mathbb{Z}_q$ are performed for which we have the cardinal-RPC envelope from Section 4.2 and from which we can derive the threshold-RPC advantage when restricting the input and output cardinal with respect to t . Finally, we have n_R groups of n_R elements of $\mathbb{Z}_q$ that are summed together. We use gadget G_{sum} with the tree structure to compute additions of 4 shared variables (we need $128 + 32 + 8 + 4 + 1 = 173$ of them). From the cardinal-RPC envelopes of G_{sum} , we derive their threshold-RPC advantage with t and we sum them to obtain the global threshold-RPC advantage of all the additions. Summing all the computed threshold-RPC advantages (following the composition property of [BCP⁺20]) gives us an upper bound the threshold-RPC advantage of G_{cmmult} .

- Lastly, we compute the global threshold-RPC-AI-O advantage for the right block, whose inputs are the outputs of G_{cmmult} . We begin by determining the cardinal-RPC-AI-O envelope for the combination of AddNoiseTo and G_{decode} , then we compute the threshold-RPC-AI-O advantage when we restrict the input shared cardinality to be less or equal to t and the cardinality of the auxiliary inputs $r_i'^j$ to be at most 15. Similar to the process in block 1, we then multiply the resulting threshold-RPC-AI advantage by $n_R = 512$ and $k = 5$ (in line with the composition property from [BCP⁺20]), yielding an upper bound on the threshold-RPC-AI-O advantage for this right block.

All these advantages are computed for different probabilities and for instantiations of RPRefresh with different values of γ for each of the three blocks. The code is provided in Supplementary Material. The resulting threshold-RPC-AI-O advantages are displayed with the corresponding complexities in Table 7. $\square$

C.2 Security Proof for Raccoon’s Signature

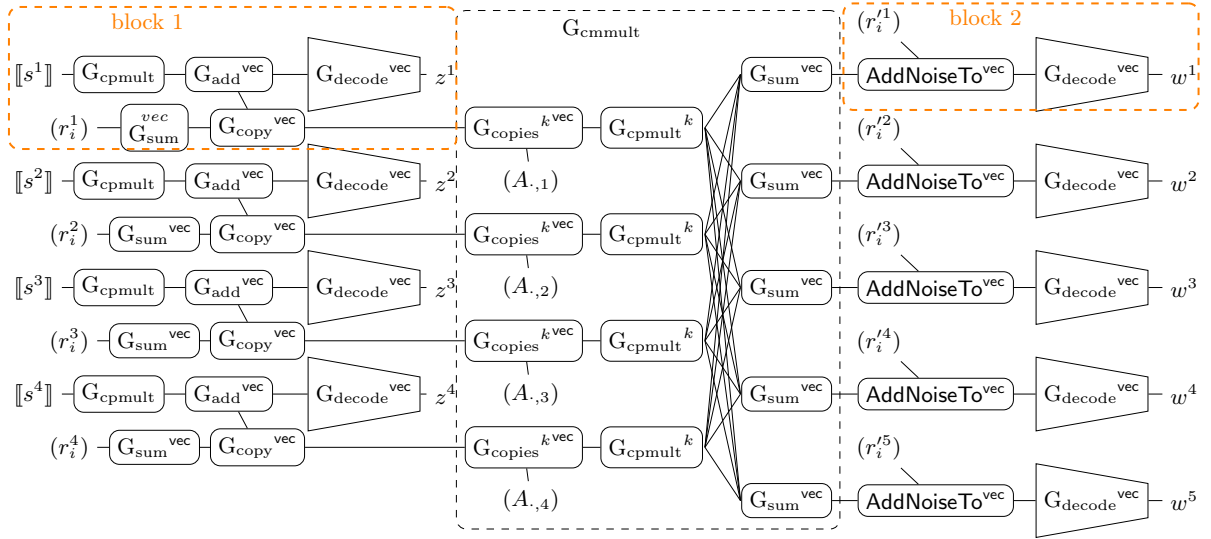


Fig. 7: RP-Sign for Raccoon-128 with $\ell = 4$ and $k = 5$

Proof. The proof for Raccoon’s signature is very close to the proof for Raccoon’s key generation, except for the computation of the threshold-RPC-AI-O advantage of the first block. Specifically, we divide this block into three sub-blocks. The first sub-block contains G_{sum} and G_{copy} gadgets. It is identical to block 1 of the key generation. We reuse the threshold-RPC-AI-O advantage for t as computed for the key generation. The

second sub-block contains G_{decode} and G_{add} gadgets, we compose their envelopes and derive the corresponding threshold-RPC-AI-O advantage for t . Finally, we compute the threshold-RPC advantage of G_{cpmult} for t , as in the proof of Theorem 3. Following [BCP⁺20], we sum the advantages of all three parts (the first ones being multiplied by ℓn_R and the last advantage of G_{cpmult} being multiplied by ℓ) to obtain the global threshold-RPC-AI-O of block 1. The script to compute the global threshold-RPC-AI-O of Raccoon’s signature is provided in Supplementary Material and a few values are provided in Table 7. $\square$

Appendix G

SoK: A Methodology to Achieve Provable Side-Channel Security in Real-World Implementations

Hereafter is appended the full version of our paper [\[BCM⁺25\]](#), joint work with Gaëtan Cassiers, Camille Mutschler, Matthieu Rivain, Thomas Roche, François-Xavier Standaert, and Abdul Rahman Taleb, published at **IACR Communications in Cryptology 2025**.



SoK: A Methodology to Achieve Provable Side-Channel Security in Real-World Implementations

Sonia Belaïd¹ , Gaëtan Cassiers¹ , Camille Mutschler²,
Matthieu Rivain¹ , Thomas Roche³, François-Xavier Standaert⁴ and
Abdul Rahman Taleb¹

¹ CryptoExperts, Paris, France

² Idemia, Courbevoie, France

³ NinjaLab, Montpellier, France

⁴ UCLouvain, Louvain-la-Neuve, Belgium

Abstract. A wide range of countermeasures have been proposed to defend against side-channel attacks, with masking being one of the most effective and commonly used techniques. While theoretical models provide formal security proofs, these often rely on assumptions—sometimes implicit—that can be difficult to assess in practice. As a result, the design of secure masked implementations frequently combines proven theoretical arguments with heuristic and empirical validation. Despite the significant body of work, the literature still lacks a cohesive and well-defined framework for translating theoretical security guarantees into practical implementations on physical devices. Specifically, there remains a gap in connecting provable results from abstract models to quantitative security guarantees at the implementation level.

In this Systematization of Knowledge (SoK), we aim to provide a comprehensive methodology to transform abstract cryptographic algorithms into physically secure implementations against side-channel attacks on microcontrollers. We introduce new tools to adapt the ideal noisy leakage model to practical, real-world scenarios, and we integrate state-of-the-art techniques to build secure implementations based on this model. Our work systematizes the design objectives necessary for achieving high security levels in embedded devices and identifies the remaining challenges in concretely applying security reductions. By bridging the gap between theory and practice, we seek to provide a foundation for future research that can develop implementations with proven security against side-channel attacks, based on well-understood leakage assumptions.

Keywords: masking · provable side-channel security · random probing model · noisy leakage model · methodology · physical assumptions

1 Introduction

Cryptographic algorithms are traditionally analyzed within the black-box model, where the adversary’s knowledge is limited to inputs and outputs. However, since the late 1990s, side-channel attacks have demonstrated that physical implementations of cryptographic algorithms on embedded devices can be vulnerable to physical emanations, such as

E-mail: sonia.belaid@cryptoexperts.com (Sonia Belaïd), gaetan.cassiers@uclouvain.be (Gaëtan Cassiers), camille.mutschler@idemia.com (Camille Mutschler), matthieu.rivain@cryptoexperts.com (Matthieu Rivain), thomas@ninja-lab.io (Thomas Roche), fstandae@uclouvain.be (François-Xavier Standaert), taleb.abdulrahman1@gmail.com (Abdul Rahman Taleb)



the execution time [Koc96], device temperature [HS13], power consumption [KJJ99], or electromagnetic radiation [QS01] during the algorithm execution. These physical emissions leak information about the internal state of cryptographic operations, thus breaking the theoretical security guarantees provided by the black-box model.

In response, several countermeasures have been studied to protect cryptographic algorithms. Among the different approaches, one of the most widely used is known as *masking*, simultaneously introduced by Chari et al. [CJRR99], and by Goubin and Patarin [GP99] in 1999. It consists in splitting a sensitive variable x into n random *shares*, among which any combination of $n - 1$ shares does not reveal any secret information. This can be achieved by generating $n - 1$ shares uniformly at random $x_1, \dots, x_{n-1}$ and computing the last share x_n so that $x = x_1 * \dots * x_{n-1} * x_n$ according to some group law $*$. The motivation is to make it more difficult for an attacker to recover a secret by manipulating the shares instead of the sensitive value. Indeed, the adversary must recombine information from all the shares to learn something about the sensitive value. Assuming that the information gathered through side-channel observations involves some kind of noise, it has been shown that it becomes exponentially harder to recover the secret as the number of shares grows [BCG⁺23, BS21, GS18].

Meanwhile, proving or validating such security levels in practice remains a challenging task. Generally, providing security guarantees against side-channel attacks is tricky, and several works tackle this issue [DFS15a, JS17, GPSS18, BS21]. The approaches currently found in the literature range from purely qualitative solutions such as leakage detection (e.g., ISO17825 [2716, WO19]) or test vector leakage assessment (TVLA) [GGJR⁺11], which aim to detect information leakage using statistical analysis, to more quantitative solutions such as mounting known attacks on the implementation and inferring the security level from the best attacks. For instance, common-criteria certification procedures currently follow this empirical approach to validate the security of implementations for smartcards against side-channel attacks [BS21, BGNT15]. Yet, these methods often lack the formal rigor and comprehensive security guarantees that theoretical models offer. This dichotomy highlights a critical gap between theory and practice in the field of side-channel security.

To address this gap, the community has introduced formal leakage models that provide a theoretical foundation for analyzing masked implementations. The t -probing model, introduced by Ishai, Sahai, and Wagner in 2003 [ISW03], is the most well-known and allows for formal security proofs by assuming that an adversary can observe the exact values of up to t intermediate variables. An arithmetic circuit is then secure in this model if no such leakage of t variables reveals information about the sensitive variables. Despite its wide use by the community [SP06, RP10, CPRR14, BBP⁺16, CRZ18] thanks to its convenience to build security proofs, the t -probing model sometimes fails to reflect the reality of embedded devices. For instance, it does not capture *horizontal attacks* [BCPZ16], which exploit the repeated manipulation of variables within an execution.

These issues motivated the formalization of the *noisy leakage model* [PR13]. This model better captures the reality of embedded devices by assuming that each intermediate variable leaks a noisy function of its value. However, proving security in the noisy leakage model [BCG⁺23, MS23] is more complex than in the t -probing model. In 2014, Duc, Dziembowski, and Faust [DDF14, DDF19] proposed a security reduction from the noisy model to the t -probing model, relying on an intermediate model, the *random probing model*, which benefits from a tighter reduction with the noisy leakage model. In a nutshell, it assumes that every wire in the circuit leaks with some constant leakage probability. This leakage probability is related to the amount of side-channel noise in practice. The random probing model captures *horizontal attacks*, and has been studied recently in many works [Ajt11, ADF16, AIS18, BCP⁺20, BRT21, BRTV21, CFOS21].

Despite the usefulness of these leakage models in providing formal guarantees, their application to practical implementations remains an ongoing challenge. The literature currently lacks a unified methodology for translating these theoretical constructions into secure physical implementations. Additionally, practical implementations often violate the key assumptions of the models, particularly the *data isolation* assumption—which states that the leakage of an elementary operation depends only on its inputs—and the *noise independence* assumption, which assumes that noise from different operations is independent.

The data isolation assumption can be easily broken, for instance, due to physical effects on a device. In particular, transitions occurring on memory buses or CPU registers between a previously processed value $\mathbf{x}_{i-1}$ and the current one $\mathbf{x}_i$ usually leak some information correlated to $\mathbf{x}_{i-1} \oplus \mathbf{x}_i$, which violates the data isolation principle [CGP⁺12, BGG⁺14]. On the hardware level, glitches further make the successive gates’ leakages mutually dependent on their respective inputs [MPG05, MPO05]. On the software level, CPU synchronization limits, but does not eliminate, the issue of glitches. These issues can be avoided by adding registers and controlling transitions [FGM⁺18, CS21] in hardware, and by trying to avoid transitions using assembly programming tricks in software [GMPO19, BC22, BGG⁺22]. However, these techniques still rely on abstract models for the leakage, and current techniques in the literature test the data isolation assumption only indirectly, by estimating the statistical security order of an implementation [BDF⁺17, SM15].

The noise independence assumption is similarly difficult to guarantee in practice. Side-channel leakage is typically multivariate, and noise from successive operations often exhibits dependencies, complicating the assumption of independence. Although this assumption is frequently made, its impact on security has only been explored at a high level in the literature [GS18, Cho15].

Contributions. Our contributions can be summarized as follows.

Methodology for Secure Implementations. We present a comprehensive and systematic methodology to transform abstract cryptographic algorithms into concrete implementations that are secure against side-channel attacks on microcontrollers. This process leverages a random probing compiler and carefully applies the reduction from the noisy leakage model to the random probing model, making it applicable to physical implementations. While this reduction is well understood in theoretical contexts, our contribution lies in systematically summarizing the practical assumptions and challenges that must be addressed to achieve formally secure implementations on real-world devices.

Novel Tools for Practical Challenges. We introduce new tools to address key technical challenges in adapting theoretical models to physical implementations:

- **Data Isolation Assumption Validation:** We explain how to enforce the data isolation assumption and introduce a novel practical test for its validation on a physical implementation. This test, conducted on a real STM32F3 MCU target using NewAE’s ChipWhisperer-Lite CW1173 board, offers a direct and practical approach, standing in contrast to indirect validation methods previously discussed in the literature. While the test does not offer formal proof of the assumption, it is the first of its kind to directly address and empirically validate this hypothesis in a systematic and reproducible manner.
- **Noise Independence Analysis:** We propose a method for integrating the noise independence assumption into security analyses, allowing for the quantification of security loss when this assumption is not fully met. Specifically, we introduce a relaxation of the assumption that splits the noise affecting each operation during algorithm

execution into independent components. We first demonstrate a straightforward method for achieving this split and then formulate the problem as a constrained optimization to improve scalability for larger computations. While we provide a non-optimal but practical solution, we leave the development of an optimal and efficient solution as an open problem for future research.

Identifying Design Goals and Remaining Challenges. We systematically identify the key design objectives necessary to achieve high-security levels in physical implementations using our security reduction methodology. Furthermore, we highlight the remaining limitations and open questions concerning the practical usability of leakage models. Our SoK contribution lies in bridging the gap between theoretical models and practical implementation challenges, motivating further research to close this gap. For example, future work could explore the quantification of signal independence loss or identify an optimal solution for the noise split relaxation to maximize security in practice.

Organization. The organization of the paper is as follows. In [Section 2](#), we provide the necessary background and formalize key assumptions. [Section 3](#) outlines our proposed methodology and states our main result: our methodology can produce implementations that are practically secure against side-channel attacks. We further detail the procedures to enforce or relax the identified assumptions in [Section 4](#) and demonstrate the application of our methodology on a real-world target (STM32F3 MCU with ChipWhisperer-Lite CW1173). Finally, we conclude with discussions and future research directions in [Section 5](#).

2 Technical Background

2.1 Notations

We denote by $\mathcal{V}$ a finite set called the variable space and by $\mathcal{X}$ the input space for the leakage. We denote by $\mathcal{Y}$ the leakage distribution. We use capital letters to denote random variables over a set or a distribution, e.g., X denotes a random variable over $\mathcal{X}$, and $Y(\mathbf{x})$ denotes a random variable (or equivalently a leakage function) over the distribution $\mathcal{Y}$, taking as input $\mathbf{x}$, a value over the input space $\mathcal{X}$. We denote by $\mathbf{y}$ a leakage trace, i.e. a realization of $Y(\mathbf{x})$. Any two probability distributions D_1 and D_2 are said ε -close, denoted $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε , that is $\text{SD}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon$, where $p_{D_1}(\cdot)$ and $p_{D_2}(\cdot)$ denote the probability mass functions of D_1 and D_2 .

2.2 Abstract Circuits

We recall the definition of an abstract circuit (family) and the definition of a circuit compiler (CC, Enc, Dec) which turns an abstract circuit into a randomized circuit.

Definition 1 (Abstract Circuit Family). An *abstract circuit family* is a pair $\mathbb{C} = (\mathcal{V}, \mathcal{G})$ such that

- $\mathcal{V}$ is the *variable space*,
- $\mathcal{G}$ is a set of functions, called the gate family. For each function $g \in \mathcal{G}$, there exists $\ell, m \in \mathbb{N}$ such that $g : \mathcal{V}^\ell \rightarrow \mathcal{V}^m$.

An *abstract circuit* C belonging to the family $\mathbb{C} = (\mathcal{V}, \mathcal{G})$, which is written $C \in \mathbb{C}$, is defined as an acyclic directed graph whose edges are *wires* carrying values over $\mathcal{V}$, and vertices are *gates* processing operations over $\mathcal{V}$. It is further formally composed of input gates of fan-in 0 and fan-out 1 and output gates of fan-in 1 and fan-out 0. Evaluating

an ℓ -input m -output circuit C consists of writing an input $\mathbf{x} \in \mathcal{V}^\ell$ in the ℓ input gates, processing the gates from input gates to output gates, then reading the output $\mathbf{z} \in \mathcal{V}^m$ from the m output gates. This is denoted by $\mathbf{z} = C(\mathbf{x})$. During the evaluation process, each wire in the circuit is assigned with a value on $\mathcal{V}$. We call the tuple of all these wire values a *wire assignment* of C (on input $\mathbf{x}$).

Definition 2 (Circuit Compiler). A *circuit compiler* is a triplet of algorithms (CC, Enc, Dec) defined as follows:

- CC (circuit compilation) is a deterministic algorithm that takes as input an abstract circuit C from a family of circuits $\mathbb{C} = (\mathcal{V}, \mathcal{G})$ and outputs a randomized circuit $\hat{C}$.
- Enc (input encoding) is a probabilistic algorithm that maps an input $\mathbf{x} \in \mathcal{V}^\ell$ to an encoded input $\hat{\mathbf{x}} \in \mathcal{V}^{\ell'}$.
- Dec (output decoding) is a deterministic algorithm that maps an encoded output $\hat{\mathbf{z}} \in \mathcal{V}^{m'}$ to a plain output $\mathbf{z} \in \mathcal{V}^m$.

These three algorithms satisfy the following properties:

- **Correctness:** For every circuit C of input length ℓ , and for every $\mathbf{x} \in \mathcal{V}^\ell$, we have

$$P[\text{Dec}(\hat{C}(\hat{\mathbf{x}})) = C(\mathbf{x}) \mid \hat{\mathbf{x}} \leftarrow \text{Enc}(\mathbf{x})] = 1 ,$$

where $\hat{C} = \text{CC}(C)$.

- **Efficiency:** For some security parameter $\lambda \in \mathbb{N}$, the running time of $\text{CC}(C)$ is $\text{poly}(\lambda, |C|)$, the running time of $\text{Enc}(\mathbf{x})$ is $\text{poly}(\lambda, |\mathbf{x}|)$ and the running time of $\text{Dec}(\hat{\mathbf{z}})$ is $\text{poly}(\lambda, |\hat{\mathbf{z}}|)$, where $\text{poly}(\lambda, q) = O(\lambda^{k_1} q^{k_2})$ for some constants k_1, k_2 .

2.3 Random-Probing Model

Let $p \in [0, 1]$ be some constant leakage probability parameter, usually called *leakage rate*. The random probing leakage can be defined in two ways depending on whether we consider leakage on the wires or the gates of an abstract circuit C from a family $\mathbb{C} = (\mathcal{V}, \mathcal{G})$.

In the *wire leakage* setting, the p -random probing model states that during the evaluation of a circuit C , each wire leaks its value with probability p (and leaks nothing otherwise), where all the wire leakage events are mutually independent. To formally define this leakage, we consider two probabilistic algorithms:

- The *leaking-wires sampler* takes as input an abstract circuit C and a probability $p \in [0, 1]$, and outputs a set W , denoted as

$$W \leftarrow \text{LeakingWires}(C, p) ,$$

where W is constructed by including each wire label from the circuit C with probability p to W (where all the probabilities are mutually independent).

- The *assign-wires sampler* takes as input an abstract circuit C , a set of wire labels W (subset of the wire labels of C), and an input $\mathbf{x} \in \mathcal{V}^\ell$, and it outputs a $|W|$ -tuple $\mathbf{w} \in \mathcal{V}^{|W|}$, denoted as

$$\mathbf{w} \leftarrow \text{AssignWires}(C, W, \mathbf{x}) ,$$

where $\mathbf{w}$ corresponds to the assignments of the wires of C with label in W for an evaluation on input $\mathbf{x}$.

By convention, we do not consider leakage on the output wires (i.e. input wires of the output gate) of a circuit, since when composing several circuits, these wires become input wires to the next circuit.

We can analogously define the *gate leakage* setting with similar probabilistic procedures, with the difference between both leakages illustrated in Figure 1. The leaking-gates sampler $G \leftarrow \text{LeakingGates}(C, p)$ outputs a set G of gates labels instead of wire labels (LeakingGates ignores the input and output gates, which are merged when composing circuits). Then, the assign-gates sampler $\mathbf{g} \leftarrow \text{AssignGates}(C, G, \mathbf{x})$ assigns to each gate of label in G its internal state during the evaluation of C (i.e. $\mathbf{g}$ is the assignment of the internal states of the gates of C with label in G for an evaluation on input $\mathbf{x}$). The internal state of a gate is modelled as an arbitrary function of the gate's inputs.

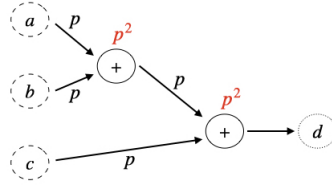


Figure 1: Toy circuit illustrating random probing leakage. Dashed circles (a, b, c) are input gates, while the dotted circle (d) is the output gate. In the wire leakage setting, each wire leaks with probability p , while in the gate leakage setting, each gate leaks its internal state with probability p^2 . **Lemma 1** states that if the circuit is (p, ε) -RP secure in the wire setting, then it is (p^2, ε) -RP secure in the gate setting.

Based on these notions, we now formally define the (wire or gate) random probing leakage of a circuit.

Definition 3 (Random Probing Leakage). The p -random probing wire leakage of an abstract circuit C with ℓ inputs, on input $\mathbf{x} \in \mathcal{V}^\ell$ is the distribution $\mathcal{L}_p^{\text{wire}}(C, \mathbf{x})$ obtained by composing the leaking-wires and assign-wires samplers as

$$\mathcal{L}_p^{\text{wire}}(C, \mathbf{x}) \stackrel{\text{id}}{=} \text{AssignWires}(C, \text{LeakingWires}(C, p), \mathbf{x}) .$$

For the p -random probing gate leakage, $\mathcal{L}_p^{\text{gate}}(C, \mathbf{x})$ is obtained as

$$\mathcal{L}_p^{\text{gate}}(C, \mathbf{x}) \stackrel{\text{id}}{=} \text{AssignGates}(C, \text{LeakingGates}(C, p), \mathbf{x}) .$$

We can define the random probing security of an abstract circuit C .

Definition 4 (Random Probing Security). An abstract circuit C with ℓ inputs, from a family of circuits $\mathbb{C} = (\mathcal{V}, \mathcal{G})$, is (p, ε) -random probing secure (RPS) in the wire leakage setting with respect to encoding Enc if there exists a simulator Sim such that for every $\mathbf{x} \in \mathcal{V}^\ell$:

$$\text{Sim}(C) \approx_\varepsilon \mathcal{L}_p^{\text{wire}}(C, \text{Enc}(\mathbf{x})) . \quad (1)$$

A circuit compiler $(\text{CC}, \text{Enc}, \text{Dec})$ is (p, ε) -random probing secure in the wire leakage setting if, for every circuit C , the compiled circuit $\hat{C} = \text{CC}(C)$ is $(p, |\hat{C}| \cdot \varepsilon)$ -random probing secure in the wire leakage setting where $|\hat{C}|$ is the size of the original circuit.

We equivalently define (p, ε) -random probing security for a circuit and a circuit compiler in the gate leakage setting, where we use $\mathcal{L}_p^{\text{gate}}$ instead of $\mathcal{L}_p^{\text{wire}}$.

We have the following reduction of security, which states that if a circuit is random probing secure in the wire leakage setting, then it is secure in the gate leakage setting ¹.

¹Note that **Lemma 1** extends to gates with larger fan-in. Considering gates with n inputs, one gets $p' = p^n$

Lemma 1. *Let C be an abstract circuit with ℓ inputs from a family of circuits $\mathbb{C} = (\mathcal{V}, \mathcal{G})$ such that each gate $g \in \mathcal{G}$ has at most two input wires. If C is (p, ε) -random probing secure with respect to encoding Enc in the wire leakage setting, then C is (p', ε') -random probing secure in the gate leakage setting, with $p' = p^2$ and $\varepsilon' = \varepsilon$.*

Proof of Lemma 1. Let C be an abstract circuit with ℓ inputs and suppose that C is (p, ε) -random probing secure in the wire leakage setting. Then, there exists a simulator that we shall denote Sim_{wire} such that $\text{Sim}_{\text{wire}}(C) \approx_{\varepsilon} \mathcal{L}_p^{\text{wire}}(C, \text{Enc}(\mathbf{x}))$. We now construct another simulator Sim_{gate} as follows. Sim_{gate} starts by running Sim_{wire} , and if Sim_{wire} fails (or aborts), then Sim_{gate} aborts too. Otherwise, for each gate g in C , if all input wires to g are simulated and output by Sim_{wire} , we let Sim_{gate} output a simulation of the inner state of g using the simulation of its input wires by Sim_{wire} . Note that this is possible since the inner state of g only depends on its input wires. Since we have that $\text{Sim}_{\text{wire}}(C) \approx_{\varepsilon} \mathcal{L}_p^{\text{wire}}(C, \text{Enc}(\mathbf{x}))$, then each wire in C is simulated by Sim_{wire} with probability p independently of all the other wires. Consequently, each gate in C is simulated by Sim_{gate} with probability at least p^2 independently of all the other gates. Finally, since Sim_{gate} aborts if and only if Sim_{wire} aborts with probability ε , we get that $\text{Sim}_{\text{gate}}(C) \approx_{\varepsilon} \mathcal{L}_p^{\text{gate}}(C, \text{Enc}(\mathbf{x}))$. Hence, C is (p^2, ε) -random probing secure in the gate leakage setting, which concludes the proof. $\square$

2.4 Noisy Leakage Model

The noisy leakage model was formalized in [PR13]. In this model, a leaking computation is modeled by a sequence of *elementary operations* $(g_i)_i$ accessing a common memory called *internal state*. Each elementary operation reads its input and writes its output on the internal state. When processed on some input $\mathbf{x}$, an elementary operation g_i reveals $f_i(\mathbf{x})$ to the adversary for some *noisy leakage function* f_i . A noisy leakage function takes two arguments: the value $\mathbf{x}$ held by the accessed part of the internal state (*data isolation assumption*) and a random string ρ long enough to model the leakage noise. Each execution leaks the values $(f_i(\mathbf{x}_i, \rho_i))_i$ where the $\mathbf{x}_i$'s are the successive intermediate values (from the internal state) in input of the elementary operations g_i 's and ρ_i 's are fresh random strings. We stress that all the ρ_i 's involved in successive executions are uniformly and independently drawn (*independent noise assumption*).

We note that from a formal point of view, there is an equivalence between the circuit model used by the gate-leakage random probing model and the internal state model used by the noisy leakage model. In both cases the computation is divided into sub-computations (either gates or elementary operations) and the full leakage is composed of the outputs of leakage functions (either random probing functions or noisy functions) applied to all the sub-computations input. The internal state model has the advantage of being cosmetically closer to a real software implementation, moreover it is useful to consider the order of operations while relaxing the data isolation and noise independence assumptions (as discussed later).

For the sake of simplicity, we shall omit the random string parameter, which leads to the notation $f_i(\mathbf{x})$ where $\mathbf{x}$ is the accessed value. Note that $f_i(\mathbf{x})$ can be seen as the output of a probabilistic algorithm. In particular, $f_i(\mathbf{x})$ can take several values with a given probability distribution, and can therefore be considered as a random variable. The noisy property of f is captured by assuming that the bias introduced in the distribution of a uniform random variable X given the leakage $f(X)$ is bounded. This is formalized in the next definition:

Definition 5 (Noisy Function). Let $\mathcal{X}$ be a finite set and let $\delta \in \mathbb{R}$. A δ -noisy leakage function f on $\mathcal{X}$ is a function of domain $\mathcal{X} \times \{0, 1\}^{|\rho|}$ for some $|\rho| \in \mathbb{N}$ such that

$$\beta(X|Y) := \sum_{y \in \text{Range}(f)} \Pr(Y = y) \cdot \Delta((X | Y = y); X) \leq \delta, \quad (2)$$

where Δ is a statistical distance measure, X is a uniform random variable over $\mathcal{X}$ and where $Y = f(X, R)$ for a uniform random variable R over $\{0, 1\}^{|\rho|}$.

The above definition depends on the notion of statistical distance. In the original definition from [PR13], the authors use the L_2 norm. The authors of [DDF14] then suggested to use the L_1 norm (normalized by $\frac{1}{2}$). It was later suggested in [PGMP19] to use a statistical distance notion based on the *relative error*. Noisy functions based on this distance are referred to as *average relative error* (ARE) noisy leakage functions in [PGMP19] since the relative error is averaged over the distribution of the leakage Y in Equation 2.

As recalled hereafter, the noisy leakage metrics based on the L_1 statistical distance (SD) and the ARE enjoy useful security reductions to the random probing model. We recall the definition of these two metrics based on the *pointwise mutual information*.

Definition 6 (Pointwise Mutual Information). Let X, Y be random variables over $\mathcal{X}, \mathcal{Y}$ respectively. For any $x \in \mathcal{X}, y \in \mathcal{Y}$, the exponential form of the pointwise mutual information (PMI) is defined as:

$$\text{PMI}_{X,Y}(x, y) = \frac{P[X = x, Y = y]}{P[X = x] \cdot P[Y = y]} - 1 .$$

Definition 7. Let X, Y be random variables over $\mathcal{X}, \mathcal{Y}$ respectively. We can define the L1 statistical distance (SD) as follows:

$$\text{SD}(X|Y) = \frac{1}{2} \mathbb{E}_{Y=y} \mathbb{E}_{X=x} [|\text{PMI}_{X,Y}(x, y)|] .$$

The average relative error (ARE) can also be expressed as:

$$\text{ARE}(X|Y) = \mathbb{E}_{Y=y} \left[\max_x |\text{PMI}_{X,Y}(x, y)| \right] .$$

From random probing to noisy leakage security. In [DDF14], Duc, Dziembowski, and Faust show the following security reduction: any circuit which is (p, ε) -secure in the random probing model is also (δ, ε) -secure in the noisy leakage model (for the same parameter ε) defined w.r.t. the metric $\beta(X|Y) = \text{SD}(X|Y)$ and for any $\delta \leq p/|\mathcal{X}|$, where $\mathcal{X}$ is the input space of the abstract gates / elementary operations.² This result was later extended to the noisy leakage model defined w.r.t. the metric $\beta(X|Y) = \text{ARE}(X|Y)$ in the work of Prest et al. [PGMP19]. Those security reductions directly hold from the following key lemma.

Lemma 2 ([DDF14, PGMP19]). Let $\phi_p : \mathcal{X} \rightarrow \mathcal{X} \cup \{\perp\}$ the randomized function defined for every $p \in [0, 1]$ as

$$\phi_p(x) = \begin{cases} \perp & \text{with probability } 1 - p \\ x & \text{with probability } p \end{cases} \quad (3)$$

Let $f : \mathcal{X} \rightarrow \mathcal{Y}$ be a δ -noisy leakage function (w.r.t. SD or ARE). There exists a randomized function $f' : \mathcal{X} \cup \{\perp\} \rightarrow \mathcal{Y}$ such that for every $x \in \mathcal{X}$ we have

$$f(x) = f'(\phi_p(x)) \quad \text{with} \quad \begin{cases} p \leq \delta \cdot |\mathcal{X}| & \text{if } \text{SD}(X|f(X)) \leq \delta \\ p \leq \delta & \text{if } \text{ARE}(X|f(X)) \leq \delta \end{cases} \quad (4)$$

²The input space $\mathcal{X}$ is different than the variable space $\mathcal{V}$ for the variables in a circuit. Typically, when the leakage is defined on the internal state of the gate, the latter can be described by both its input wires, and hence the input space is $\mathcal{X} = \mathcal{V}^2$.

We recall that the ARE is a worst-case metric, contrary to the SD, which is an average-case metric. This explains the tighter reduction (i.e. no loss induced by the size of the input space) using the ARE from the noisy model to the random probing model since the latter is also a worst-case model.

Besides this worst-case vs. average-case question, we note that the SD and ARE can be connected to metrics that are used in practice to evaluate the security of a leaking implementation. For example, the SD can be expressed using Mutual Information (MI) thanks to [Dod12] and the Mutual Information can (under some conditions) be expressed using the Signal-to-Noise Ratio (SNR) [Man04] and the correlation coefficient [BCO04] thanks to [MOS11]. The MI is a standard metric to analyze multivariate leakages while the SNR and correlation coefficient are among the most popular tools for univariate security assessments.

In the following, we shall refer to the reduction from [DDF14] using the SD metric as the *DDF reduction*, and to that from [PGMP19] using the ARE metric as the *PGMP reduction*.

2.5 Physical Assumptions

The noisy leakage model has been argued to capture power and electromagnetic leakages effectively. In all generality, an elementary operation processing a value $\mathbf{x}$ gives rise to a leakage trace $Y(\mathbf{x})$ which is a multivariate random variable (a.k.a. a random vector) following a distribution whose parameters depend on $\mathbf{x}$. In most practical contexts, this distribution is well approximated by a multivariate Gaussian $\mathcal{N}(\mathbf{m}_x, \Sigma)$ for some parameters $\mathbf{m}_x$ (mean vector) and Σ (covariance matrix), see e.g. [CRR03, SLP05]. Such parameters can be inferred in practice through a profiling of the device, from which we obtain the noisy leakage metric δ by evaluating Equation 2.

We still need to stress that, as is, the noisy leakage model relies on two assumptions about the underlying physical device which might not be verified in practice without further care.

Assumption 1 (Data isolation). *A leakage function f_i corresponding to the elementary operation $g_i(\mathbf{x}_i)$ only depends on the current state $\mathbf{x}_i$ and not on previously accessed parts of the state: $\mathbf{x}_{i-1}, \mathbf{x}_{i-2}, \dots$. The leakage is then assumed to respect some data isolation between successive elementary operations.*

However, as mentioned in the introduction, physical effects such as glitches and transitions will likely break this implicit assumption. Hence, one should take special care and enforce data isolation for the model to be valid.

Note that this assumption can be relaxed to allow the leakage to be composed of linear combinations of independent noisy leakage functions [BDF⁺17]. Such linear combinations do not affect the statistical security order since given leakage samples of which the statistical moments up to a given order are independent of any sensitive variable, their linear combination will have the same guarantee

Assumption 2 (Noise independence). *The leakage noises from the successive elementary operations are independent from each other. Formally, the random tape ρ_i in each $f_i(\mathbf{x}_i, \rho_i)$ is sampled as a fresh uniform string.*

In practice, this assumption does not easily hold: if one cuts a leakage trace into several sub-traces corresponding to successive elementary operations, the noises in the successive sub-traces would likely include some part of dependency. Indeed, a correlation exists between successive leakage points, which makes multivariate statistics particularly useful for side-channel attacks [CRR03].

In the following, we shall refer to the original noisy leakage model, which relies on the two aforementioned assumptions as the *idealized noisy leakage model*. We will explore

how to relax or enforce those physical assumptions to reduce the security of a physical implementation to that of an abstract implementation in the idealized noisy leakage model, which subsequently reduces to the random probing security.

3 Methodology

The theoretical community introduced many constructions proven secure in the (random) probing and noisy leakage models with a quantified security level. Meanwhile, it is unclear how to implement such constructions on physical devices while fully/quantitatively leveraging the proven security guarantees. Indeed, the existing literature does not explain all the steps nor states all the hypotheses required for preserving proven security claims in practice. In this section, we rigorously exhibit all the steps to turn an abstract circuit into a physical implementation satisfying provable security against side-channel attacks³. As illustrated in Figure 2, these steps can be split into two phases:

- (i) a *characterization phase* which only depends on the device and the side-channel acquisition tool (i.e., without any knowledge of the abstract circuit). It includes the implementation of specific gates (Step 1), the analysis of **Assumption 1** (data isolation) to exhibit a relevant whitening procedure (Step 2), the characterization of the leakage (Step 3), the enforcement and relaxation of **Assumption 2** (noise independence) (Step 4), and the estimation of the noisy leakage parameter (Step 5),
- (ii) a *compilation phase* using the outputs of the characterization phase to turn an abstract circuit $C \in \mathbb{C}$, with $\mathbb{C} = (\mathcal{V}, \mathcal{G})$ an abstract circuit family, into a practically secure implementation for a given security level λ (Step 6). This phase relies on the usage of a secure random probing compiler.

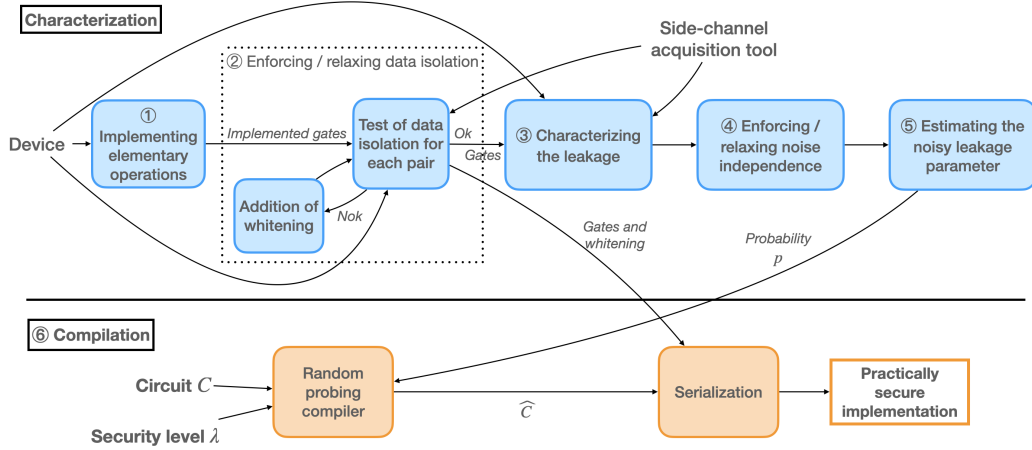


Figure 2: Illustration of our methodology

Overall, our methodology provably ensures security against side-channel attacks, as shown in Section 3.7. All the intermediate steps are described at a high level in the next subsections. In Section 4, we describe the experiments that we ran to validate the practicality of our methodology and develop some dedicated procedures for data and noise isolation.

³We describe our methodology in the context of software implementation, where elementary calculations align with software routines. The generalization to hardware implementations is left for future work, see Section 5.

3.1 Step 1: Implementing Abstract Gates

The first step of our methodology consists of implementing abstract gates as software routines. A physical elementary operation abstracted as a gate by the noisy leakage model (see Section 2.4) first looks up its operands from memory (the computation state), then executes a sequence of arithmetic instructions (implementing the gate functionality $g \in \mathcal{G}$), and finally writes back the result to memory. This process generates some side-channel leakage depending on the executed instructions and the processed data, which is the leakage of the physical elementary operation abstracted by the noisy leakage model. A developer must first translate this behavior into a software routine on a physical device. We propose to implement such a routine in ARM assembly as follows (with the `xor` operation as an example):

```
operation_xor:
    ldr r0, [r0]
    ldr r1, [r1]
    eor r0, r1, r0 // For other operations, change instruction
    str r0, [r2]
```

with the following C signature:

```
void operation_xor(const uint32* aPtr, const uint32* bPtr, uint32* cPtr);
```

We define a routine for each abstract gate $g \in \mathcal{G}$ which, when executed on the target device, behaves as a physical elementary operation abstracted by the noisy model. From these implementations of the abstract gates, any circuit $C \in \mathbb{C}$ can be compiled into a physical implementation on the target device. This implementation takes the form of a sequence of calls to the elementary operations, looking like the following C-syntax example:

```
operation1(a1Ptr, b1Ptr, c1Ptr);
operation2(a2Ptr, b2Ptr, c2Ptr);
...
```

The routines `operation1`, `operation2`... are all among the implemented gate routines which are mapped from the gates of the circuit. The pointer arguments (`a1Ptr`, `b1Ptr`, `c1Ptr`), (`a2Ptr`, `b2Ptr`, `c2Ptr`), ... are constant addresses triplets which encode the data dependency of the implementation, i.e., the wires in the abstract circuit.⁴

3.2 Step 2: Enforcing / Relaxing Data Isolation

Once the syntax of elementary operations is fixed, the physical assumptions made in the noisy leakage model must be satisfied by the implementations in order to use the security reduction. Our methodology first focuses on the data isolation assumption (i.e., Assumption 1), which requires that the leakage of an elementary operation only depends on its inputs, i.e., is independent of the inputs of the previous and the following operations. This assumption rarely holds in practice, since elementary operations executed successively might leak jointly on their manipulated data. Indeed, after the execution of an elementary operation, the data it has processed might be stored in the physical state of the CPU. The leakage of the following elementary operation will then be a (probabilistic) function of the data it processes and of the physical state of the CPU, hence of the previously processed data. This is a well-known issue in the side-channel literature. In particular, this data non-isolation includes the so-called transition leakage observed and analyzed in many works [MOW17, MPW22, MKSM22]. These pitfalls have a direct practical impact, typically leading to losing security orders in the masking scheme [BGG⁺14]. In the provable security setting, this translates to breaking the data isolation assumption. Assuming that each elementary operation leaks a (probabilistic) function of the accessed part of the state

⁴The proposed implementation style is admittedly not very efficient. This paper mainly targets security and simplicity, leaving optimization to future works.

is incorrect: the leakage also depends on the state’s previously accessed part(s). Hence, a developer can not simply implement a circuit as a sequence of the routines introduced in Section 3.1, as the side-channel security can no longer be reduced to the random probing model.

As data isolation plays a crucial role in upholding security proofs and stands as a critical step in our methodology, we introduce a method to enforce it, inspired by prior works (e.g., [BC22, CS21]). Additionally, we design a dedicated test to validate data isolation on a target device.

Enforcing data isolation. We use *data whitening* to enforce the data isolation assumption in our methodology. The principle is to call a routine on constant or random data whose sole purpose is to clean the CPU state from any dependency on the previously processed data. Specifically, after each call to an elementary operation routine, we insert one or more calls for which the arguments point to random or constant data in memory. The intuition is that by relying on a call to a similar elementary operation routine, we expect to clean the data path, namely to write random or constant data in any hardware register containing data-dependent information from the previous call. Nevertheless, although natural, this solution might not suffice to ensure data isolation on some devices. The effectiveness of a whitening routine depends on the microarchitecture of the device’s CPU. Therefore, a developer might have to empirically test several approaches before reaching successful and efficient isolation.

Even with an isolation that avoids all transition and glitches effects across operations, it might not be possible to partition the leakage trace in time intervals whose leakage corresponds to only a single operation. Indeed, the leakage is often subject to low-pass filtering inside the target chip or the measurement chain. As a result, the independent intrinsic leakage of many operations will be linearly combined in the measured trace. As previously mentioned, we can relax the noisy leakage model to allow the leakage to be composed of linear combinations of independent noisy leakage functions. In this case, we aim at ensuring that a leakage function f_i corresponding to the elementary operation $g_i(\mathbf{x}_i)$ does not jointly depend on the current state $\mathbf{x}_i$ and previously accessed parts of the state: $\mathbf{x}_{i-1}, \mathbf{x}_{i-2}, \dots$. In other words, f_i depends on the current state and has at most linear dependencies on the previous parts $\mathbf{x}_{i-1}, \mathbf{x}_{i-2}, \dots$. With this relaxation, we still provide independence between the inputs of the different operations, i.e., data isolation.

Testing data isolation. In Section 4.2, we introduce a novel way to test the effectiveness of a data whitening routine. The idea is to suppose that the leakage distribution can be modeled as a sum of a deterministic function of the first operation’s inputs, a deterministic function of the second operation’s inputs, and some noise value. In other words, we test that the leakage can be decomposed in additive parts that do not jointly depend on the inputs of both operations.

3.3 Step 3: Characterizing the Leakage

Once data isolation is enforced and tested, one can safely infer the leakage distribution of each physical elementary operation. This is a classical problem in the side-channel literature, and we can rely on a solid theoretical and practical ground for this step. We rely on the common assumption [CK13, SLP05] that the leakage distribution $\mathcal{Y}$ of an elementary operation with inputs $\mathbf{x} \in \mathcal{V}^\ell$ takes the form of a deterministic function of $\mathbf{x}$ plus an additive Gaussian noise:

$$\mathcal{Y}_{\mathbf{x}} = d(\mathbf{x}) + \mathcal{N}(\mathbf{0}, \Sigma) , \quad (5)$$

where the deterministic part of the leakage can be written as a linear combination of a predetermined basis of functions $\mathcal{H} = \{h_1, \dots, h_m\}$, i.e.:

$$d(\mathbf{x}) = \sum_{i=1}^m \alpha_i \cdot h_i(\mathbf{x}) . \quad (6)$$

The choice of the basis of functions $\mathcal{H}$ is determined for each elementary operation routine depending on its internal variables. The basis should at least contain one function for each internal variable bit but might also include monomials of higher degrees due to possible coupling effects [FGM⁺18].

In our methodology, we suggest relying on linear regression in order to estimate the deterministic leakage $d(\cdot)$. It involves acquiring an initial set of ℓ_1 traces, which measure the leakage while executing the operation on ℓ_1 inputs generated uniformly at random. We then use this set to infer the coefficients $\{\alpha_i\}_{i=1, \dots, m}$. Subsequently, we can compute the covariance matrix using a new set of ℓ_2 traces on uniform random inputs, allowing us to recover (an estimation of) the covariance matrix Σ .

It's worth emphasizing that while we propose linear regression for leakage estimation, our methodology remains adaptable to other estimation methods. Techniques such as template attacks [CRR03], combined with dimensionality reduction [SA08, CDSU23], and the emerging use of machine learning for side-channel analysis [MDP19, PPM⁺23], all offer viable alternatives. Furthermore, we assume that the engineering effort of optimizing the measurement setup has been performed properly, such that it is infeasible for an adversary to acquire more informative measurements than the ones used for the characterization. Indeed, the characterization steps of the methodology and in particular the estimation of the noise level depend heavily on the optimization of the experimental setup [BUS21, CM24].

3.4 Step 4: Enforcing / Relaxing Noise Independence

Next, we consider the noise independence assumption (Assumption 2) needed for the reduction from the noisy leakage to the random probing model. Namely, in the idealized noisy leakage model, the noise that occurs during the execution of an elementary operation is drawn independently of the noise that occurs during the execution of the previous ones. Hence, this assumption must be satisfied in practice. Meanwhile, it is hard to enforce and test since no clear separation of the noise occurs during a leakage trace. In our methodology, we propose a novel way to relax this assumption. Namely, we keep the Gaussianity hypothesis, but we allow the leakage of the different operations to overlap. We characterize this relaxation and directly reflect it on the security level by providing a reduction from the noisy leakage model with potential noise dependence to the idealized noisy leakage model.

Concretely, we propose to partition the noise distribution into multiple distributions, all while minimizing leakage during each operation. In simpler terms, given k consecutive elementary operations of inputs $\{(\mathbf{x}_i)\}_{1 \leq i \leq k}$, we can represent the overall leakage distribution as

$$\mathcal{Y} = \sum_{i=1}^k d_i(\mathbf{x}_i) + \mathcal{N}(\mathbf{0}, \Sigma) \quad (7)$$

where $d_i(\mathbf{x}_i)$ are the different deterministic signals of the operations, and the noise drawn from $\mathcal{N}(\mathbf{0}, \Sigma)$ is the global noise. Thanks to the data isolation enforcement and test from Section 3.2, the deterministic signals are mutually data independent. Specifically, while the d_i 's might overlap on some time samples, they independently apply to the inputs $\mathbf{x}_i$. This ensures that the global deterministic leakage can be expressed as a sum in Equation 7.

In order to relax the noise independence, our approach consists of finding a set of covariance matrices $\{\Sigma_i\}_{i \in [k]}$ such that $\sum_{i \in [k]} \Sigma_i = \Sigma$. This way, we can split the Gaussian noise distribution $\mathcal{N}(\mathbf{0}, \Sigma)$ into k independent Gaussian distributions $\mathcal{N}(\mathbf{0}, \Sigma_1), \dots, \mathcal{N}(\mathbf{0}, \Sigma_k)$. This representation enables us to split the leakage distribution into several functions $\mathcal{Y}_i = d_i(\mathbf{x}_i) + \mathcal{N}(\mathbf{0}, \Sigma_i)$ for every $i \in \{1, \dots, k\}$. An adversary given a leakage sample of each $\mathcal{Y}_i$ is more powerful than an adversary given a sample of the global leakage $\mathcal{Y}$ because the former can always sum the $\mathcal{Y}_i$ samples to get a $\mathcal{Y}$ sample.

Instantiation. Section 4.4 introduces several methods to define such matrices Σ_i , which sum to the original covariance matrix Σ while minimizing the mutual information between the leakage and the signals.

3.5 Step 5: Estimating the Noisy Leakage Parameter

Recall that once data isolation is enforced and tested as described in Section 3.2, we can empirically characterize the leakage distribution of each elementary operation as described in Section 3.3. Assuming that the leakage takes the form of Equation 5, we compute the coefficients in the deterministic function for each elementary operation (as defined in Equation 6). Then, we can apply the noise relaxation described in Section 3.4 to get the covariance matrix of the Gaussian noise, which we suppose is the same for each elementary operation, following the representation in our optimization problem.

We can then compute the noisy leakage parameter δ related to the δ -noisy leakage model. As explained in Section 2.4, reducing the *idealized* noisy leakage model to the random probing model provides the leakage probability in the latter. More precisely, in order to achieve (δ, ε) -security in the idealized noisy leakage model, an abstract circuit should achieve (p, ε) -security in the random probing model with $p = \gamma \cdot \delta$ for some constant factor γ depending on the noisy leakage metric (e.g. $\gamma = |\mathcal{X}|$ for the SD metric, $\gamma = 1$ for the ARE metric). We describe the two main options:

- The original security reduction [DDF14] (DDF reduction) relies on the statistical distance between a (uniform) variable X and the same variable conditioned on its leakage Y : $\delta = E_y[\text{SD}((X|Y = \mathbf{y}); X)]$. When reducing to the random probing model, this value is multiplied by the size of the input space $\mathcal{X}$ (i.e., the definition set of inputs $\mathbf{x}$ of an elementary operation), hence losing tightness in the tolerated leakage rate through the reduction. In this SD metric, $\gamma = |\mathcal{X}|$.
- In a more recent work [PGMP19] (PGMP reduction), the authors express δ using different noisiness metrics from the pointwise mutual information. From the security reduction viewpoint, the most interesting metric is the average relative error (ARE), a worst-case metric (just like the random probing model), contrary to the statistical distance, which is an average-case metric. When computing $\delta = \text{ARE}(X; X|Y)$, the reduction to the random probing model thus yields tighter results with $\gamma = 1$ and thus $p = \delta$.

In order to estimate the noisy leakage parameter δ in our methodology, we compute both ARE and SD metrics using the inferred leakage model to compare both reductions to the random probing model. Using the pointwise mutual information (Definition 7), we have

$$\begin{aligned} \text{ARE} &= \mathbb{E}_Y \max_{X=\mathbf{x}} \left| \frac{P[X = \mathbf{x}, Y = \mathbf{y}]}{P[X = \mathbf{x}] \cdot P[Y = \mathbf{y}]} - 1 \right| \\ &= \mathbb{E}_Y \max_{X=\mathbf{x}} \left| \frac{P[Y = \mathbf{y} | X = \mathbf{x}]}{\sum_{X=\mathbf{x}'} P[Y = \mathbf{y} | X = \mathbf{x}']} \cdot \frac{1}{P[X = \mathbf{x}]} - 1 \right| \end{aligned} \quad (8)$$

and

$$SD = \mathbb{E}_Y \frac{1}{2} \sum_{X=\mathbf{x}} \left| \frac{P[Y = \mathbf{y}|X = \mathbf{x}]}{\sum_{X=\mathbf{x}'} P[Y = \mathbf{y}|X = \mathbf{x}']} \cdot \frac{1}{P[X = \mathbf{x}]} - 1 \right| \quad (9)$$

From the equations above, we need to compute the sum of the conditional probabilities $P[Y = \mathbf{y}|X = \mathbf{x}']$ for $\mathbf{x}' \in \mathcal{X}$ for ARE and SD estimations. Then, in the case of ARE estimation, we need to find the maximal value of the expression given between $|\cdot|$ in Equation 8 over the values taken by X . While in the case of the SD estimation, we compute a sum over the expression given between $|\cdot|$ in Equation 9. Finally, we must compute the expected value for ARE and SD for Y .

Computing the conditional distribution. In order to estimate the conditional distribution $P[Y = \mathbf{y}|X = \mathbf{x}]$ given a leakage trace and $\mathbf{x}$, we can use the leakage characterization computed earlier, since the conditional distribution is known to be expressed as

$$P[Y = \mathbf{y}|X = \mathbf{x}] = \frac{\exp\left(-\frac{1}{2}(\mathbf{y} - d(\mathbf{x}))^T \Sigma^{-1}(\mathbf{y} - d(\mathbf{x}))\right)}{\sqrt{(2\pi)^n |\Sigma|}} \quad (10)$$

where n is the number of samples in $\mathbf{y}$ [CRR03].

Estimating the expected value. Each sample point in the leakage distribution $\mathcal{Y}$ is a continuous random variable over $\mathbb{R}$. Hence, the expected value $\mathbb{E}_Y$ is computed as an integral. Instead of computing the integral, we use a Monte Carlo integration method to estimate the expected value. Namely, we draw several random leakage values to estimate the expected value. The ARE is then computed as

$$ARE = \frac{\sum_{Y=\mathbf{y}} \max_{X=\mathbf{x}} \left| \frac{P[Y = \mathbf{y}|X = \mathbf{x}]}{\sum_{X=\mathbf{x}'} P[Y = \mathbf{y}|X = \mathbf{x}']} \cdot \frac{1}{P[X = \mathbf{x}]} - 1 \right|}{k}, \quad (11)$$

and the SD is computed as

$$SD = \frac{\sum_{Y=\mathbf{y}} \sum_{X=\mathbf{x}} \left| \frac{P[Y = \mathbf{y}|X = \mathbf{x}]}{\sum_{X=\mathbf{x}'} P[Y = \mathbf{y}|X = \mathbf{x}']} \cdot \frac{1}{P[X = \mathbf{x}]} - 1 \right|}{2 \cdot k}. \quad (12)$$

In both equations, k denotes the number of leakage vectors $\mathbf{y}$ drawn from the leakage distribution $\mathcal{Y}$. Each leakage vector $\mathbf{y}$ is generated as $\mathbf{y} = d(\mathbf{x}) + \boldsymbol{\varphi}$, where $\mathbf{x}$ is an input generated uniformly at random, $d(\cdot)$ is the deterministic function for an elementary operation, and $\boldsymbol{\varphi}$ is generated from the Gaussian distribution $\mathcal{N}(\mathbf{0}, \Sigma)$ following the noise covariance Σ . The ARE being a worst-case metric, it is more expensive to estimate than the (average-case) SD metric, which is the price to pay for the tighter reductions it enables.

Thanks to the law of large numbers, we know that as k approaches infinity, this estimation converges to the expected value $\mathbb{E}_Y$. We show in Section 4 in our experimental results that the convergence curve reaches a plateau after some number of samples, which corresponds to the value of $\mathbb{E}_Y$.

3.6 Step 6: Compiling the Cryptographic Implementation

At this stage, we have estimated the leakage parameter of each isolated noise-independent elementary operation. As defined in Section 3.5, this parameter can be computed using the SD or ARE metric. We obtain an equivalent leakage probability p in the random probing

model in both cases by applying the reduction. As already mentioned, the reduction is tighter in the case of ARE, where the leakage probability in the random probing model can be directly set to the estimated noisy leakage parameter.

Different vs. maximum leakage probabilities. As we might obtain different noisy leakage parameters δ_i for the different elementary operations, leading to different leakage probabilities in the random probing model, we consider that all the gates leak with the maximum probability corresponding to maximum noisy metric $\delta = \max_i \delta_i$. A random probing secure circuit with maximum leakage probability is straightforwardly random probing secure with different leakage probabilities.

Gate vs. wire leakage model. In our characterization, we rely on the leakage of the elementary operations abstracted as gates in a circuit. Meanwhile, most random probing secure constructions suppose leakage on wires instead. Our methodology applies an additional transition from the gate to the wire leakage model to circumvent this issue. As proven in Lemma 1, we can reduce the security of a circuit in the gate random probing model with leakage probability p , to the wire random probing model with leakage probability $\sqrt{p}$, assuming that each gate has at most two inputs.

Compiling a cryptographic implementation. The final step consists of a two-stage compilation process applied to the input abstract circuit $C \in \mathbb{C}$ representing the target cryptographic implementation.

- One first applies the random probing secure compiler which, for the obtained leakage probability p and the target security level $\varepsilon = 2^{-\lambda}$, transforms C into a functionally equivalent randomized circuit $\hat{C}$ achieving (p, ε) -random probing security.
- One then serializes $\hat{C}$ into a physical implementation, making a sequence of calls to the (whitened) elementary operation routines on the target device. Each elementary operation in the sequence corresponds to a gate in $\hat{C}$ whose output is written in a fresh memory cell. The circuit wiring is hard-coded in the pointer arguments passed to the successive calls to the elementary operation routines.

The obtained physical implementation achieves λ bits of side-channel security for the target side-channel acquisition tool and physical device, and under relaxed or empirically verified physical assumptions. The overall process and provable security guaranty are wrapped up in Section 3.7.

3.7 Security Proof

After describing our methodology, we aim to state our main result.

Theorem 1. *Let C be an abstract circuit in the abstract circuit family $\mathbb{C}$ with $\mathbb{C} = (\mathcal{V}, \mathcal{G})$. Let $\mathcal{D}$ be a target device and $\mathcal{A}$ a target side-channel acquisition tool, for which the six-step methodology described in Section 3 is applied to $\mathcal{D}$ and C . Let us assume that:*

1. *The data isolation effectively ensures that the deterministic signal can be expressed as a sum $\sum_{i=1}^k d_i(a_i, b_i)$;*
2. *The leakage characterization yields the exact leakage distribution (i.e., the exact deterministic functions $\{d_i\}$ and covariance matrix Σ).*

Further, if $\hat{C}$ is a $(\sqrt{\gamma\delta}, \varepsilon)$ -random probing secure circuit functionally equivalent to C where δ is the noisy leakage parameter computed in the fifth step of the methodology and γ is the associated reduction constant, then there exists a simulator $\mathcal{S}$ such that

$$\mathcal{S}(C) \approx_\varepsilon \mathcal{A}(\mathcal{D}, \hat{C})$$

where $\approx_\varepsilon$ statistical distance bounded by ε .

While the ideal data isolation and leakage characterize assumptions might not be perfectly met in practice, they can be naturally relaxed. We thus assume that an adversary cannot effectively exploit the approximation error between this ideal world and the actual leakage to increase the advantage beyond ε .

Proof. Figure 3 gives a specific view of our characterization phases with the leakage assumptions, we next sketch how these fit together to provide the guarantees of Theorem 1. At a high level, our proof builds a sequence of leakage simulators, working backwards w.r.t. the steps of Figure 3: starting from the most abstract leakage model, then each simulator takes the output of the previous simulator in the sequence, and adapts the simulated leakage to the next leakage model, ending with physical leakage.

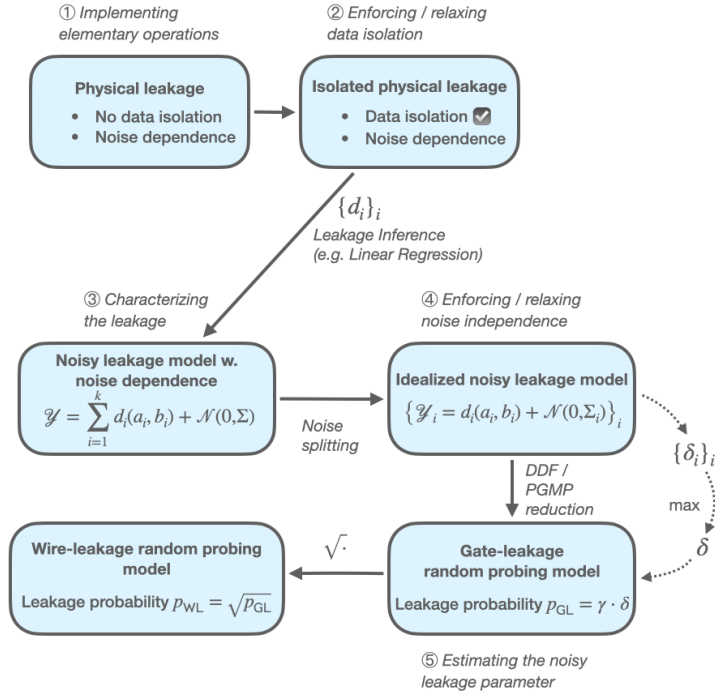


Figure 3: Illustration of the characterization phase with the leakage assumptions

As a starting point, we have $\mathcal{S}_{WL}$, a ε -close simulator for $\hat{C}$ in the $\sqrt{\gamma\delta}$ -wire random probing model. Next, using Lemma 1, we get $\mathcal{S}_{GL}$, a ε -close simulator for $\hat{C}$ in the $\gamma\delta$ -gate random probing model. Moving to the idealized noisy leakage model, we have $\mathcal{S}_{INL}$, a ε -close simulator for the idealized noisy leakage model. When using the SD metric, $\gamma = |\mathcal{X}|$ and $\mathcal{S}_{INL}$ is derived from $\mathcal{S}_{GL}$ using the DDF reduction [DDF14]. Otherwise, we use the ARE with $\gamma = 1$, and the simulator is built by the DDF reduction [PGMP19].

Let us now discuss the (perfect) simulation of the physical leakage from the ideal noisy leakage. The noisy leakage with noise dependence can be perfectly simulated by summing all the components of the ideal noisy leakage, thanks to Equation 7 and the construction of

Σ_i such that $\Sigma = \sum_{i \in [k]} \Sigma_i$. We conclude the proof by remarking that the two hypotheses of the theorem imply that the noisy leakage with noise dependence is equal to the actual physical leakage $\mathcal{A}(\mathcal{D}, \hat{C})$. $\square$

4 Application of the Methodology: Procedures and Case Study

In order to validate our methodology, we conducted tests on an STM32F3 MCU using NewAE's ChipWhisperer-Lite CW1173 board, which is based on an ARM Cortex-M4 processor. We first describe how we implemented elementary operations and our whitening approach. We then introduce and apply a novel test to verify data isolation (Assumption 1, Step 2). Next, we conducted a leakage characterization for the operations, computed the device's noise covariance, and estimated the noisy leakage parameters. We also introduce noise splitting procedures that implement the relaxation of the noise independence (Assumption 2, Step 4). Our findings revealed low noise levels on the device, rendering it unsuitable for masking purposes, thus highlighting the need for secure hardware offering sufficient noise levels to attain robust provable security. Finally, we showcase an application of our methodology on a masked AES implementation. Overall, although the realizations of some steps are basic proof-of-concept versions that require further refinement, when considered alongside our contributions in Section 3, they collectively constitute a practical instantiation of our overall methodology.

4.1 Step 1: Implementing the Operations

We implement operations as routines described in Section 3.1. Since the cost of the data isolation test is quadratic in the number of elementary operations, we limit ourselves to four elementary operations for this proof-of-concept: 8-bit XOR, 8-bit AND, 8-bit Right Shift, and 8-bit Left Shift. These operations would be enough to implement a masked bit-sliced AES at any chosen order for instance. Of course, more elementary operations would make the implementation more efficient but would imply a higher cost in side-channel characterization.

In our example, each elementary operation relates to a single ARM Cortex-M4 instruction, simplifying the analysis (listing the intermediate variables of each operation is trivial) while not mandatory for the methodology. We implement the four operations in assembly as shown in Figure 4.


```

xor_func:
    ldr r0, [r0]
    ldr r1, [r1]
    eor r0, r1, r0
    str r0, [r2]

and_func:
    ldr r0, [r0]
    ldr r1, [r1]
    and r0, r1, r0
    str r0, [r2]

left_shift_func:
    ldr r0, [r0]
    mov r0, r0, LSL 1
    str r0, [r1]

right_shift_func:
    ldr r0, [r0]
    mov r0, r0, LSR 1
    str r0, [r1]

void whitening(void) {
    xor_func(a1Ptr, b1Ptr, c1Ptr);
    xor_func(a2Ptr, b2Ptr, c2Ptr);
    xor_func(a3Ptr, b3Ptr, c3Ptr);
}

```

Figure 4: Elementary operations (xor, and, left shift, right shift) and whitening as implemented on the STM32F3 MCU.

4.2 Step 2: Data Isolation

4.2.1 Procedure for Testing Data Isolation

Masking security proofs require independence between the leakage of all operations. However, enforcing and testing this independence assumption is challenging, leading to another approach in practice based on the test vector leakage assessment (TVLA) [SM16]. This approach verifies the statistical security order (i.e., the smallest statistical moment that leaks) of a masked implementation by detecting secret-dependencies in the statistical moments of the leakage [DFS19]. While dependence in the moment corresponding to the security order is expected due to dependence in inputs of the leakage functions (e.g., all the shares of a value), lower-order moments in a threshold-probing secure implementation with independent leakage functions are independent of the secret. This test can indeed detect typical leakage independence violations due to physical defaults like glitches [MPG05, MPO05] or transitions [CGP⁺12, BGG⁺14] when they lead to a security order reduction. Due to the difficulty in enforcing strict independence in the implementation and to verify it, a commonly accepted relaxation is to ensure that if there are detectable lower-order leakages, they are of significantly lower amplitude than those at the target security order [DFS19].

While this heuristic works reasonably well in practice, it has two significant limitations. First, by verifying only a security order, it cannot detect leakage dependence issues that would result in other kinds of weaknesses than security order reductions, e.g., easing horizontal attacks. Second, while this approach is always applicable in theory, it requires testing all the mixed statistical moments corresponding to all the tuples of leakage points in the traces of a masked implementation [BDF⁺17], and it is therefore computationally impractical (it scales exponentially with the length of the trace) at large security orders (even the second order can be challenging).

Therefore, we propose another approach with much improved practical efficiency, which can detect leakage dependencies that do not reduce the security order. Our approach is based on testing the independence between the leakage of two consecutive operations. Then, we use an argument based on physics to extend the result of this test to long sequences of operations.

Leakage independence for adjacent operations. Let us consider two operations op_1 and op_2 with their respective inputs $\mathbf{x}_1$ and $\mathbf{x}_2$. We assume that these two operations are executed sequentially, giving rise to a leakage trace Y .

We say that the operations have independent leakage if

$$Y(\mathbf{x}_1, \mathbf{x}_2) = d_1(\mathbf{x}_1) + d_2(\mathbf{x}_2) + N \quad (13)$$

where d_1 and d_2 are the deterministic functions (like in [SLP05]) and N follows a Gaussian noise distribution $\mathcal{N}$. This definition indeed ensures independence, as it is possible to decompose N into two independent Gaussian noises N_1 and N_2 , giving $Y = (d_1(\mathbf{x}_1) + N_1) + (d_2(\mathbf{x}_2) + N_2)$. Despite the sequential execution context, we cannot assume that the leakage is a sequential combination of the leakage of the operations (e.g., $Y = (d_1(\mathbf{x}_1) + N_1, d_2(\mathbf{x}_2) + N_2)$) due to data dependency effect between successive operations (e.g. transitions in CPU buses/registers) and low-pass filtering in the measured circuit or acquisition chain.

We use the following statistical test to verify that Equation 13 holds:

1. Generate uniformly at random two pairs α and β of inputs $(\mathbf{x}_1^\alpha, \mathbf{x}_2^\alpha)$ and $(\mathbf{x}_1^\beta, \mathbf{x}_2^\beta)$ (similarly to *fixed-vs-fixed* leakage assessment).
2. Acquire a set $T_{(\alpha,0)}$ of ℓ traces corresponding to executing the operations with the inputs set to $(\mathbf{x}_1^\alpha, \mathbf{0})$ and compute the average trace $\bar{T}_{(\alpha,0)}$.
3. Do the same thing for $T_{(\beta,0)}$ with inputs $(\mathbf{x}_1^\beta, \mathbf{0})$, $T_{(0,\alpha)}$ with inputs $(\mathbf{0}, \mathbf{x}_2^\alpha)$, $T_{(0,\beta)}$ with inputs $(\mathbf{0}, \mathbf{x}_2^\beta)$, $T_{(\alpha,\alpha)}$ with inputs $(\mathbf{x}_1^\alpha, \mathbf{x}_2^\alpha)$ and $T_{(\beta,\beta)}$ with inputs $(\mathbf{x}_1^\beta, \mathbf{x}_2^\beta)$. We hence acquire in total $6 \cdot \ell$ traces.
4. Compute the following

$$T'_{(\alpha,\alpha)} = T_{(\alpha,\alpha)} - \bar{T}_{(\alpha,0)} - \bar{T}_{(0,\alpha)} \quad T'_{(\beta,\beta)} = T_{(\beta,\beta)} - \bar{T}_{(\beta,0)} - \bar{T}_{(0,\beta)},$$

(i.e., from each trace in $T_{(\alpha,\alpha)}$ (resp. $T_{(\beta,\beta)}$), we subtract $\bar{T}_{(\alpha,0)} + \bar{T}_{(0,\alpha)}$ (resp. $\bar{T}_{(\beta,0)} + \bar{T}_{(0,\beta)}$)).

5. Compute the statistical mean equality test on the sets $T'_{(\alpha,\alpha)}$ and $T'_{(\beta,\beta)}$ as

$$t = \frac{\bar{T}'_{(\alpha,\alpha)} - \bar{T}'_{(\beta,\beta)}}{\sqrt{\frac{s_{(\alpha,\alpha)}^2 + s_{(0,\alpha)}^2 + s_{(\alpha,0)}^2 + s_{(\beta,\beta)}^2 + s_{(\beta,0)}^2 + s_{(0,\beta)}^2}{\ell}}} \quad (14)$$

where $s_{(i,j)}^2$ is the unbiased estimator for the population variance of $T_{(i,j)}$. If no significant difference pops up (e.g., $|t| < 4.5^5$), conclude that Equation 13 holds (at least, we could not contradict it).

The motivation for this test is that under the null hypothesis (i.e., Equation 13 holds), $\bar{T}_{(\alpha,0)}$ converges to $d_1(\mathbf{x}_1^\alpha) + d_2(\mathbf{0})$ and $\bar{T}_{(0,\alpha)}$ to $d_1(\mathbf{0}) + d_2(\mathbf{x}_2^\alpha)$. Therefore, the distribution of $T'_{(\alpha,\alpha)}$ converges to the distribution of $N + d_1(\mathbf{0}, \mathbf{0}) + d_2(\mathbf{0}, \mathbf{0})$, and likewise for $T'_{(\beta,\beta)}$. We finally compute t such that, under the null hypothesis, it follows a standard normal distribution.

Since the noise comes from physical, electronic phenomena, its Gaussian distribution and independence on the data is a reasonable assumption. However, in case of doubt, further statistical tests can be performed. For instance, a test of Gaussianity of $Y(\mathbf{x}_1, \mathbf{x}_2)$ can be performed for fixed $(\mathbf{x}_1, \mathbf{x}_2)$, as well as an equality test for the (co)variance of $Y(\mathbf{x}_1, \mathbf{x}_2)$ across different $(\mathbf{x}_1, \mathbf{x}_2)$.

Finally, it is worth clarifying that a statistical test can demonstrate the inability to detect dependencies with a specific number of measurements but does not prove independence. Nonetheless, if the test fails to identify a dependency for ℓ traces, it reasonably suggests

⁵The 4.5 threshold is given for simplicity, a better approach would be to adapt the t -score threshold with respect to the length of the traces (e.g. as proposed in [DZD⁺17]).

that this dependency is unlikely to be exploited for an attack with significantly fewer than ℓ traces. In contrast to traditional higher-order TVLA, our test offers greater statistical power: being a first-order test, it exhibits lower sensitivity to noise, regardless of the masking order under consideration.

Independence in longer operation sequences. Let us now discuss the dependencies between operations that are not adjacent. In this section, we argue that, given an understanding of the evaluated processor’s architecture and within reasonable physical assumptions, the absence of dependency between adjacent operations ensures the independence of leakage for non-adjacent operations.

Considering the processor (excluding the memory), we first assume that the “core” leakage for any clock cycle is a function of all the state of the processor (and the input data, e.g., on the memory bus).⁶ This “core” leakage may then get filtered (i.e., undergo a linear transformation) before it is measured (*linear physics* hypothesis, denoted LP). Next, given a sufficiently simple processor, we may assume that when the processor executes m well-chosen non-branching/conditional instructions, the microarchitectural state of the processor does not depend on the state computed by the first of these operations (provided that the other operations do not also compute this state, and a few cycles after the last instruction retires) — *m-state-erasing* (denoted *m-SE*) hypothesis. Concretely, for an elementary processor whose state is only the architectural state, the 2-SE hypothesis is satisfied. For more complex processors, m might be larger (or even not exist). For a simple in-order processor, *m-SE* with m close to the pipeline depth appears as a reasonable assumption — this assumption can easily be verified if the design of the processor is known.

Our operations all follow the same structure: load the operands in two registers, perform a logic instruction, and store the result (always using the same registers). Then, between two operations, we clear the register state, then we execute a sequence of “cleaning” operations that operate on constant public data (i.e., whitening operations). The *m-SE* hypothesis, combined with LP, implies independent leakage when $m - 1$ cleanings separate the operations. Our two operations test presented in the next section is a way to validate the hypotheses (and the m parameter).

Finally, regarding the memory leakage, it is reasonable to assume LP for the static leakage from the memory cells and *m-SE* for the remaining logic.

Let us conclude this section by remarking that if an open-source processor is used, the analysis of leakage independence is greatly simplified. Indeed, as the hardware is known, we may apply the robust-probing leakage model to instructions sequences (or to verify the *m-SC* hypothesis). Providing hardware support for micro-architectural state “cleaning” through a dedicated fence instruction may also solution to improve code simplicity and performance, with very low hardware cost, in the spirit of masking-supporting instruction set extensions [GMPP20]. The software-level overhead can be further reduced by integrating the isolation semantics inside the data-processing instructions themselves, instead of needing an explicit fence. For example, [CPW24] proposes to add instruction prefixes or processor control registers leading to semantics “preventing any architectural and micro-architectural overwriting”.

4.2.2 Data Isolation on the STM32F3

To perform our data isolation test, we need to capture the side-channel execution traces of two consecutive elementary operations separated by a whitening process and use the test to validate or not data isolation between the two operations. This approach must be re-iterated for all combinations of two successive elementary operations.

⁶This assumption is conservative since it is weaker than the well-studied robust probing model for hardware leakage [FGM⁺18].

The whitening process does not have to be the same for all pairs of elementary operations, but for our operations selection, the acquisition setup, and the chip, a single whitening process allows us to pass all tests: three consecutive **xor** operations with constant public inputs⁷. The corresponding assembly code is shown in Figure 4, where $\{\text{aiPtr}, \text{biPtr}, \text{ciPtr}\}$ for $i \in \{1, 2, 3\}$ are memory pointers to the two constant operands $\{\text{ai}, \text{bi}\}$ and the memory location to store the result ci .

The test procedure is applied as follows:

- For each pair of elementary operations, the target code is the following sequence of calls:

```
whitening();
operation1(aiPtr, biPtr, ciPtr);
whitening();
operation2(a2Ptr, b2Ptr, c2Ptr);
```

The inputs are pre-generated and loaded in memory once for all before iterating on the target code.

- Triggers surround the target code to ease the trace acquisition.
- For randomly chosen values $(\mathbf{x}_1^\alpha, \mathbf{x}_2^\alpha), (\mathbf{x}_1^\beta, \mathbf{x}_2^\beta)$ ⁸ we collect 10^6 traces for each set $T_{(\alpha, \alpha)}, T_{(\alpha, 0)}, T_{(0, \alpha)}, T_{(\beta, \beta)}, T_{(\beta, 0)}, T_{(0, \beta)}$, for a total of 6×10^6 traces.
- From the sets of traces, we test if Equation (13) holds.

The above process is applied for each pair of elementary operations and iterated 2-3 times for different values of $(\mathbf{x}_1^\alpha, \mathbf{x}_2^\alpha), (\mathbf{x}_1^\beta, \mathbf{x}_2^\beta)$. Figure 5 illustrates the test result for the pair of operations (**xor**, **and**) and a single choice of fixed inputs. Figure 5b (resp. Figure 5c) illustrates the captured leakage (through the T-Test) of the computation of **xor** (resp. **and**) and the manipulation of its inputs/outputs. Namely, Figure 5b shows high T-test values at the moment of the execution of **xor** with a residual leakage slowly decreasing afterward. While Figure 5c shows high T-test values later at the moment of the execution **and**, and no leakage is detected before, ensuring that the inputs of **and** were not manipulated before the execution of the operation. Then, Figure 5d illustrates the captured leakage of both **xor** and **and** simultaneously, including the individual leakages of **xor** and **and**. Figure 5e is the result of our proposed test: it represents the captured leakage of both **xor** and **and** simultaneously while individual leakages of **xor** and **and** are removed. The T-Test results show that the data isolation process (whitening function) successfully removes the combined leakage of **xor** and **and**.

4.3 Step 3: Linear Regression as the Leakage Function

We use the experimental setting described in Section 4.2, where we consider operations with at most 2 inputs of 8 bits. We start by inferring each elementary operation's deterministic part of the leakage function separately. We use the linear regression with a specific choice of basis of functions $\mathcal{H} = \{h_1, \dots, h_m\}$. The result of the linear regression is the set of $\{\alpha_i\}_i$ such that, for all inputs (a, b) of the selected elementary operation,

$$d(a, b) = \sum_{i=1}^m \alpha_i h_i(a, b) \quad (15)$$

holds. In order to capture the leakage function fully, we construct the basis of functions as follows (where n is the bit-length of the inputs a and b , here $n = 8$):

⁷Using only two consecutive **xor** operations was not enough to pass all tests for all pairs of elementary operations.

⁸where $\mathbf{x}_i$ (resp. $\mathbf{x}'_i$) contains the two inputs of **operation** i

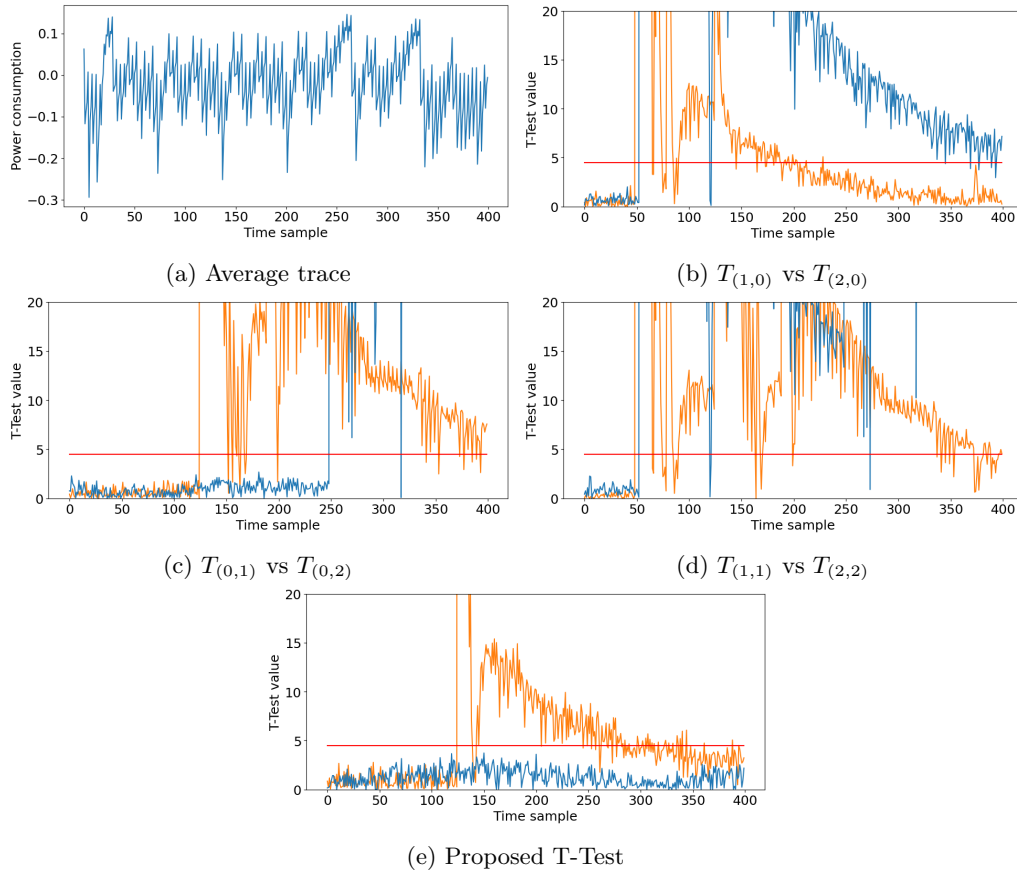


Figure 5: Data isolation test: curves with whitening in blue and without whitening in orange.

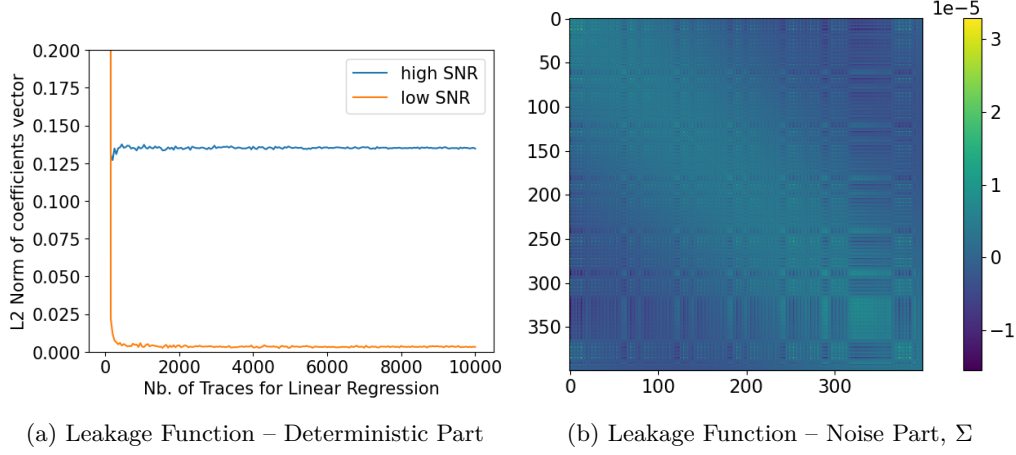


Figure 6: Linear Regression of the Leakage Function.

- $h_0(a, b) = 1$
- for all $i \in [1 \cdots n]$, $h_i(a, b)$ returns the i th bit of a
- for all $i \in [1 \cdots n]$, $h_{n+i}(a, b)$ returns the i th bit of b
- for all $(i, j) \in [1 \cdots 2n]^2$, with $i < j$, $h_{i,j}(a, b)$ returns $h_i(a, b) \oplus h_j(a, b)$

This gives a total of $m = 1 + 2 \cdot n + n \cdot (2 \cdot n - 1) = 2 \cdot n^2 + n + 1$ functions in the function basis $\mathcal{H}$.

For each elementary operation, the target code is the following concatenation

```
operation(aPtr, bPtr, cPtr);
whitening();
```

surrounded by triggers to ease the trace acquisition (similarly to [Subsubsection 4.2.2](#)). The inputs to the operations are randomly generated (using a Mersenne Twister RNG) on the chip before each execution of the target code. Then, we collect 10^6 traces in a single set and apply linear regression over the function basis $\mathcal{H}$ on each independent time sample. The output is the set of vectors $\{\alpha_i\}_i$.

[Figure 6a](#) illustrates the convergence of the L2 norm of the coefficient vectors at two different time samples for the `xor` operation. We consider one vector where the SNR ratio is high (i.e., more information leakage) and one where the ratio is low. We can see that for both vectors, the L2 norm converges from a few hundred traces, meaning that the linear regression can quickly estimate the coefficients of the deterministic part of the function. This behavior can further be explained by the low noise in the leakage depicted through the covariance matrix in [Figure 6b](#). We compute this covariance matrix on the same time samples of the operation as for the deterministic function. The covariance matrix clearly shows low noise levels, which implies more information leakage. We can also observe through [Figure 6a](#) that for a sample with high SNR, the coefficients converge to more significant values than for a sample with low SNR, which gives more confidence about the results, since for samples with more information leakage, the deterministic functions should have more weight than when there is not much information leakage.

4.4 Step 4: Implementing Noise Splitting

4.4.1 Noise Splitting Procedures

A necessary physical assumption in the noisy leakage model is noise independence as described in [Assumption 2](#). Since enforcing this assumption is hard to achieve, we propose a way to relax it instead, as discussed in [Section 3.4](#).

Consider a sequence of k operations and the corresponding leakage trace Y . Assuming that the noise is additive, we have the decomposition $Y = S + N$ where S is the signal (typically a deterministic function of the input data as presented in the previous sections), and N is the data-independent noise. Further, thanks to the data isolation test of the previous section, we know that the signal can be rewritten as $S = \sum_{i=1}^k S_i$, where S_i is the leakage caused by operation i . We aim to decompose the noise into a sum of $k + 1$ independent contributions (one for each operation and a “leftover” one) $N = \sum_{i=0}^k N_i$. Assuming that the noise contributions N_i are Gaussian, we only have to ensure that they are not correlated to ensure independence. This gives us a decomposition of the leakage signal $Y = \sum_{i=0}^k Y_i$ where $Y_0 = N_0$ and $Y_i = S_i + N_i$ for $i \neq 0$, which ensures signal and noise independence between the components.

We argue that an implementation secure against an adversary with access to $\{Y_i\}_{i \in \{0, \dots, k\}}$ is also secure against an adversary with access to the global leakage trace Y . The former is stronger than the latter adversary since each sample in Y can be obtained by the sum of the corresponding samples in $\{Y_i\}_{i \in \{0, \dots, k\}}$. In our methodology, we suggest computing the noisy leakage parameter based on the split leakages Y_i . This approach results in a weaker noisy leakage parameter compared to that computed from Y , as the noise N is divided into $k + 1$ smaller components. Consequently, each independent operation inherently leaks more information. Hence, an abstract circuit secure against an adversary with access to the split leakage is also secure against an adversary with access to the global leakage Y .

We will present two solutions to the above decomposition problem in the following. We first discuss a trivial solution, which has the advantage of being easily applicable but induces a loss in the security level as the size of the implementation grows. Then, we express the decomposition as an optimization problem that better scales with the size of the circuit but is more challenging to solve. We propose a direct solution to the optimization and leave the question of optimally solving it as an open problem.

Trivial Solution. We can perform a trivial split of the noise described above. Namely, for a sequence of k operations, we can split the Gaussian noise $N = \mathcal{N}(\mathbf{0}, \Sigma)$ such that $N_0 = \mathbf{0}$ and $N_i = \mathcal{N}(\mathbf{0}, (1/k) \cdot \Sigma)$ for $i \in \{1, \dots, k\}$. This decomposition ensures that the leakage Y can be expressed as a sum of Y_i with $Y_0 = \mathbf{0}$ and $Y_i = S_i + \mathcal{N}(\mathbf{0}, (1/k) \cdot \Sigma)$, with noise and signal independence.

Meanwhile, the above decomposition scales poorly with the size of the circuit. As the number of operations grows, the noise occurring on each operation decreases, leading to lower security in the noisy leakage model when applying the reduction after the relaxation (i.e., increasing the ARE leakage).

Hence, the noise decomposition, in addition to ensuring the independence of the components, should minimize this leakage. As explained in [Section 3.6](#), the chosen ARE (or SD) metric for the noisy leakage model is the maximum among all operations executed. Hence, the chosen decomposition should balance the noise on all operations and scale with the number of operations executed.

Better Noise Splitting. We propose a better way to split the noise taking advantage of a relaxed noise independence assumption. For a sequence of k elementary operations, we can split the leakage trace into k sub-traces, all of the same size and including the time samples of one elementary operation each. We call distance d between two sub-traces the number of operations (or sub-traces) between them during the computation’s sequence (for example, two consecutive sub-traces have distance $d = 1$). For the sake of simplicity, we assume that all the sub-traces have identical noise distributions and that the dependence (i.e., the covariance matrix) between the noises of two sub-traces solely depend on their

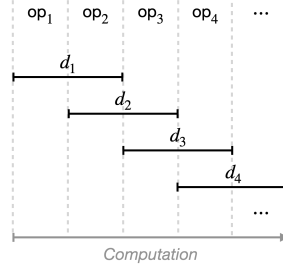


Figure 7: Data dependency spanning.

distance d .⁹ This means that each operation's noise covariance matrix is the same denoted Σ'_0 , and that the covariance matrix between two sub-traces with distance d is the same along the computation denoted Σ'_d . We then formulate the following relaxed noise-independence assumption.

Assumption 3 (Relaxed noise independence assumption). *There exists $d_{max} \in [0, k)$ such that the sub-traces with a distance $d > d_{max}$ have null covariance: $\Sigma'_d = 0 \ \forall d > d_{max}$.*

Intuitively, the above assumption captures the expectation that, after some delay, the noise in an operation sub-trace is fully independent of the noise in an earlier operation sub-trace. While we introduce it as a “relaxed assumption”, we stress that it is without loss of generality since there always exists such a d_{max} . In particular, the case $d_{max} = k - 1$ captures that the independence between the noise of two operations is never reached.

Under this relaxed noise independence assumption, the global covariance matrix for k operations has the following structure (assuming $d_{max} = 1$):

$$\Sigma = \begin{pmatrix} \Sigma'_0 & \Sigma'_1 & & & \\ \Sigma'_1 & \Sigma'_0 & \Sigma'_1 & & \\ & \Sigma'_1 & \Sigma'_0 & \Sigma'_1 & \\ & & & \ddots & \ddots & \ddots \end{pmatrix} \quad (16)$$

We introduce another parameter, which we call the *data-dependency depth*, ℓ_{max} . This is the number of sub-traces over which the data dependency of an elementary operation spans. Specifically, the deterministic part of the leakage d_i of the i -th operation is non-zero for samples spanning on sub-traces $i, i + 1, \dots, i + \ell_{max}$. This is represented in Figure 7 for $\ell_{max} = 1$.

We now explain how a better splitting of the noise can be achieved, first by assuming $d_{max} = \ell_{max} = 1$ (and generalize later). Consider a split of the leakage in three as:

$$\begin{cases} L_1 := (S_1 + S_4 + \dots) + N_1 \\ L_2 := (S_2 + S_5 + \dots) + N_2 \\ L_3 := (S_3 + S_6 + \dots) + N_3 \end{cases}$$

where $S_i = d_i(\mathbf{x}_i)$ denotes the signal of the i -th operation, which spans over time samples as represented on Figure 7, and with $N_i \sim \mathcal{N}(0, (1/3)\Sigma)$ so that $N_1 + N_2 + N_3 = N \sim \mathcal{N}(0, \Sigma)$. We have that $(L_1 + L_2 + L_3) \sim Y$, the global leakage. Let us now consider $(1/3)\Sigma = AA^T$ the Cholesky decomposition of the global covariance matrix (scaled by $1/3$), so that the

⁹This assumption is not strictly necessary to the application of our method but makes the presentation much simpler.

N_i noises follow a distribution $N_i \sim A \cdot X_i$ with $X_i \sim \mathcal{N}(0, I)$, for I the identity matrix. We have that A^{-1} has the same zero matrix blocks as Σ (see Equation 16). Namely, it can be written as:

$$A^{-1} = \begin{pmatrix} B_0 & B_1^T & & & \\ B_1 & B_0 & B_1^T & & \\ & B_1 & B_0 & B_1^T & \\ & & \ddots & \ddots & \ddots \end{pmatrix} \quad (17)$$

for some matrices B_0, B_1 (with B_0 being symmetric). Then we get

$$\begin{aligned} A^{-1} \cdot L_1 &:= (S'_1 + S'_4 + \dots) + X_1 \\ A^{-1} \cdot L_2 &:= (S'_2 + S'_5 + \dots) + X_2 \\ A^{-1} \cdot L_3 &:= (S'_3 + S'_6 + \dots) + X_3 \end{aligned} \quad (18)$$

with $S'_i = A^{-1} \cdot S_i$. One can then check that for each of the three leakages, L_1, L_2 , and L_3 , the successive signals $S'_i, S'_{i+3}, \dots$ are strictly disjoint (meaning that they are non-zero over disjoint time samples). This is due to the structure of A^{-1} (see Equation 17) and the fact that each S_i spans over two sub-traces. Then the S'_i span over three sub-traces so that S'_i and S'_{i+3} are disjoint. Moreover, the normalized noises $A^{-1} \cdot N_i = X_i \sim \mathcal{N}(0, I)$ can be trivially separated as

$$\begin{aligned} X_1 &= X'_1 + X'_4 + \dots \\ X_2 &= X'_2 + X'_5 + \dots \\ X_3 &= X'_3 + X'_6 + \dots \end{aligned} \quad (19)$$

such that the X'_i span the same time samples as the S'_i . We finally split the leakage in variables $Y_i = A \cdot (S'_i + X'_i)$ which satisfy $Y = \sum_{i=1}^k Y_i$. In this splitted leakage, the amount of noise is scaled by a factor $1/3$ compared to the factor $1/k$ of the trivial solution.

The same reasoning applies to higher values of d_{max} and ℓ_{max} . But instead of dividing the noise in 3 and scaling the covariance by factor $1/3$, one has to divide it in $d_{max} + \ell_{max} + 1$ and hence scale the covariance by a factor $1/(d_{max} + \ell_{max} + 1)$. Depending on the noise dependency depth and data dependency depth, this might still be way better than a factor $1/k$.

Towards Optimal Noise Splitting. While better than the trivial solution, the above method is non-optimal since it roughly splits the noise in $d_{max} + \ell_{max} + 1$ regardless of the signals S_i . While the signal S_i may span over the $(i + 1)$ -th sub-trace, it might be much weaker than on the i -th sub-trace and should receive a smaller amount of noise than the signal S_{i+1} on these time samples.

Once again, we state our optimization problem for $d_{max} = \ell_{max} = 1$ but stress that it can be generalized to higher depths. Recall that we want to split the global covariance matrix into $k + 1$ covariance matrices $\Sigma_0, \dots, \Sigma_k$ such that

$$\Sigma = \Sigma_0 + \Sigma_1 + \dots + \Sigma_k \quad (20)$$

to split the leakage into n leakages: $Y_i := S_i + N_i$ with $N_i \sim \mathcal{N}(0, \Sigma_i)$.

Given the data-dependency spanning (c.f. Figure 7), Σ_i is only required to span the same leakage samples as d_i . Then the (lowered) global covariance matrix Σ has the following structure:

$$\begin{pmatrix} \Sigma_1 & & & & \\ & \Sigma_2 & & & \\ & & \Sigma_3 & & \\ & & & \Sigma_4 & \\ & & & & \ddots \end{pmatrix}$$

From this structure, we observe that for an operation, say the i -th one, we need to split the covariance matrix Σ'_0 between Y_i and Y_{i-1} (since d_{i-1} spans over time samples of the i -th operation). On the other hand, Σ'_1 does not need to be split. Namely, defining Σ_i , for every $i \in \{1, \dots, n\}$, as the symmetric positive semi-definite matrix

$$\Sigma_i := \begin{pmatrix} \bar{\Sigma}_0^{(0)} & \bar{\Sigma}_1 \\ \bar{\Sigma}_1 & \bar{\Sigma}_0^{(1)} \end{pmatrix} \quad (21)$$

with

$$\bar{\Sigma}_0^{(0)} + \bar{\Sigma}_0^{(1)} \leq \Sigma'_0 \quad \text{and} \quad \bar{\Sigma}_1 \leq \Sigma'_1, \quad (22)$$

we ensure Equation 20, where by $\Sigma' \leq \Sigma$ we mean that there exists a positive semi-definite matrix Σ'' (i.e., Σ'' is a covariance matrix) such that $\Sigma' + \Sigma'' = \Sigma$.

Let δ_i be a leakage metric corresponding to Y_i . Our optimization goal is

$$\min_{\bar{\Sigma}_0^{(0)}, \bar{\Sigma}_0^{(1)}, \bar{\Sigma}_1} \max \{\delta_i\}_i$$

under the constraints of Equation 22, and for Σ_i symmetric and positive semi-definite.

To sum up, under the assumptions stated above, we infer the leakage parameters which are the functions d_i , and the covariance matrices Σ'_0 and Σ'_1 and we look for a matrix Σ_i as defined in Equation 21 (in particular, a split of the Σ'_0 matrix into $\bar{\Sigma}_0^{(0)} + \bar{\Sigma}_0^{(1)}$) for which the maximal δ_i is minimized.

Choosing δ_i . Ideally, we would find the decomposition as the one that minimizes the SD or ARE leakage metric. Meanwhile, choosing metrics simpler to express can lead to optimization problems with simpler constraints, theoretically and efficiently solvable with current tools. For instance, we can choose our metric to be the multivariate SNR denoted SNR_i for the leakage Y_i , defined as the maximal eigenvalue of the matrix $\Sigma_{d_i} \tilde{\Sigma}_i^{-1}$, where Σ_{d_i} is the covariance matrix of $d_i(X)$, for X uniform over $\mathcal{X}$. Then, our optimization goal becomes

$$\min_{\bar{\Sigma}_0^{(0)}, \bar{\Sigma}_0^{(1)}, \bar{\Sigma}_1} \max \{\text{SNR}_i\}_i$$

under the same constraints as earlier, which leads to a convex optimization problem. Minimizing the SNR ultimately leads to lowering the SD or ARE and therefore appears as a natural first step before solving the more general case. It is in line with our goal to exhibit a first complete connection between the theory and practice of the masking countermeasure, leaving the question of an optimized methodology relying on the best combination of metrics and proofs as an interesting direction for further research.

4.4.2 Noise Splitting in Practice

Our experimental results on the ChipWhisperer show that the noise levels on the ChipWhisperer we use are very low. Namely, Figure 6 shows deterministic leakage of variance about 1×10^{-1} while the highest noise variance is about 3×10^{-5} , hence our measurements are essentially noise-free.

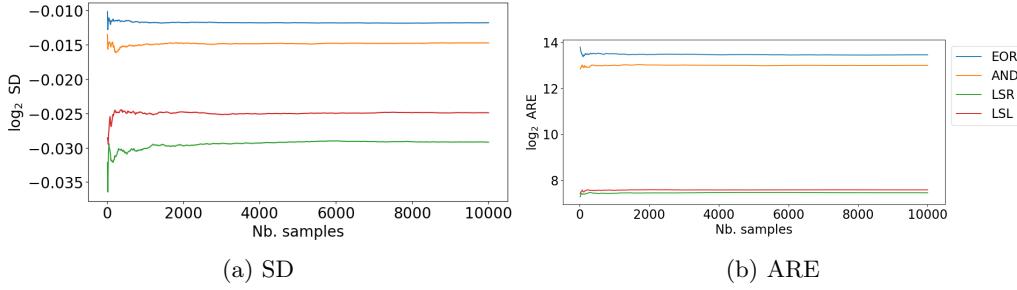


Figure 8: ARE and SD Monte-Carlo convergence as described in Section 3.5.

We nevertheless used the noise covariance matrix Figure 6b (computed from a set of traces with fixed input value for the same operation) to apply the relaxation from the previous section. The very low noise failed our attempts to apply the optimization problem of Section 4.4.1. In this case, we can apply the trivial noise decomposition from Section 4.4, making the security reduction work at the cost of decreasing the noise levels even further as the size of the circuit grows.

4.5 Step 5: Estimating the Noisy Leakage Parameter

Still with the same experimental setting as Subsubsection 4.2.2, we consider operations with at most 2 inputs of 8 bits. Then, iterating through all possible values in $\mathcal{X}$ for a given $\mathbf{y}$ amounts to performing 2^{16} iterations. An extra 2^{16} iterations are performed to compute the max for ARE or the sum for SD. Indeed, iterating over all possible values in $\mathcal{X}$ does not scale well when considering larger inputs. In such a case, other methods can be applied to make the computation tractable. For instance, one can use the nearest-neighbor-based approach from [CDSU23] to efficiently and quickly compute the conditional probabilities and to find the max over $\mathbf{x} \in \mathcal{X}$ in the case of ARE. We leave the computation of the noisiness metric for larger inputs as an open research question.

To simplify our analysis, and since we already observe inefficient noise levels on the ChipWhisperer, we estimate the ARE and SD metrics using the original covariance matrix from Figure 6b to exhibit the noisiness levels achievable on this device in the best cases. We end our estimations by discussing the challenge of designing hardware that generates enough noise to implement circuits secure in the noisy leakage mode with reasonable security levels and finding optimal ways to solve the relaxation on such a device.

Figure 8 shows the convergence of the Monte Carlo estimation of ARE and SD metrics for the four operations as considered in Section 4.2. The curves show that both metrics converge to a stable value after around 4000 samples for each operation. For the ARE metric, it converges to a maximal value of $\approx 2^{13.4}$ for the `xor` operation, which is enormous as this value is the same as the leakage probability in the random probing model (c.f. Section 2). Recall that the final ARE value in the noisy leakage model is the maximal ARE among all operations. This result means no constructions secure in the random probing model can be used on this device. We compare this value to the SD metric, which converges to a maximal value of $\approx 2^{-0.0093}$ for the `xor` operation. While this value is much lower, we recall that the reduction to the random probing model with the SD metric (c.f. Section 2) induces a factor of 2^{16} , equal to the size of the input space (2 inputs of 8 bits each). In other words, the leakage probability in the random probing model using the SD reduction would be almost 2^{16} , which is even higher than with the ARE reduction. We observe that the values of the SD and ARE metrics are smaller for the shift operations than for the `xor` and `and` operations. We argue that this comes from the fact that the `xor` and `and` operations have two operands of 8 bits and perform an additional instruction between

registers, contrary to the shift operations, leading to more information leakage and hence higher values for the noisiness metrics.

Such values for the ARE and SD noisiness metrics imply critical leakage levels on this component, making attacks most likely possible with very few traces. It also matches the conclusions of previous works (e.g. [CDSU23]) on this component. In addition, such levels of noisiness metrics make it challenging to have provably secure implementations on the device. To show the amount of noise that needs to be added to this component to be able to use secure constructions from the literature, we present in the following section concrete results on the AES cipher and use artificial noise that we add to the samples to demonstrate the obtained security levels.

4.6 Step 6: Applying a Random Probing Secure Compiler

To achieve arbitrary levels of security in the random probing model, current literature proposes using an *expanding compiler* [BCP⁺20, BRT21, BRTV21]. We recall that the latter consists of recursively applying some base gadgets to the original circuit until achieving the desired security level. After k applications, the achieved random probing security is $\varepsilon = f^k(p_{\text{WL}})$ where p_{WL} is the random probing wire-leakage probability and f is the simulation failure function achieved by the set of gadgets. The maximal tolerated leakage probability for current 3-share and 5-share constructions is around $p_{\text{WL}} \approx 2^{-7.5}$ [BRT21]. In our context, p_{WL} is the square root of the maximal ARE metric over the different operations, meaning that the maximum tolerated ARE is of $ARE \approx 2^{-15}$. This value is very far from what we estimate in Section 4.5 on the ChipWhisperer.

4.6.1 Adding artificial noise and impact on ARE / SD

We illustrate the impact of noise on security in the noisy leakage model by adding artificial noise to the traces acquired with the ChipWhisperer. For simplicity, we add noise to each time sample of each trace, drawn from a univariate Gaussian distribution of mean 0 and standard deviation σ . We illustrate the evaluation of the ARE value for the `xor` operation, which showed the highest ARE and SD values in Section 4.5. This operation’s signal variance is about $\sigma_{\text{signal}}^2 \approx 10^{-5}$ at the leakiest point during the execution of the operation. Table 1 shows the values of convergence for the SD and ARE metrics as done in Section 4.5, after adding different amounts of noise to the traces (i.e., different σ values). The table shows that the ARE value reaches 2^{-7} when adding a univariate Gaussian noise of mean 0 and standard deviation $\sigma = 5$ to each sample in the traces. Recall that this corresponds to a leakage probability of $2^{-3.5}$ in the random probing model. Meanwhile, the SD value reaches 2^{-10} , which must then be multiplied by 2^{16} to obtain the leakage probability in the random probing model (because we consider 2-input 8-bit `xor` operation), making the reduction still not usable. These results showcase the difference in the tightness of the reduction from the noisy leakage to the random probing model using the SD and ARE metrics on this device. We also recall that the reduction using the ARE metric is theoretically tighter (c.f. Section 2) because the latter is a worst-case metric, matching the definition of the random probing model, a worst-case model. We then remark that the values of the ARE and SD metrics decrease as the σ value increases by the same factor. For instance, the ARE and SD values are halved whenever the σ is doubled. In order to use random probing secure gadgets from the literature, as mentioned above, we need to tolerate a leakage probability of almost $2^{-7.5}$, translating to an ARE value of 2^{-15} . This value is reached when adding Gaussian noise with a significant standard deviation $\sigma \approx 1280$.

Table 1: ARE and SD values after adding noise to the leakage traces on the ChipWhisperer.

σ	ARE	SD
5	2^{-7}	2^{-10}
10	2^{-8}	2^{-11}
20	2^{-9}	2^{-12}
40	2^{-10}	2^{-13}
1280	2^{-15}	2^{-18}

Table 2: AES operations complexity.

AES Operation	Complexity (N_{xor} , N_{ls1} , N_{copy} , N_{and} , N_{rnd})
AddRoundKey	(16, 0, 0, 0, 0)
SubBytes	(174, 0, 111, 64, 0)
Linear layer	(54, 16, 46, 0, 0)
AES-128 (10 rounds)	(2440, 160, 1570, 640, 0)

4.6.2 Application to AES

We now illustrate the impact of the implementation’s noise level on the complexity of the expanding compiler in the random probing model. We choose a provably secure implementation of AES as in [BCP⁺20], under the verified and relaxed leakage assumptions. We consider a bitslice implementation of AES using the 8-bit bitwise operations (**xor**, **and**, **left shift logical**). Table 2 gives the operation counts for such an implementation. The **copy** operation outputs two values equal to the single input value, and the **rnd** operation outputs a fresh uniform random value. N_g denotes the number of operations for the operation g in the circuit.

For the s-box, we use the optimized Boolean circuit from [BMP13]. This circuit computes the AES s-box with 32 ANDs, 83 XORs and 4 NOTs. Moreover, it involves 111 copies. In our context, NOTs are computed as XORs with a constant operand, which makes 32 ANDs + 87 XORs + 111 copies. For a full SubBytes layer, composed of 16 bitsliced s-boxes, this makes 32 ANDs + 87 XORs + 111 copies in terms of 16-bit operations, which is 64 ANDs + 174 XORs + 222 copies in terms of 8-bit operations.

For the linear layer, we rely on the *fixslicing approach* proposed in [AP21]. For the linear layer in one round, this approach requires 27 32-bit XORs, 16 word-wise rotations, 16 byte-wise rotations and 23 copies. In our context, word-wise rotations are free since they are by multiples of 8. A byte-wise rotation requires 2 LSHs (one left shift, one right shift). This makes a total of 108 XORs + 32 LSH + 92 copies in terms of 8-bit operations for the MixColumn layer for two blocks, which is 54 XORs + 16 LSH + 46 copies per block.¹⁰

We apply the expanding compiler proposed in [BCP⁺20] with the 3-share gadgets proposed in [BRT21]. The LSL gadget applies the LSL operation to each input share before refreshing the sharing using a refresh gadget. Table 3 summarizes the complexities of these gadgets. As for the failure functions for the set of gadgets, we compute them using the verification tool IronMask [BMRT22].

The operation counts after k applications of the expanding compiler is given by $\mathbf{N}^k \cdot \vec{N}_{\text{AES}}$

¹⁰We note that this is for the even rounds, while odd rounds are further optimized in [AP21] but we consider the same count for all the rounds.

Table 3: Complexities for the 3-share gadgets from [BRT21] achieving $(t = 1, f)$ -RPE.

Gadget	Complexity
	$(N_{\text{XOR}}, N_{\text{LSH}}, N_{\text{copy}}, N_{\text{AND}}, N_{\text{RND}})$
G_{refresh}	$(4, 0, 2, 0, 2)$
G_{XOR}	$(11, 0, 4, 0, 4)$
G_{LSL}	$(4, 3, 2, 0, 2)$
G_{copy}	$(8, 0, 7, 0, 4)$
G_{AND}	$(40, 0, 29, 9, 17)$

Table 4: Masked AES for different levels of expansion.

Expansion level k	Masked AES Complexity	ARE for $2^{-\lambda}$ security		
		$\lambda = 32$	$\lambda = 64$	$\lambda = 128$
1	0.24 Mop	2^{-40}	2^{-72}	2^{-136}
2	6.14 Mop	2^{-28}	2^{-44}	2^{-76}
3	163 Mop	2^{-22}	2^{-30}	2^{-46}
4	4.33 Gop	2^{-18}	2^{-22}	2^{-30}

(c.f. [BCP⁺20]) where $\mathbf{N}$ is the gadget gate-count matrix defined as

$$\mathbf{N} = (\vec{N}_{\text{XOR}} \mid \vec{N}_{\text{LSH}} \mid \vec{N}_{\text{copy}} \mid \vec{N}_{\text{AND}} \mid \vec{N}_{\text{RND}}), \quad (23)$$

and $\vec{N}_{\text{AES}}$ is the gate-count vector for AES given by the last row of Table 2.

Table 4 summarizes the complexities of the obtained masked AES with expansion levels $k \in \{1, 2, 3, 4\}$. For each expansion level, it further gives the maximal value of the ARE in order to reach a provable security of $\varepsilon = 2^{-\lambda}$, for $\lambda \in \{32, 64, 128\}$. In order to compute the ARE value, we use the failure functions computed with `IronMask` and numerically estimate the required leakage probability p to achieve the security level given the expansion level k . This translates to the required noise level (or ARE) on the physical device to achieve the proven security level. We also recall that the ARE value is then obtained as p^2 .

Table 4 shows that by doing one level of expansion, which consists of replacing each gate of the circuit with the corresponding gadget, the required levels of ARE are very high and reach 2^{-136} for 128-bit security. As we further apply the expansion, this required level decreases to almost 2^{-30} for 128-bit security, but the complexity of the circuit becomes quickly impractical (4.33×10^9 operations). Meanwhile, to have an ARE value of 2^{-30} on the chip we use for our tests, for example, huge amounts of noise must be added ($\sigma \approx 5 \cdot 2^{23}$, c.f. Table 1).

Our results emphasize the trade-off between the physical noise on a device and the complexity of circuit implementation on this device with proven security. Higher noise levels lead to less complex constructions while achieving such noise requires specialized hardware that enables considerable noise independent of the operations. This also emphasizes the challenge of constructing such hardware, taking provable security into account and the limitations of the noise levels that can be achieved in practice. We recall that the `ChipWhisperer` we use is far from suitable for such a case, and the question of having adapted hardware needs to be studied further in the literature.

While the complexities obtained through our results are yet to be practical, they show that it is possible to obtain physical implementations with provable security and that the noisy leakage security of the considered device highly influences the complexity of the constructions.

5 Discussions and Perspectives

In this paper, we propose the first comprehensive methodology that systematically bridges the gap between the theory of provable security for masked implementations and their practical deployment. Our main objective is to provide more rigorous security guarantees than the heuristic approaches currently predominant in the field. By integrating models and metrics from the literature in a structured manner, we enable a transfer of formal security claims into quantitative security levels, grounded on hypotheses that can be experimentally validated. The main technical novelty lies in relaxing ideal assumptions of the noisy leakage model—namely, data isolation and noise independence—into more realistic but still stringent requirements. These relaxed assumptions maintain a strong connection to the ideal hypotheses, minimizing the gap between theoretical proofs and practical implementations.

We also introduce and validate an experimental methodology to assess the validity of these relaxed assumptions, demonstrating its application through the analysis of a masked AES implementation on a commercial off-the-shelf (COTS) Cortex-M4 microcontroller. This case study reveals three main challenges: low noise levels inherent to the device, performance overheads caused by the costly isolation, and the non-tightness of the security proofs for the masking scheme.

The issue of insufficient noise on COTS microcontrollers is well-known in the literature on software masking [CDSU23, BS21, BCS21] and is therefore not specific to our methodology. Simple COTS microcontrollers typically exhibit low noise, which can compromise security. While more complex microcontrollers often have higher noise levels, ensuring isolation on these devices—especially when treated as black boxes—poses additional challenges. From a research perspective, the noise level is a significant constraint but not a fundamental limitation, as it is possible to design custom microcontrollers with higher noise levels (e.g., through the addition of noise engines).

Our isolation technique of inserting “cleaning” instructions is suboptimal: it has a large performance cost ($\times 13$ to $\times 16$ overhead in the number of instructions) while being best-effort (and therefore heavily relying on empirical validation). Leveraging micro-architectural knowledge can facilitate achieving high-confidence and high-performance isolation. Further, strengthening the ISA definition to describe leakage [BGG⁺22] or even provide support for isolation [GMPP20, CPW24], could significantly simplify design, improve guarantees, and dramatically reduce the isolation performance overheads at a low hardware cost.

Finally, an alternative solution for the noise and isolation challenges is to apply our methodology to hardware implementations. Indeed, such implementations have typically higher noise levels and perform operations in parallel. The approach for ensuring isolation (i.e., avoiding glitches, transitions, and couplings) will be different but is already well-understood [FGM⁺18, CS21].

Regarding the non-tightness of security proofs, the high noise requirements and the number of shares needed to achieve a given security level (Table 4) may seem excessive from a practitioner’s perspective, especially when confronted with the current state-of-the-art attacks or the security level recently proven for a single sharing [BCG⁺23]. Below, we identify several sources of non-tightness and suggest directions for improvement.

- In our experiments, we consider a masking scheme based on the expansion technique [BCP⁺20]. While this scheme has the state-of-the-art minimum noise level requirement, the scaling of the security level with respect to the leakage parameter p is sub-optimal: an optimal masking scheme with n shares would scale as $\mathcal{O}(p^n)$, while ours has a lower exponent [BRT21, CFOS21]. This issue can be solved by using tighter random probing security proofs such as the ones based on probe distribution

tables [CFOS21], but this approach would require further development to scale with large computations.

- The next step in the proof is the reduction of noisy leakage to random probing. Using the ARE metric over the SD metric is already a significant gain, as it avoids a field-size loss in the proof. However, in our experiments, the ARE worst-case metric is significantly larger than the SD, canceling part of the gain. This may be due to the low noise level, as adding noise reduces the ARE vs. SD gap. If the use of worst-case leakage metrics remains an issue on noisy devices, one potential solution is to use the reduction to the average random probing model [DFS15b], which would relax the noise requirements at the cost of a stronger security model (hence more complex masking compiler).
- Finally, the reduction from gate-probing to wire-probing introduces a quadratic loss in the random probing parameter. This loss appears to be an artifact of the proof and should be optimized in future work.

On the practical side, our methodology relies on well-defined and realistic physical assumptions. We propose concrete tests to validate some of these assumptions, while others are supported by physics-based arguments. Although our approach advances towards experimentally validated hypotheses, some assumptions (e.g., noise Gaussianity, isolation of consecutive operations implying isolation of non-consecutive ones) could benefit from more direct experimental validation. Further, due to the nature of statistical tests, quantitative assumptions (e.g., independence, isolation) can never be proven, only invalidated. A fully-proven approach would instead rely on quantitative variants of these assumptions, which could be proven with high confidence using statistical tests (e.g., using statistical power and effect size [WO19]). Finally, there is room for efficiency improvements, such as optimizing the noise splitting or reducing the data isolation performance overhead.

In conclusion, our work demonstrates how to achieve provable side-channel security in practice under relaxed leakage assumptions, although current state-of-the-art constructions remain inefficient for the noise levels typical of COTS devices. We have identified several concrete paths for improving the tightness of security proofs, both from a theoretical and practical standpoint. Along with a more optimized isolation implementation, these improvements could lead to significant performance enhancements. By addressing these challenges, we aim to close the gap between the theory and practice of masked implementations, moving towards more efficient and secure designs for embedded cryptographic systems.

Acknowledgements

Gaëtan Cassiers and François-Xavier Standaert are respectively postdoctoral researcher and senior research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded in part by the European Union through the ERC projects 101077506 (acronym AMaskZONE) and 101096871 (acronym BRIDGE). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

- [2716] ISO/IEC JTC 1/SC 27. Information technology – Security techniques – Testing methods for the mitigation of non-invasive attack classes against cryptographic

- modules. Standard, International Organization for Standardization, Geneva, CH, January 2016.
- [ADF16] Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_21.
- [AIS18] Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland. doi:10.1007/978-3-319-96878-0_15.
- [Ajt11] Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 715–724, San Jose, CA, USA, June 6–8, 2011. ACM Press. doi:10.1145/1993636.1993731.
- [AP21] Alexandre Adomnicaï and Thomas Peyrin. Fixslicing AES-like ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):402–425, 2021. URL: <https://tches.iacr.org/index.php/TCHES/article/view/8739>, doi:10.46586/tches.v2021.i1.402-425.
- [BBP⁺16] Sonia Belaïd, Fabrice Benhamouda, Alain Passelègue, Emmanuel Prouff, Adrian Thillard, and Damien Vergnaud. Randomness complexity of private circuits for multiplication. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 616–648, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-49896-5_22.
- [BC22] Olivier Bronchain and Gaëtan Cassiers. Bitslicing arithmetic/boolean masking conversions for fun and profit with application to lattice-based KEMs. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):553–588, 2022. doi:10.46586/tches.v2022.i4.553-588.
- [BCG⁺23] Julien Béguinot, Wei Cheng, Sylvain Guilley, Yi Liu, Loïc Masure, Olivier Rioul, and François-Xavier Standaert. Removing the field size loss from duc et al.’s conjectured bound for masked encodings. In Elif Bilge Kavun and Michael Pehl, editors, *COSADE 2023: 14th International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 13979 of *Lecture Notes in Computer Science*, pages 86–104, Munich, Germany, April 3–4, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-29497-6_5.
- [BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems – CHES 2004*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29, Cambridge, Massachusetts, USA, August 11–13, 2004. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-28632-5_2.
- [BCP⁺20] Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas

- Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 339–368, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland. doi:[10.1007/978-3-030-56784-2_12](https://doi.org/10.1007/978-3-030-56784-2_12).
- [BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39, Santa Barbara, CA, USA, August 17–19, 2016. Springer Berlin Heidelberg, Germany. doi:[10.1007/978-3-662-53140-2_2](https://doi.org/10.1007/978-3-662-53140-2_2).
- [BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ASCAD with a single side-channel trace. Cryptology ePrint Archive, Report 2021/817, 2021. URL: <https://eprint.iacr.org/2021/817>.
- [BDF⁺17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology – EUROCRYPT 2017, Part I*, volume 10210 of *Lecture Notes in Computer Science*, pages 535–566, Paris, France, April 30 – May 4, 2017. Springer, Cham, Switzerland. doi:[10.1007/978-3-319-56620-7_19](https://doi.org/10.1007/978-3-319-56620-7_19).
- [BGG⁺14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In Marc Joye and Amir Moradi, editors, *Smart Card Research and Advanced Applications - 13th International Conference, CARDIS 2014, Paris, France, November 5-7, 2014. Revised Selected Papers*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014. doi:[10.1007/978-3-319-16763-3_5](https://doi.org/10.1007/978-3-319-16763-3_5).
- [BGG⁺22] Roderick Bloem, Barbara Gigerl, Marc Gourjon, Vedad Hadzic, Stefan Mangard, and Robert Primas. Power contracts: Provably complete power leakage models for processors. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022: 29th Conference on Computer and Communications Security*, pages 381–395, Los Angeles, CA, USA, November 7–11, 2022. ACM Press. doi:[10.1145/3548606.3560600](https://doi.org/10.1145/3548606.3560600).
- [BGNT15] Nicolas Bruneau, Sylvain Guilley, Zakaria Najm, and Yannick Tégli. Multivariate high-order attacks of shuffled tables recomputation. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 475–494, Saint-Malo, France, September 13–16, 2015. Springer Berlin Heidelberg, Germany. doi:[10.1007/978-3-662-48324-4_24](https://doi.org/10.1007/978-3-662-48324-4_24).
- [BMP13] Joan Boyar, Philip Matthews, and René Peralta. Logic minimization techniques with applications to cryptology. *Journal of Cryptology*, 26(2):280–312, April 2013. doi:[10.1007/s00145-012-9124-7](https://doi.org/10.1007/s00145-012-9124-7).
- [BMRT22] Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. IronMask: Versatile verification of masking security. In *2022 IEEE Symposium on Security and Privacy*, pages 142–160, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press. doi:[10.1109/SP46214.2022.9833600](https://doi.org/10.1109/SP46214.2022.9833600).

- [BRT21] Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 313–343, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland. doi:[10.1007/978-3-030-77886-6_11](https://doi.org/10.1007/978-3-030-77886-6_11).
- [BRTV21] Sonia Belaïd, Matthieu Rivain, Abdul Rahman Taleb, and Damien Vergnaud. Dynamic random probing expansion with quasi linear asymptotic complexity. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 157–188, Singapore, December 6–10, 2021. Springer, Cham, Switzerland. doi:[10.1007/978-3-030-92075-3_6](https://doi.org/10.1007/978-3-030-92075-3_6).
- [BS21] Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):202–234, 2021. URL: <https://tches.iacr.org/index.php/TCHES/article/view/8973>, doi:[10.46586/tches.v2021.i3.202-234](https://doi.org/10.46586/tches.v2021.i3.202-234).
- [BUS21] Davide Bellizia, Balazs Udvarhelyi, and François-Xavier Standaert. Towards a better understanding of side-channel analysis measurements setups. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 64–79. Springer, 2021. doi:[10.1007/978-3-030-97348-3_4](https://doi.org/10.1007/978-3-030-97348-3_4).
- [CDSU23] Gaëtan Cassiers, Henri Devillez, François-Xavier Standaert, and Balazs Udvarhelyi. Efficient regression-based linear discriminant analysis for side-channel security evaluations towards analytical attacks against 32-bit implementations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):270–293, 2023. doi:[10.46586/tches.v2023.i3.270-293](https://doi.org/10.46586/tches.v2023.i3.270-293).
- [CFOS21] Gaëtan Cassiers, Sebastian Faust, Maximilian Ortl, and François-Xavier Standaert. Towards tight random probing security. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 185–214, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland. doi:[10.1007/978-3-030-84252-9_7](https://doi.org/10.1007/978-3-030-84252-9_7).
- [CGP⁺12] Jean-Sébastien Coron, Christophe Giraud, Emmanuel Prouff, Soline Renner, Matthieu Rivain, and Praveen Kumar Vadnala. Conversion of security proofs from one leakage model to another: A new issue. In Werner Schindler and Sorin A. Huss, editors, *COSADE 2012: 3rd International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81, Darmstadt, Germany, May 3–4, 2012. Springer Berlin Heidelberg, Germany. doi:[10.1007/978-3-642-29912-4_6](https://doi.org/10.1007/978-3-642-29912-4_6).
- [Cho15] Marios O Choudary. Efficient multivariate statistical techniques for extracting secrets from electronic devices. Technical report, University of Cambridge, Computer Laboratory, 2015.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J.

- Wiener, editor, *Advances in Cryptology – CRYPTO’99*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412, Santa Barbara, CA, USA, August 15–19, 1999. Springer Berlin Heidelberg, Germany. doi:[10.1007/3-540-48405-1_26](https://doi.org/10.1007/3-540-48405-1_26).
- [CK13] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 253–270. Springer, 2013. doi:[10.1007/978-3-319-08302-5_17](https://doi.org/10.1007/978-3-319-08302-5_17).
- [CM24] Gaëtan Cassiers and Charles Momin. The SMAesH dataset. Cryptology ePrint Archive, Paper 2024/1521, 2024. URL: <https://eprint.iacr.org/2024/1521>.
- [CPRR14] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In Shiho Moriai, editor, *Fast Software Encryption – FSE 2013*, volume 8424 of *Lecture Notes in Computer Science*, pages 410–424, Singapore, March 11–13, 2014. Springer Berlin Heidelberg, Germany. doi:[10.1007/978-3-662-43933-3_21](https://doi.org/10.1007/978-3-662-43933-3_21).
- [CPW24] Hao Cheng, Daniel Page, and Weijia Wang. eLIMInate: a leakage-focused ISE for masked implementation. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(2):329–358, 2024. doi:[10.46586/tches.v2024.i2.329-358](https://doi.org/10.46586/tches.v2024.i2.329-358).
- [CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2002*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28, Redwood Shores, CA, USA, August 13–15, 2003. Springer Berlin Heidelberg, Germany. doi:[10.1007/3-540-36400-5_3](https://doi.org/10.1007/3-540-36400-5_3).
- [CRZ18] Jean-Sébastien Coron, Franck Rondepierre, and Rina Zeitoun. High order masking of look-up tables with common shares. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):40–72, 2018. URL: <https://tches.iacr.org/index.php/TCHES/article/view/832>, doi:[10.13154/tches.v2018.i1.40-72](https://doi.org/10.13154/tches.v2018.i1.40-72).
- [CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(2):136–158, 2021. URL: <https://tches.iacr.org/index.php/TCHES/article/view/8790>, doi:[10.46586/tches.v2021.i2.136-158](https://doi.org/10.46586/tches.v2021.i2.136-158).
- [DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany. doi:[10.1007/978-3-642-55220-5_24](https://doi.org/10.1007/978-3-642-55220-5_24).
- [DDF19] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. *Journal of Cryptology*, 32(1):151–177, January 2019. doi:[10.1007/s00145-018-9284-1](https://doi.org/10.1007/s00145-018-9284-1).

- [DFS15a] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part I*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-46800-5_16.
- [DFS15b] Stefan Dziembowski, Sebastian Faust, and Maciej Skorski. Noisy leakage revisited. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 159–188, Sofia, Bulgaria, April 26–30, 2015. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-662-46803-6_6.
- [DFS19] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete (or how to evaluate the security of any leaking device), extended version. *Journal of Cryptology*, 32(4):1263–1297, October 2019. doi:10.1007/s00145-018-9277-0.
- [Dod12] Yevgeniy Dodis. Shannon impossibility, revisited. In Adam Smith, editor, *ICITS 12: 6th International Conference on Information Theoretic Security*, volume 7412 of *Lecture Notes in Computer Science*, pages 100–110, Montreal, QC, Canada, August 15–17, 2012. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-642-32284-6_6.
- [DZD⁺17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yungsi Fei. Towards sound and optimal leakage detection procedure. In Thomas Eisenbarth and Yannick Teglia, editors, *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, volume 10728 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2017. doi:10.1007/978-3-319-75208-2_7.
- [FGM⁺18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(3):89–120, 2018. URL: <https://tches.iacr.org/index.php/TCHES/article/view/7270>, doi:10.13154/tches.v2018.i3.89-120.
- [GGJR⁺11] Benjamin Jun Gilbert Goodwill, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011. URL: https://csrc.nist.gov/csrc/media/events/non-invasive-attack-testing-workshop/documents/08_goodwill.pdf.
- [GMPO19] Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. Share-slicing: Friend or foe? *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):152–174, 2019. URL: <https://tches.iacr.org/index.php/TCHES/article/view/8396>, doi:10.13154/tches.v2020.i1.152-174.
- [GMPP20] Si Gao, Ben Marshall, Dan Page, and Thinh Pham. FENL: an ISE to mitigate analogue micro-architectural leakage. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(2):73–98, 2020. URL: <https://tches.iacr.org/index.php/TCHES/article/view/8545>, doi:10.13154/tches.v2020.i2.73-98.

- [GP99] Louis Goubin and Jacques Patarin. DES and differential power analysis (the “duplication” method). In Çetin Kaya Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems – CHES’99*, volume 1717 of *Lecture Notes in Computer Science*, pages 158–172, Worcester, Massachusetts, USA, August 12–13, 1999. Springer Berlin Heidelberg, Germany. doi:[10.1007/3-540-48059-5_15](https://doi.org/10.1007/3-540-48059-5_15).
- [GPSS18] Benjamin Grégoire, Kostas Papagiannopoulos, Peter Schwabe, and Ko Stoffelen. Vectorizing higher-order masking. In Junfeng Fan and Benedikt Gierlichs, editors, *COSADE 2018: 9th International Workshop on Constructive Side-Channel Analysis and Secure Design*, volume 10815 of *Lecture Notes in Computer Science*, pages 23–43, Singapore, April 23–24, 2018. Springer, Cham, Switzerland. doi:[10.1007/978-3-319-89641-0_2](https://doi.org/10.1007/978-3-319-89641-0_2).
- [GS18] Vincent Grosso and François-Xavier Standaert. Masking proofs are tight and how to exploit it in security evaluations. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2018, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 385–412, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland. doi:[10.1007/978-3-319-78375-8_13](https://doi.org/10.1007/978-3-319-78375-8_13).
- [HS13] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, volume 8419 of *Lecture Notes in Computer Science*, pages 219–235. Springer, 2013. doi:[10.1007/978-3-319-08302-5_15](https://doi.org/10.1007/978-3-319-08302-5_15).
- [ISW03] Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer Berlin Heidelberg, Germany. doi:[10.1007/978-3-540-45146-4_27](https://doi.org/10.1007/978-3-540-45146-4_27).
- [JS17] Anthony Journault and François-Xavier Standaert. Very high order masking: Efficient implementation and security evaluation. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, volume 10529 of *Lecture Notes in Computer Science*, pages 623–643, Taipei, Taiwan, September 25–28, 2017. Springer, Cham, Switzerland. doi:[10.1007/978-3-319-66787-4_30](https://doi.org/10.1007/978-3-319-66787-4_30).
- [KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology – CRYPTO’99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397, Santa Barbara, CA, USA, August 15–19, 1999. Springer Berlin Heidelberg, Germany. doi:[10.1007/3-540-48405-1_25](https://doi.org/10.1007/3-540-48405-1_25).
- [Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In Neal Koblitz, editor, *Advances in Cryptology – CRYPTO’96*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113, Santa Barbara, CA, USA, August 18–22, 1996. Springer Berlin Heidelberg, Germany. doi:[10.1007/3-540-68697-5_9](https://doi.org/10.1007/3-540-68697-5_9).
- [Man04] Stefan Mangard. Hardware countermeasures against DPA – A statistical analysis of their effectiveness. In Tatsuaki Okamoto, editor, *Topics in Cryptology*

- *CT-RSA 2004*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235, San Francisco, CA, USA, February 23–27, 2004. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-24660-2_18.
- [MDP19] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):348–375, 2019. URL: <https://tches.iacr.org/index.php/TCHES/article/view/8402>, doi:10.13154/tches.v2020.i1.348-375.
- [MKSM22] Nicolai Müller, David Knichel, Pascal Sasdrich, and Amir Moradi. Transitional leakage in theory and practice unveiling security flaws in masked circuits. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(2):266–288, 2022. doi:10.46586/tches.v2022.i2.266-288.
- [MOS11] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Inf. Secur.*, 5(2):100–110, 2011. doi:10.1049/iet-ifs.2010.0096.
- [MOW17] David McCann, Elisabeth Oswald, and Carolyn Whittall. Towards practical tools for side channel aware software engineering: ‘grey box’ modelling for instruction leakages. In Engin Kirda and Thomas Ristenpart, editors, *USENIX Security 2017: 26th USENIX Security Symposium*, pages 199–216, Vancouver, BC, Canada, August 16–18, 2017. USENIX Association. URL: <https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/mccann>.
- [MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *Topics in Cryptology – CT-RSA 2005*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365, San Francisco, CA, USA, February 14–18, 2005. Springer Berlin Heidelberg, Germany. doi:10.1007/978-3-540-30574-3_24.
- [MPO05] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked AES hardware implementations. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 157–171, Edinburgh, UK, August 29 – September 1, 2005. Springer Berlin Heidelberg, Germany. doi:10.1007/11545262_12.
- [MPW22] Ben Marshall, Dan Page, and James Webb. MIRACLE: MiCRo-Architectural leakage evaluation A study of micro-architectural power leakage across many devices. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(1):175–220, 2022. doi:10.46586/tches.v2022.i1.175-220.
- [MS23] Loïc Masure and François-Xavier Standaert. Prouff and Rivain’s formal security proof of masking, revisited - tight bounds in the noisy leakage model. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023, Part III*, volume 14083 of *Lecture Notes in Computer Science*, pages 343–376, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland. doi:10.1007/978-3-031-38548-3_12.
- [PGMP19] Thomas Prest, Dahmun Goudarzi, Ange Martinelli, and Alain Passelègue. Unifying leakage models on a Rényi day. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 683–712, Santa Barbara,

- CA, USA, August 18–22, 2019. Springer, Cham, Switzerland. [doi:10.1007/978-3-030-26948-7_24](https://doi.org/10.1007/978-3-030-26948-7_24).
- [PPM⁺23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, 55(11):227:1–227:35, 2023. [doi:10.1145/3569577](https://doi.org/10.1145/3569577).
- [PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany. [doi:10.1007/978-3-642-38348-9_9](https://doi.org/10.1007/978-3-642-38348-9_9).
- [QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In Isabelle Attali and Thomas P. Jensen, editors, *Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001. [doi:10.1007/3-540-45418-7_17](https://doi.org/10.1007/3-540-45418-7_17).
- [RP10] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of AES. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems – CHES 2010*, volume 6225 of *Lecture Notes in Computer Science*, pages 413–427, Santa Barbara, CA, USA, August 17–20, 2010. Springer Berlin Heidelberg, Germany. [doi:10.1007/978-3-642-15031-9_28](https://doi.org/10.1007/978-3-642-15031-9_28).
- [SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2008*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425, Washington, DC, USA, August 10–13, 2008. Springer Berlin Heidelberg, Germany. [doi:10.1007/978-3-540-85053-3_26](https://doi.org/10.1007/978-3-540-85053-3_26).
- [SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46, Edinburgh, UK, August 29 – September 1, 2005. Springer Berlin Heidelberg, Germany. [doi:10.1007/11545262_3](https://doi.org/10.1007/11545262_3).
- [SM15] Tobias Schneider and Amir Moradi. Leakage assessment methodology - A clear roadmap for side-channel evaluations. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems – CHES 2015*, volume 9293 of *Lecture Notes in Computer Science*, pages 495–513, Saint-Malo, France, September 13–16, 2015. Springer Berlin Heidelberg, Germany. [doi:10.1007/978-3-662-48324-4_25](https://doi.org/10.1007/978-3-662-48324-4_25).
- [SM16] Tobias Schneider and Amir Moradi. Leakage assessment methodology - extended version. *Journal of Cryptographic Engineering*, 6(2):85–99, June 2016. [doi:10.1007/s13389-016-0120-y](https://doi.org/10.1007/s13389-016-0120-y).
- [SP06] Kai Schramm and Christof Paar. Higher order masking of the AES. In David Pointcheval, editor, *Topics in Cryptology – CT-RSA 2006*, volume 3860 of *Lecture Notes in Computer Science*, pages 208–225, San Jose, CA,

USA, February 13–17, 2006. Springer Berlin Heidelberg, Germany. doi:
[10.1007/11605805_14](https://doi.org/10.1007/11605805_14).

- [WO19] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of ISO 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume 11923 of *Lecture Notes in Computer Science*, pages 256–284, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland. doi:[10.1007/978-3-030-34618-8_9](https://doi.org/10.1007/978-3-030-34618-8_9).

Appendix H

Masked Circuit Compiler in the Cardinal Random Probing Composability Framework

Hereafter is appended the full version of our paper [\[BNR25\]](#), joint work with Victor Normand and Matthieu Rivain, that will be published at **ASIACRYPT 2025**.

Masked Circuit Compiler in the Cardinal Random Probing Composability Framework

Sonia Belaïd¹, Victor Normand^{1,2}, and Matthieu Rivain¹

¹ CryptoExperts, Paris, France

{firstname.lastname}@cryptoexperts.com,

² DIENS, Ecole normale supérieure, PSL University, CNRS, Inria, Paris, France

Abstract. Designing practically secure masked circuits remains a central problem in the field of cryptographic implementation. While most masking schemes have been proven secure in the classical probing model, this model fails to capture more advanced side-channel attacks such as horizontal attacks. In recent years, the community has shifted toward the more realistic random probing model, which offers stronger guarantees. Yet, balancing strong security with practical efficiency continues to be a significant challenge.

In this work, we introduce new tools and constructions that significantly improve the design and analysis of random probing secure circuits. First, we formalize new security notions that combine the benefits of *cardinal* and *general* Random Probing Composability (RPC), two recently introduced notions enabling more flexible and efficient composition of secure gadgets. We then show how uniformly random permutations can be applied to transform any cardinal or general RPC gadget into a so-called *uniformly* cardinal RPC gadget, thereby enhancing security at low cost.

Using these techniques, we propose the first non-linear multiplication gadget, inspired by the recursive construction from CHES 2016, that achieves concrete cardinal RPC security. We provide a detailed comparison with state-of-the-art multiplication gadgets in terms of both random probing advantage and implementation complexity. Building upon this gadget, we design a tighter random probing compiler that strategically uses permutations to improve security bounds while preserving efficiency. Finally, we apply our compiler to the AES and demonstrate improved performance and security compared to existing methods.

Keywords: Masking · Random probing model · Secure composition

1 Introduction

Many cryptographic schemes have been proven secure against black-box attacks by the research community. However, once implemented on physical devices, these schemes become vulnerable to *side-channel attacks*. These attacks exploit the physical characteristics of the device such as execution time, temperature, power consumption, or electromagnetic emissions during algorithm execution to infer sensitive values. *Masking* is one of the most widely deployed countermeasures against *side-channel attacks*. For a sensitive value x , it consists of splitting x into n shares such that the knowledge of any $(n - 1)$ shares is insufficient to deduce x . As a result, an adversary must determine the values of all n shares to recover x , significantly increasing the difficulty of the attack.

To assess the security of circuits protected by masking countermeasures, several *leakage models* have been formalized. These models characterize the capabilities of an adversary attempting to recover sensitive information, ultimately providing security guarantees for the implementation. The *d-probing model*, introduced in [18], allows an adversary to access d internal values from a circuit execution, known as *probes*, and determines whether this limited information suffices to deduce the secret. A circuit is said to be *d-probing secure* if no set of d probed values reveals any sensitive information about the secret. The *d-probing model* has been extensively studied due to its convenience for formal security proofs. However, it remains far from capturing the realities of physical leakages. For example, it does not model powerful horizontal attacks (see, e.g., [4]). In contrast, the *noisy leakage model* [20] assumes that every wire in the circuit leaks a noisy function

of its value, offering a more realistic representation of side-channel attacks. However, this model is much less convenient for formal proofs, which motivated a follow-up work that established a security reduction from the probing model to the noisy leakage model [15]. This reduction introduced an intermediate model, the *random probing model*, previously considered in [18,1]. In this model, each variable leaks its value independently with a fixed probability p , known as the *leakage rate*. Compared to the standard d -probing model, the random probing model enjoys a much tighter reduction to the noisy leakage model.

There are two main approaches to evaluating the random probing security of a circuit. The first consists in using automated tools that exhaustively analyze all possible probing tuples [6,7], or a representative subset using probabilistic methods [12]. The second approach involves deriving a theoretical, generic formula that upper bounds the security level for any number of shares. However, both techniques suffer from scalability limitations and become impractical for large circuits. To overcome this limitation, the more scalable strategy of *composition* consists in proving stronger local security properties for small gadgets. These properties are specifically designed to support secure composition, ensuring that the global circuit inherits a guaranteed level of random probing security. The notion of *random probing composability* (RPC), originally introduced in [6] and referred to as *threshold RPC* in [8] and in this work, enjoys trivial composability. A variant aimed at enabling tighter composition, introduced in [12] and termed *general RPC* in [8] and here, allows for more precise composition bounds but quickly becomes intractable when composing even a few gadgets. More recently, [8] formalized a new notion called *cardinal RPC*, which strikes a balance between compositional tightness and computational efficiency. An alternative approach to composability, based on *leakage diagrams*, was also proposed in [16,11], offering a different framework for reasoning about leakage propagation across composed gadgets.

An increasing number of works have focused on proposing masked circuit compilers aimed at achieving random probing security [1,3,2,17,6,9,10,11,19,8]. To the best of our knowledge, only a few of these works provide concrete security guarantees for actual circuits. This was first achieved by the line of research on *random probing expansion* [6,9,10], which builds upon the expansion strategy from [2] and attains arbitrary security levels for constant leakage probability p up to $2^{-7.5}$. More recently, [11] revisited the ISW scheme [18] within the leakage diagram framework, deriving clean analytical expressions for random probing security as a function of the leakage probability p and the number of shares n in the underlying gadgets. The work of [19] introduced an algebraic approach to random probing security and succeeded in building a secure compiler for leakage rates up to $p = 2^{-5.6}$, leveraging a variant of the multiplication gadget from [4]. Finally, the recent work of [8] provides a concrete random-probing secure implementation of Raccoon [14], relying only on linear masking gadgets and leveraging the cardinal RPC framework.

Our contributions. In this work, we revisit and extend the recent cardinal RPC framework introduced in [8]. Our contributions are as follows:

- We formalize the notion of *random-wiring circuits* and their associated random probing security and composition properties, which underlie the construction from [8] but had not been previously formalized.
- We revisit the notion of *uniformly cardinal RPC*, and show that it achieves security guarantees as tight as general RPC, while significantly reducing the complexity of composition.
- We demonstrate how any gadget satisfying cardinal or general RPC can be transformed into a uniformly cardinal RPC gadget by applying random permutations to the input and output shares. This transformation offers improved security with minimal overhead.
- While [8] focuses solely on linear gadgets, which are enough for masking the Raccoon scheme, we complete the framework by designing a non-linear multiplication gadget inspired by [4]. We prove its cardinal RPC security and provide a detailed comparison with existing multiplication gadgets in terms of random probing security advantage and implementation complexity.
- We present our complete compiler along with its composition strategy, and apply it to the AES circuit, demonstrating that our approach outperforms state-of-the-art methods in terms of the achievable security-performance trade-off.

2 Preliminaries

2.1 Notations

Along the paper, $\mathbb{K}$ shall denote a finite field or a finite ring. For any $n \in \mathbb{N}$, we shall denote $[n]$ the integer set $[n] = [1, n] \cap \mathbb{Z}$. We shall further denote $\mathcal{P}([n]) = \{S \subseteq [n]\}$ the power set of $[n]$ and $\mathcal{S}_n$ the symmetric group on n elements, i.e. the set of all possible permutations of the set $\{1, 2, \dots, n\}$. Each element of $\mathcal{S}_n$ is a permutation, which is a bijective function that maps the set $\{1, 2, \dots, n\}$ to itself.

For some tuple $\mathbf{x} = (x_1, \dots, x_n) \in \mathbb{K}^n$ and for some set $I \subseteq [n]$, the tuple $(x_i)_{i \in I}$ is denoted $\mathbf{x}|_I$. We use the general notation of $\llbracket x \rrbracket = (x_1, \dots, x_n) \in \mathbb{K}^n$ for a sharing of a secret value x .

Distributions. Two probability distributions D_1 and D_2 are ε -close, denoted as $D_1 \approx_\varepsilon D_2$, if their statistical distance is upper bounded by ε , namely

$$\text{SD}(D_1; D_2) := \frac{1}{2} \sum_x |p_{D_1}(x) - p_{D_2}(x)| \leq \varepsilon$$

where $p_{D_1}(\cdot)$ and $p_{D_2}(\cdot)$ represent the probability mass functions of D_1 and D_2 .

Distribution Envelopes. Since exact probabilities are often difficult to compute directly, we follow the approach of [8] and instead rely on upper bounds expressed through the notion of a *probability envelope*. Specifically, for a discrete random variable $X \sim \mathcal{D}$, we define $\mathcal{E}$ as an upper envelope of the distribution, meaning that for any value x , the probability $\mathbb{P}(X = x)$ is upper bounded by $\mathcal{E}(x)$, which we denote by $X \lesssim \mathcal{E}$.

When dealing with probability distributions conditioned on disjoint events, we use collections of probability envelopes indexed by potentially complex structures. For example, we may consider collections $\mathcal{E} = (\mathcal{E}_{(t_1^{\text{out}}, \dots, t_m^{\text{out}})})$ indexed by tuples $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$ as well as collections $\mathcal{E} = (\mathcal{E}_{(J_1, \dots, J_m)})$ indexed by sets $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$. In this case, each tuple corresponds to a distinct distribution and its associated envelope.

2.2 Linear Sharing

In the following, the *n-linear decoding mapping*, denoted by LinDec , refers to the function $\mathbb{K}^n \rightarrow \mathbb{K}$ defined, for any $n \in \mathbb{N}$ and $(x_1, \dots, x_n) \in \mathbb{K}^n$, as

$$\text{LinDec} : (x_1, \dots, x_n) \mapsto x_1 + \dots + x_n$$

We shall further consider that, for every $n, \ell \in \mathbb{N}$, on input $(\llbracket x_1 \rrbracket, \dots, \llbracket x_n \rrbracket) \in (\mathbb{K}^n)^\ell$ the *n-linear decoding mapping* acts as

$$\text{LinDec} : (\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \mapsto (\text{LinDec}(\llbracket x_1 \rrbracket), \dots, \text{LinDec}(\llbracket x_\ell \rrbracket))$$

Definition 1 (Linear Sharing from [6]). Let $n, \ell \in \mathbb{N}$. For any $x \in \mathbb{K}$, an *n-sharing* of x is a random vector $\llbracket x \rrbracket \in \mathbb{K}^n$ such that $\text{LinDec}(\llbracket x \rrbracket) = x$. It is *uniform* if for any set $I \subseteq [n]$ with $|I| < n$ the tuple $\llbracket x \rrbracket|_I$ is uniformly distributed over $\mathbb{K}^{|I|}$. An *n-linear encoding* is a probabilistic algorithm LinEnc which on input a tuple $\mathbf{x} = (x_1, \dots, x_\ell) \in \mathbb{K}^\ell$ outputs a tuple $\llbracket \mathbf{x} \rrbracket = (\llbracket x_1 \rrbracket, \dots, \llbracket x_\ell \rrbracket) \in (\mathbb{K}^n)^\ell$ such that $\llbracket x_i \rrbracket$ is a uniform *n-sharing* of x_i for every $i \in [\ell]$.

2.3 Circuits and Gadgets

An *arithmetic circuit* on a finite field (or finite ring) $\mathbb{K}$ is a labeled directed acyclic graph whose edges are *wires* and vertices are *arithmetic gates* processing operations on $\mathbb{K}$. The circuit is built from a base set of gates $\mathbb{B} = \{g : \mathbb{K}^\ell \rightarrow \mathbb{K}^m\}$, such as addition gates $(x_1, x_2) \mapsto x_1 + x_2$, multiplication gates $(x_1, x_2) \mapsto x_1 \cdot x_2$, and copy gates $x \mapsto (x, x)$. We consider *probabilistic* (or *randomized*) arithmetic circuit that further includes random gates. Each random gate outputs a fresh uniform random value from $\mathbb{K}$.

In the following, we shall call an $(n\text{-share}, \ell\text{-to-}m)$ *gadget*, a randomized arithmetic circuit that maps an input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ to an output $\llbracket \mathbf{y} \rrbracket \in (\mathbb{K}^n)^m$ such that $\mathbf{x} = \text{LinDec}(\mathbf{x}) \in \mathbb{K}^\ell$ and $\mathbf{y} = \text{LinDec}(\mathbf{y}) \in \mathbb{K}^m$ satisfy $\mathbf{y} = g(\mathbf{x})$ for some function g .

3 Random Probing Security for Randomized-Wiring Circuits

In this section, we define the randomized-wiring arithmetic circuits that our constructions will rely on. We also recall the simulation-based notions used to formalize random probing security and adapt them to randomized-wiring arithmetic circuits.

3.1 Randomized-Wiring Circuits

A *randomized-wiring arithmetic circuit* is a (probabilistic) arithmetic circuit, as defined in Section 2.3, in which the wiring, i.e. the connections between gates, is determined randomly. Specifically, a randomized-wiring arithmetic circuit C consists of a fixed set of gates, while the wires connecting these gates are sampled according to a predefined probability distribution. Formally, C is characterized by: (1) a set of gates, and (2) a distribution over possible wire configurations. The randomness of this distribution is parameterized by a *random tape* ρ . We denote by C^ρ the deterministic arithmetic circuit obtained by instantiating C with a particular realization of ρ .

This model is particularly useful for capturing masked circuits and gadgets in which operations are performed on shares whose indices are randomized. An example is the cardinal RPC refresh gadget, **RPRfresh**, introduced in [8]. This shall also capture the gadgets proposed in this paper that apply random permutations to their input or output shares (see Section 4).

It is important to emphasize that in our model, the random tape ρ is *fully leaked to the adversary*. That is, the simulator in the random probing model receives C^ρ for a randomly chosen ρ . Consequently, the randomness in wiring is fundamentally different from the randomness from random gates. First, they operate over different domains: random wiring typically involves selecting indices from $[n]$, whereas random gates produce values in the base field (or ring) $\mathbb{K}$. Second, and more critically, while random gates must be implemented with mechanisms that limit leakage (quantified by the leakage probability p in the random probing model), randomized wiring does *not* require such protection. Since the wiring randomness is entirely leaked to the adversary, its generation might rely on much lighter mechanisms in practice.

In the following, we extend existing notions of random probing security and composition to encompass randomized-wiring arithmetic circuits and gadgets.

3.2 Random Probing Security

Let $p \in [0, 1]$ be some constant leakage probability parameter, a.k.a. the *leakage rate*. In the p -random probing model, an evaluation of an arithmetic circuit C^ρ , for a randomly chosen ρ , leaks the value carried by each wire with a probability p , all the wire leakage events being mutually independent. We also consider that the adversary knows the circuit instance C^ρ and its underlying wiring which is captured by allowing a full leakage of the random tape ρ .

As in [6], we formally define the random-probing leakage of an arithmetic circuit from the two following probabilistic algorithms:

- The *leaking-wires sampler* takes as input an arithmetic circuit C^ρ , for a randomly chosen ρ , and a probability $p \in [0, 1]$, and outputs a set $\mathcal{W}$, denoted as

$$\mathcal{W} \leftarrow \text{LeakingWires}(C^\rho, p) ,$$

where $\mathcal{W}$ is constructed by including each wire label from the circuit with probability p to $\mathcal{W}$ (where all the probabilities are mutually independent).

- The *assign-wires sampler* takes as input an arithmetic circuit C^ρ , for a randomly chosen ρ , a set of wire labels $\mathcal{W}$ (subset of the wire labels of C^ρ), and an input $\mathbf{x}$, and it outputs a $|\mathcal{W}|$ -tuple $\mathbf{w} \in \mathbb{K}^{|\mathcal{W}|} \cup \{\perp\}$ ($\{\perp\}$ if $|\mathcal{W}| = 0$), denoted as

$$\mathbf{w} \leftarrow \text{AssignWires}(C^\rho, \mathcal{W}, \mathbf{x}) ,$$

where $\mathbf{w}$ corresponds to the assignments of the wires of C^ρ with label in $\mathcal{W}$ for an evaluation on input $\mathbf{x}$.

We now extend the definitions of *random probing leakage* and *random probing security* from [6] to the setting of randomized-wiring circuits. We remark that in the special case where the wiring distribution is a singleton (i.e. the circuit has a fixed wiring), these definitions coincide with the original ones for standard arithmetic circuits.

Definition 2 (Random Probing Leakage). *The p -random probing leakage of a randomized-wiring arithmetic circuit C on input $\mathbf{x}$ is the distribution $\mathcal{L}_p(C, \mathbf{x})$ obtained by picking a wiring random tape ρ and composing the leaking-wires and assign-wires samplers applies to C^ρ :*

$$\mathcal{L}_p(C, \mathbf{x}) \stackrel{id}{=} (\text{AssignWires}(C^\rho, \text{LeakingWires}(C^\rho, p), \mathbf{x}) \mid \rho \leftarrow \$)$$

Definition 3 (Random Probing Security). *A randomized-wiring arithmetic circuit C with $\ell \cdot n \in \mathbb{N}$ input gates is (p, ε) -random probing secure with respect to encoding Enc if there exists a simulator Sim such that for every $\mathbf{x} \in \mathbb{K}^\ell$:*

$$\text{Sim}(C^\rho) \approx_\varepsilon \mathcal{L}_p(C, \text{Enc}(\mathbf{x}))$$

3.3 Random Probing Composability

Proving the random probing security directly is generally feasible only for small gadgets and does not scale for larger circuits. To address this, we rely on composability notions: by proving stronger local security properties for individual gadgets, we can demonstrate that their composition guarantees global random probing security for the entire circuit.

Several composability notions for random probing security have been introduced in the literature, each offering different trade-offs between precision and verification efficiency. The *(threshold) random probing composability* (RPC) notion, introduced in [6], requires that the random probing leakage of a gadget's internal wires, along with any set of up to t output shares, can be perfectly simulated using only up to t input shares, except with a simulation failure probability ε . A key advantage of this notion is its *trivial composability*: any circuit composed of threshold RPC gadgets remains threshold RPC, with the overall failure probability being the sum of the individual failure probabilities. While powerful, the threshold RPC notion lacks tightness, as requiring more than t input shares to simulate a single gadget does not necessarily imply failure in a composed setting.

To address this limitation, the notion of *general random probing composability* (general RPC) was later introduced [12,8]. In this framework, the simulation failure probability is specified for each combination of output shares (to be simulated along with internal random probing leakage) and input shares (used in the simulation). This leads to the concept of the *probe distribution table* (PDT) introduced in [12], also referred to as the general RPC envelope in [8] and in our work. However, computing such tables or envelopes is exponential in the number of input and output shares, quickly becoming intractable when composing even a small number of gadgets into a larger cluster. Specifically, a cluster with m input sharings and ℓ output

sharings has a general RPC envelope that contains (bounds on) $2^{(m+\ell) \cdot n}$ distinct failure probabilities, where n is the number of shares per sharing.

To mitigate this, the notion of *cardinal random probing composability* (cardinal RPC) was introduced in [8] as a compromise between tightness and efficiency. This approach characterizes the simulation failure probability for every number of output shares (to be simulated) and input shares (used in the simulation), considering only the cardinality of these sets rather than their specific elements. Thus, a cardinal RPC envelope for a cluster with m input sharings and ℓ output sharings consists of (bounds on) $(n+1)^{(m+\ell)}$ distinct failure probabilities. This is polynomial in the number of shares n , rather than exponential (as general RPC), though it remains exponential in the number of input and output sharings. This abstraction hence significantly reduces the complexity of composition while still preserving a meaningful degree of tightness.

We adapt these three composability notions to the setting of randomized-wiring circuits. As before, in the special case where the wiring distribution is a singleton, these adapted definitions coincide with their original counterparts.

Definition 4 (Threshold Random Probing Composability). *Let $n, \ell, m, t \in \mathbb{N}$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is (p, t, ε) -threshold random probing composable (threshold RPC for short) for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket x \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \dots, J_m)$ such that $|J_1| \leq t, \dots, |J_m| \leq t$, the outputs of the following experiment:*

$$\begin{aligned} \rho &\leftarrow \$ \\ \mathcal{W} &\leftarrow \text{LeakingWires}(G^\rho, p) \\ I_1, \dots, I_\ell &\leftarrow \text{Sim}_1(G^\rho, \mathcal{W}, J_1, \dots, J_m) \\ \text{out}_1, \text{out}_2 &\leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket|_{I_1}, \dots, \llbracket x_\ell \rrbracket|_{I_\ell}) \\ \text{return } &(I_1, \dots, I_\ell, \text{out}_1, \text{out}_2) \end{aligned}$$

are such that

1. $\mathbb{P}(|I_1| \leq t, \dots, |I_\ell| \leq t) \leq \varepsilon$,
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G^\rho, \llbracket x \rrbracket, \llbracket y \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$ and $\llbracket y \rrbracket \leftarrow G^\rho(\llbracket x \rrbracket)$.

Definition 5 (General Random Probing Composability). *Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^g$ be a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and defined over $\mathcal{P}([n])^\ell$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^g)$ -general random probing composable (general RPC for short) for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for every shared input $\llbracket x \rrbracket \in (\mathbb{K}^n)^\ell$ and for every set collection $(J_1, \dots, J_m)$, the outputs of the following experiment:*

$$\begin{aligned} \rho &\leftarrow \$ \\ \mathcal{W} &\leftarrow \text{LeakingWires}(G^\rho, p) \\ I_1, \dots, I_\ell &\leftarrow \text{Sim}_1(G^\rho, \mathcal{W}, J_1, \dots, J_m) \\ \text{out}_1, \text{out}_2 &\leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket|_{I_1}, \dots, \llbracket x_\ell \rrbracket|_{I_\ell}) \\ \text{return } &(I_1, \dots, I_\ell, \text{out}_1, \text{out}_2) \end{aligned}$$

are such that

1. $(I_1, \dots, I_\ell) \lesssim \mathcal{E}_{J_1, \dots, J_m}^g$,

2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\mathcal{L}_{\mathcal{W}}(G^\rho, \llbracket \mathbf{x} \rrbracket, \llbracket \mathbf{y} \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(C, p)$ and $\llbracket \mathbf{y} \rrbracket \leftarrow G^\rho(\llbracket \mathbf{x} \rrbracket)$.

Definition 6 (Uniformly Cardinal RPC - Formalization from [8]). Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^u$ be a collection of probability envelopes indexed by $(t_1^o, \dots, t_m^o) \subseteq [n]^m$ and defined over $[n]^\ell$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is uniformly $(p, \mathcal{E}^u)$ -cardinal random probing composable (uniformly cardinal RPC for short) for some $p \in [0, 1]$ if it is $(p, \mathcal{E}^u)$ -cardinal RPC and if it is $(p, \mathcal{E}^g)$ -general RPC such that the collection of general RPC envelopes $\mathcal{E}^g$ satisfies the two following conditions:

1. for all $(J_1, \dots, J_m), (J'_1, \dots, J'_m) \in \mathcal{P}([n])^m$ such that $(|J_1|, \dots, |J_m|) = (|J'_1|, \dots, |J'_m|)$, and for all $(I_1, \dots, I_\ell), (I'_1, \dots, I'_\ell) \in \mathcal{P}([n])^\ell$ such that $(|I_1|, \dots, |I_\ell|) = (|I'_1|, \dots, |I'_\ell|)$, we have

$$\mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \mathcal{E}_{J'_1, \dots, J'_m}^g(I'_1, \dots, I'_\ell).$$

2. for all $(t_1^i, \dots, t_\ell^i) \in [n]^\ell$, $(t_1^o, \dots, t_m^o) \in [n]^m$, $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ such that $|J_1| = t_1^o, \dots, |J_m| = t_m^o$ and $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ such that $|I_1| = t_1^i, \dots, |I_\ell| = t_\ell^i$, we have

$$\mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^i} = \mathcal{E}_{t_1^o, \dots, t_m^o}^u(t_1^i, \dots, t_\ell^i).$$

Definition 7 (Cardinal Random Probing Composability). Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^c$ be a collection of probability envelopes indexed by $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$ and defined over $[0, n]^\ell$. An n -share randomized-wiring gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^c)$ -cardinal random probing composable ($(p, \mathcal{E}^c)$ -cardinal RPC) for some $p \in [0, 1]$ if there exists a PPT stateful two-stage simulator $(\text{Sim}_1, \text{Sim}_2)$ such that for all shared input $\llbracket \mathbf{x} \rrbracket \in (\mathbb{K}^n)^\ell$ and for every collection $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$, the outputs of the Experiment 1:

Experiment 1: RPC	
$\rho \leftarrow \$$ $\mathcal{W} \leftarrow \text{LeakingWires}(G^\rho, p)$ $I_1, \dots, I_\ell \leftarrow \text{Sim}_1(G^\rho, \mathcal{W}, J_1, \dots, J_m)$ $out_1, out_2 \leftarrow \text{Sim}_2(\llbracket x_1 \rrbracket _{I_1}, \dots, \llbracket x_\ell \rrbracket _{I_\ell})$ $\text{return } (I_1, \dots, I_\ell, out_1, out_2)$	<i>Sampling the random tape for the wires.</i> <i>Sampling the leaking wires.</i> <i>From the indexes of the leaking outputs, Sim₁ determines the indexes of the required inputs.</i> <i>From the selected input shares, Sim₂ returns simulated values of the internal leaking wires out₁ and the output leaking wires out₂</i>

are such that

1. $(|I_1|, \dots, |I_\ell|) \lesssim \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^c$, where $t_1^{\text{out}} = |J_1|, \dots, t_m^{\text{out}} = |J_m|$, and
- 2.

$$(out_1, out_2) \stackrel{id}{=} \left(\underbrace{\text{AssignWires}(G^\rho, R, \mathcal{W}, \llbracket \mathbf{x} \rrbracket)}_{\text{Induced Random probing leakage}}, \underbrace{(\llbracket y_1 \rrbracket|_{J_1}, \dots, \llbracket y_m \rrbracket|_{J_m})}_{\text{Output leakage}} \right),$$

where $\mathcal{W} \leftarrow \text{LeakingWires}(G^\rho, p)$ and $\llbracket \mathbf{y} \rrbracket \leftarrow G^\rho(\llbracket \mathbf{x} \rrbracket)$.

We now explicitly derive the envelope for the cardinal RPC property of a gadget from its envelope for the general RPC property in Lemma 1. This follows directly from Definitions 5 and 7, using the same simulators for both definitions.

Lemma 1. *Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^g$ be a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and defined over $\mathcal{P}([n])^\ell$. Let $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ be an n -share randomized-wiring gadget achieving $(p, \mathcal{E}^g)$ -general RPC for some $p \in [0, 1]$. Then, G is also $(p, \mathcal{E}^c)$ -cardinal RPC with $\mathcal{E}^c$ defined for all $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$ and for all $(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \in [0, n]^\ell$, as follows:*

$$\mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^c(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) = \max_{\substack{J_1, \dots, J_m \\ |J_1| = t_1^{\text{out}}, \dots, |J_m| = t_m^{\text{out}}}} \sum_{\substack{I_1, \dots, I_\ell \\ |I_1| = t_1^{\text{in}}, \dots, |I_\ell| = t_\ell^{\text{in}}}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell)$$

4 Uniformly Cardinal RPC Property from Permutations

In this section, we revisit and refine the notion of uniformly cardinal RPC introduced in [8], and demonstrate how to enforce it effectively. Our goal is to improve the trade-offs offered by earlier simulation-based composition notions (namely threshold, cardinal, and general RPC) particularly with respect to the balance between tight security bounds and scalability.

As discussed previously, general RPC offers the tightest bounds on the adversarial advantage but incurs exponential composition complexity. With threshold RPC, the verification of the composition is highly efficient, yet it yields looser bounds. Cardinal RPC strikes a middle ground, trading some tightness for improved compositional efficiency by restricting leakage set structures. Uniformly cardinal RPC, briefly introduced in [8], emerges as a better compromise: when a gadget is uniformly cardinal RPC, then the general RPC and cardinal RPC envelopes are equivalent, so one retains the tightness of general RPC while inheriting the lower verification complexity of cardinal RPC. In particular, [8, Corollary 1] shows that leveraging this property when composing two refresh gadgets inside a global copy gadget improves the overall bounds compared to using cardinal RPC alone. That said, [8] does not provide concrete gadgets satisfying the uniformly cardinal RPC property, nor design principles to obtain them, leaving construction strategies as an open problem.

In the following, we decompose the uniformly cardinal RPC notion into two jointly equivalent sub-properties: *UI cardinal RPC* and *UO cardinal RPC*, which isolate the effects of uniformity on the input and output shares, respectively. We show that UI cardinal RPC provides clear advantages in certain composition scenarios, while several existing gadgets already satisfy UO cardinal RPC for tight envelopes. Moreover, we demonstrate that applying uniform random permutations to the input (resp. output) shares of a gadget is sufficient to ensure UI (resp. UO) cardinal RPC, with a small computational overhead.

We conclude the section with a concrete example, a butterfly network, that illustrates the strengths and limitations of each composability notion across various settings. The proofs of the different lemmas introduced in this section are provided in Appendix B.

4.1 Uniformly Cardinal RPC Property

Informally, a gadget is said to be uniformly cardinal RPC if it is cardinal RPC, and also general RPC for an envelope collection in which, for any set of input shares of a given cardinality, the envelopes defined over all sets of output share of the same cardinality are identical. This notion is formalized in Definition 6. If a gadget is uniformly cardinal RPC, then its general RPC and cardinal RPC envelopes are equivalent. Consequently, we retain the tight security guarantees of general RPC (for the corresponding envelope collection) while retaining the efficiency of cardinal RPC for verifying its composition in a larger circuit. Indeed, the uniformity condition ensures that no information is missing beyond what is already captured by the cardinal RPC envelope.

Building on this property, we introduce two intermediate notions: UI cardinal RPC (Definition 8) and UO cardinal RPC (Definition 9). Informally, UI cardinal RPC captures the idea of uniformity with respect

to the input shares, while UO cardinal RPC captures uniformity with respect to the output shares. These two notions are jointly equivalent to uniformly cardinal RPC, as stated in Lemma 2.

This decomposition is motivated by the observation that a gadget may satisfy neither, one, or both of these sub-properties, each case having distinct implications for composability. By isolating the role of uniformity on input and output shares separately, we gain more flexibility to efficiently enforce uniformity in gadget construction or composition.

Definition 8 (UI Cardinal RPC). Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^{ui}$ be a collection of probability envelopes indexed by $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$ over $[0, n]^\ell$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^{ui})$ -UI cardinal random probing composable for some $p \in [0, 1]$ if it is $(p, \mathcal{E}^{ui})$ -cardinal RPC and if it is $(p, \mathcal{E}^g)$ -general RPC such that the collection of general RPC envelopes $\mathcal{E}^g$ satisfies the two following conditions:

1. for all $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$, $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$, $(I'_1, \dots, I'_\ell) \in \mathcal{P}([n])^\ell$ such that $(|I_1|, \dots, |I_\ell|) = (|I'_1|, \dots, |I'_\ell|)$, we have

$$\mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \mathcal{E}_{J_1, \dots, J_m}^g(I'_1, \dots, I'_\ell),$$

2. for all $(t_1^{in}, \dots, t_\ell^{in}) \in [0, n]^\ell$, $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$, $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ such that $|I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}$, we have

$$\max_{\substack{J_1, \dots, J_m \\ |J_1| = t_1^{out}, \dots, |J_m| = t_m^{out}}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{in}} = \mathcal{E}_{t_1^{out}, \dots, t_m^{out}}^{ui}(t_1^{in}, \dots, t_\ell^{in}).$$

Definition 9 (UO Cardinal RPC). Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^{uo}$ be a collection of probability envelopes indexed by $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$ over $[0, n]^\ell$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is $(p, \mathcal{E}^{uo})$ -UO cardinal random probing composable for some $p \in [0, 1]$ if it is $(p, \mathcal{E}^{uo})$ -cardinal RPC and if it is $(p, \mathcal{E}^g)$ -general RPC such that the collection of general RPC envelopes $\mathcal{E}^g$ satisfies the two following conditions:

1. for all $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$, $(J'_1, \dots, J'_m) \in \mathcal{P}([n])^m$ such that $(|J_1|, \dots, |J_m|) = (|J'_1|, \dots, |J'_m|)$, for all $(t_1^{in}, \dots, t_\ell^{in}) \in [0, n]^\ell$, we have

$$\sum_{\substack{I_1, \dots, I_\ell \\ |I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \sum_{\substack{I_1, \dots, I_\ell \\ |I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}}} \mathcal{E}_{J'_1, \dots, J'_m}^g(I_1, \dots, I_\ell).$$

2. for all $(t_1^{in}, \dots, t_\ell^{in}) \in [0, n]^\ell$, $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$, $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ such that $|J_1| = t_1^{out}, \dots, |J_m| = t_m^{out}$,

$$\sum_{\substack{I_1, \dots, I_\ell \\ |I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \mathcal{E}_{t_1^{out}, \dots, t_m^{out}}^{uo}(t_1^{in}, \dots, t_\ell^{in}).$$

Lemma 2. Let $n, \ell, m \in \mathbb{N}$. Let $\mathcal{E}^u, \mathcal{E}^{ui}, \mathcal{E}^{uo}$ be collections of probability envelopes indexed by $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$ over $[0, n]^\ell$. An n -share gadget $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ is uniformly- $(p, \mathcal{E}^u)$ -cardinal RPC for some $p \in [0, 1]$ if and only if G is $(p, \mathcal{E}^{ui})$ -UI cardinal RPC and $(p, \mathcal{E}^{uo})$ -UO cardinal RPC with $\mathcal{E}^u = \mathcal{E}^{ui} = \mathcal{E}^{uo}$.

4.2 Improved Properties of UI/UO Cardinal RPC Gadgets

As discussed earlier, for a $(p, \mathcal{E}^G)$ -general RPC gadget G with a tight envelope collection, if G also satisfies the uniform cardinal RPC property with equality constraints satisfied on the envelope collection $\mathcal{E}^G$, then the corresponding $\mathcal{E}^u$ uniformly cardinal RPC envelope collection carries equivalent information in terms of security, while making it possible to compute the advantage of the composition much more efficiently.

More specifically, we show that the UI cardinal RPC property alone is already sufficient to achieve tighter security bounds in certain composition settings. In [8, Corollary 1], it was demonstrated that composing two uniformly cardinal RPC refresh gadgets into a copy gadget improves the resulting security bound. Here, we extend this result in three important directions:

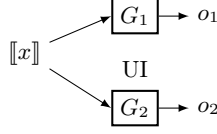


Fig. 1: Tree-UI gadget built from a cardinal RPC gadget G_1 and a UI cardinal RPC gadget G_2 .

- (i) the result holds for arbitrary gadgets, not just refresh gadgets;
- (ii) the UI cardinal RPC property alone is sufficient (UO cardinal RPC is not required);
- (iii) it suffices for only one of the two gadgets in the copy composition to satisfy UI cardinal RPC.

Concretely, we consider a gadget constructed as the parallel composition of two gadgets, G_1 and G_2 , which share a common input sharing $\llbracket x \rrbracket$ (as depicted in Figure 1). We refer to this type of construction as a tree-UI gadget. To simulate the overall leakage, shares of $\llbracket x \rrbracket$ are needed both for G_1 and G_2 . The key observation is that if at least one of these gadgets, say G_2 , satisfies the UI cardinal RPC property, then the number of shares required to simulate the combined leakage can be tightly bounded, as formalized in Lemma 3. In contrast, if neither gadget satisfies UI cardinal RPC, the estimation becomes looser: in the worst case, the number of required shares must be treated as the sum of those needed for each individual gadget, potentially up to the full number of shares n .

Lemma 3 (Cardinal RPC of Tree-UI). *Let $\llbracket \mathbf{x} \rrbracket$ a n -sharing of a secret value $\mathbf{x} \in \mathbb{K}$. Let $G_1 : \mathbb{K}^n \rightarrow \mathbb{K}^n$ and $G_2 : \mathbb{K}^n \rightarrow \mathbb{K}^n$ two n -share gadgets. Let $p \in [0, 1]$. If G_1 is $(p, \mathcal{E}^1)$ -cardinal RPC and G_2 is $(p, \mathcal{E}^{ui2})$ -UI cardinal RPC, then the Tree-UI circuit described in Figure 1 is $(p, \mathcal{E}^{TUI})$ -cardinal RPC such that for all $t^{in} \in [0, n]$ and all $(t_1^{out}, t_2^{out}) \in [0, n]^2$ we have*

$$\mathcal{E}_{t_1^{out}, t_2^{out}}^{TUI}(t^{in}) = \sum_{i_1=0}^{t^{in}} \sum_{i_2=t^{in}-i_1}^{t^{in}} \frac{\binom{i_1}{i_1+i_2-t^{in}} \cdot \binom{n-i_1}{t^{in}-i_1}}{\binom{n}{i_2}} \cdot \mathcal{E}_{t_1^{out}}^1(i_1) \cdot \mathcal{E}_{t_2^{out}}^{ui2}(i_2)$$

Following the approach of [8], we consider copy gates whenever a sharing is duplicated to serve as input to two distinct gadgets. In this setting, the envelope of Tree-UI, corresponding to Lemma 3 and denoted by $\mathcal{E}^{TUI}$, takes the form

$$\mathcal{E}_{t_1^{out}, t_2^{out}}^{TUI}(t^{in}) = \sum_{i=0}^{t^{in}} \mathcal{E}_{t_1^{out}, t_2^{out}}^{TUI}(i) \cdot \binom{n-i}{t^{in}-i} \cdot p^{t^{in}-i} \cdot (1-p)^{n-t^{in}}.$$

If copy gates are not explicitly taken into account, i.e. if duplications are modeled only by gadgets with fan-out 2, then we can rely directly on Lemma 3, or interpret the above expression, as well as our subsequent empirical results, as upper bounds.

4.3 Transforming Gadgets to UI/UO or Uniformly Cardinal RPC

Any $(p, \mathcal{E}^G)$ -general RPC gadget G for a given envelope collection $\mathcal{E}^G$ is trivially a $(p, \mathcal{E}^u)$ -uniformly RPC gadget when considering the following:

- A less tight general RPC envelope collection $\mathcal{E}^{G'}$, where for all $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$, the envelopes satisfy

$$\mathcal{E}_{J_1, \dots, J_m}^{G'}(I_1, \dots, I_\ell) = \max_{\substack{(J'_1, \dots, J'_m) \in \mathcal{P}([n])^m \\ |J_1|=|J'_1|, \dots, |J_m|=|J'_m| \\ (I'_1, \dots, I'_\ell) \in \mathcal{P}([n])^\ell \\ |I_1|=|I'_1|, \dots, |I_\ell|=|I'_\ell|}} \mathcal{E}_{J'_1, \dots, J'_m}^G(I'_1, \dots, I'_\ell)$$

which relaxes the envelope structure while maintaining the essential properties for uniformly cardinal RPC.

- A uniform cardinal RPC envelope collection $\mathcal{E}^u$ for all $(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \in [0, n]^\ell$, for all $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$, and $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ such that $|J_1| = t_1^{\text{out}}, \dots, |J_m| = t_m^{\text{out}}$, and $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ with $|I_1| = t_1^{\text{in}}, \dots, |I_\ell| = t_\ell^{\text{in}}$, defined as

$$\mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^u(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) = \mathcal{E}_{J_1, \dots, J_m}^{G'}(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{\text{in}}}.$$

This ensures that the transformed envelope $\mathcal{E}^u$ retains all the information from $\mathcal{E}^{G'}$.

Similar results can be derived for the UI and UO cardinal RPC notions as well. In this section, we demonstrate that existing gadgets from the literature, specifically the refresh, addition, and scalar multiplication gadgets from [8], are directly UO cardinal RPC, derived from the tight general RPC envelopes we exhibit. We also show how any general RPC gadget can be transformed into a tighter (or, at worst, equivalent) UI or UO cardinal RPC gadget by composing it with uniformly random permutations of the input and/or output shares.

Direct UO Cardinal RPC of Existing Gadgets. In [8], the authors demonstrate that their refresh gadget RPPRefresh^γ satisfies the cardinal RPC property, using an intermediate security notion named *Random Probing Partitioning* (RPP), that we recall in Appendix A. This gadget is built using a zero encoding gadget, RPZeroEnc^γ , as described in Algorithm 4 in Appendix C.1. Informally, at each of the γ iterations, two distinct indices $i_1, i_2 \in [n]$ are sampled uniformly and independently, and a fresh random value from $\mathbb{K}$ is added to the shares at these positions. In fact, the security of the gadget improves as γ increases, until some threshold.

In Lemma 4, we present what we believe to be a tight and general RPC envelope collection for the refresh gadget. Our construction builds upon the RPP envelope of the intermediate gadget $\text{RPZeroEnc}_+^{n, \gamma}$, as introduced in [8]. This gadget corresponds to the zero-encoding phase of the refresh gadget, augmented with additional leakage on the output shares. The RPP envelope characterizes the distribution of output shares of $\text{RPZeroEnc}_+^{n, \gamma}$ by defining an upper bound to the probability of each possible partition into independent, uniformly random 0-sharings. These bounds serve as a key ingredient to define a tight general RPC envelope collection for RPPRefresh . We then establish in Corollary 1 that, based on this general RPC envelopes collection, the gadget also satisfies the UO cardinal RPC property.

Lemma 4 (General RPC of RPPRefresh). *Let $n, \gamma \in \mathbb{N}$ and $p \in [0, 1]$. If $\text{RPZeroEnc}_+^{n, \gamma}$ is $(p, \mathcal{E}^\gamma)$ -RPP, then RPPRefresh is $(p, \mathcal{E}^g)$ -general RPC such that for all $I \subseteq [n], J \subseteq [n]$, we have*

$$\mathcal{E}_J^g(I) = \sum_{K \subseteq I} p^{|I|-|K|} \cdot (1-p)^{n-|I|} \cdot \left(\sum_{P \in \mathbb{P}(n)} \sum_{(k_1, \dots, k_\ell) \in \mathcal{A}_{K, J}^P} \mathcal{E}^\gamma(P) \cdot \frac{\prod_{i=1}^{\ell} \binom{c_i}{k_i}}{\binom{|J|}{|K|} \cdot \binom{n}{|J|}} \right)$$

where $P := (c_1, \dots, c_\ell)$ is an integer partition of n , $\mathbb{P}(n)$ denotes the set of all integer partitions of n , and

$$\mathcal{A}_{K, J}^P := \{(k_1, \dots, k_\ell) \in \mathbb{N}^\ell \mid 0 \leq k_i \leq c_i \wedge (\sum_{i=1}^{\ell} k_i = |J|) \wedge (\sum_{i=1}^{\ell} k_i \mathbf{1}_{(k_i=c_i)} = |K|)\}.$$

The UO cardinal RPC property follows from the expression of the general RPC envelope collection of RPPRefresh , as stated in Corollary 1.

Corollary 1 (UO cardinal RPC of RPPRefresh from [8]). *The n -share gadget RPPRefresh^γ from [8] is $(p, (\mathcal{E}_J^{uo})_{J \subseteq [n]})$ -UO cardinal RPC. The envelopes $\mathcal{E}_J^{uo}$ are derived from the general RPC envelopes $\mathcal{E}_J^g$ as in Definition 9.*

The linear gadgets from [8] are built from the RPPRefresh^γ refresh gadget. The addition gadget, Add^γ , refreshes both input n -sharings before performing the addition, share by share, as outlined in Algorithm 6 in Appendix C.1. Similarly, the scalar multiplication gadget, cMult^γ , refreshes its input n -sharing and then

performs the scalar multiplication, share by share, as described in Algorithm 7 in Appendix C.1. Building on the results for the refresh gadget, we compute what we expect to be tight general RPC envelope collections for these gadgets in Lemmas 5 and 6 and show that, based on these envelope collections, they also satisfy the UO cardinal RPC property (in Corollaries 2 and 3 whose proofs are given in Appendix C.1).

Lemma 5. *Let $n \in \mathbb{N}$. Let $p \in [0, 1]$. Let $\mathcal{E}^g$ represent a collection of probability envelopes indexed by $J \subseteq [n]$. Then Add^γ is $(p, \mathcal{E}^g)$ -general RPC. For all sets $(I_1, I_2) \subseteq [n]$ and set $J \subseteq [n]$, we have:*

$$\mathcal{E}_J^g(I_1, I_2) = \sum_{I'_1 \subseteq [n]} \sum_{I'_2 \subseteq [n]} \mathcal{E}_{I'_1}^{g_1}(I_1) \cdot \mathcal{E}_{I'_2}^{g_1}(I_2) \cdot \mathcal{E}_J(I'_1, I'_2)$$

where $\mathcal{E}^{g_1}$ is the general RPC envelope of RPRrefresh and for all $(I'_1, I'_2) \subseteq [n]^2$, we have

$$\mathcal{E}_J(I'_1, I'_2) = \begin{cases} 0 & \text{if } J \not\subseteq I'_1 \text{ or } J \not\subseteq I'_2 \\ p^{|I'_1| - |J|} \cdot (1-p)^{n - |I'_1|} \cdot p^{|I'_2| - |J|} \cdot (1-p)^{n - |I'_2|} & \text{otherwise.} \end{cases}$$

Corollary 2. Add^γ is $(p, \mathcal{E}^{uo})$ -UO cardinal RPC. Where $\mathcal{E}^{uo}$ is defined as in Definition 9, derived from the general RPC envelopes $\mathcal{E}^g$ from Lemma 5.

Lemma 6. *Let $n \in \mathbb{N}$. Let $p \in [0, 1]$. Let $\mathcal{E}^g$ represent a collection of probability envelopes indexed by $J \subseteq [n]$. Then cMult^γ is $(p, \mathcal{E}^g)$ -general RPC. For all set $I \subseteq [n]$ and set $J \subseteq [n]$, we have:*

$$\mathcal{E}_J^g(I) = \sum_{I_1 \subseteq [n]} \mathcal{E}_{I_1}^{g_1}(I) \cdot \mathcal{E}_J(I_1)$$

where $\mathcal{E}^{g_1}$ is the general RPC envelope of RPRrefresh and

$$\mathcal{E}_J(I_1) = \begin{cases} 0 & \text{if } J \not\subseteq I_1 \\ p^{|I_1| - |J|} \cdot (1-p)^{n - |I_1|} & \text{otherwise} \end{cases}$$

Corollary 3. cMult^γ is $(p, \mathcal{E}^{uo})$ -UO cardinal RPC. Where $\mathcal{E}^{uo}$ is defined as in Definition 9, derived from the general RPC envelopes $\mathcal{E}^g$ from Lemma 6.

Tighter UI Cardinal RPC Gadgets using Permutations. When a gadget is not uniformly cardinal RPC for a given general RPC envelope, we show that composing it with uniformly random permutations on its input and/or output share makes it uniformly cardinal RPC and yields tighter security bounds when composing it. Specifically, in Lemma 7, we show that applying an independent uniformly random permutation to the shares of each input sharing is sufficient to ensure the UI cardinal RPC property. Similarly, Lemma 8 establishes that independently permuting the shares of each output sharing guarantees the UO cardinal RPC property. As a direct corollary, applying independent uniformly random permutations to both the input and output shares of a gadget is sufficient to ensure the uniform cardinal RPC property.

Lemma 7. *Let $n, \ell, m \in \mathbb{N}$ and $p \in [0, 1]$. Let $\mathcal{E}^G$ represent a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$. Let $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ be an n -share $(p, \mathcal{E}^G)$ -general RPC gadget and $G \circ \sigma : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ denote the composition of G with σ , the concatenation of ℓ independent random permutations $(\sigma_1, \dots, \sigma_\ell) \in \mathcal{S}_n^\ell$ on the ℓ input n -sharings, as depicted in Figure 2a. Then $G \circ \sigma$ is $(p, \mathcal{E}^{G \circ \sigma})$ -general RPC with, for all $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$,*

$$\mathcal{E}_{J_1, \dots, J_m}^{G \circ \sigma}(I_1, \dots, I_\ell) = \frac{1}{\prod_{i=1}^\ell \binom{n}{|I_i|}} \cdot \sum_{\substack{I_1^*, \dots, I_\ell^* \subseteq [n] \\ |I_1^*| = |I_1|, \dots, |I_\ell^*| = |I_\ell|}} \mathcal{E}_{J_1, \dots, J_m}^G(I_1^*, \dots, I_\ell^*) \quad (1)$$

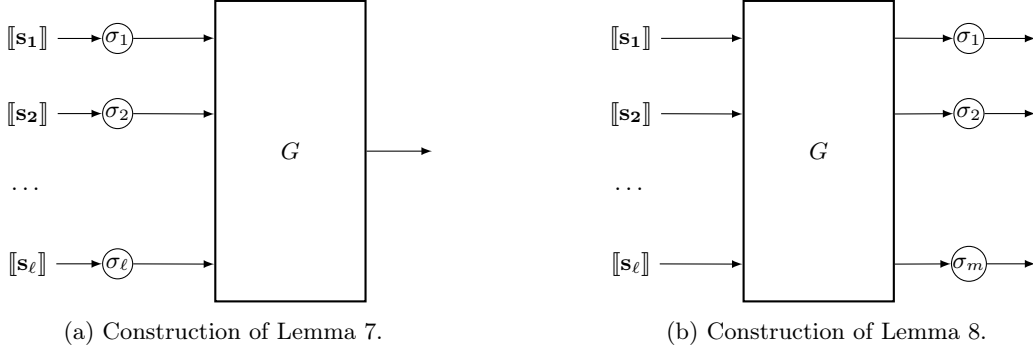


Fig. 2: Composing gadgets with random permutations.

In particular, $G \circ \sigma$ is $(p, \mathcal{E}^{ui})$ -UI cardinal RPC with the envelopes $\mathcal{E}^{ui}$ defined as follows. For any $(t_1^{in}, \dots, t_\ell^{in}) \subseteq [0, n]^\ell$ and $(t_1^{out}, \dots, t_m^{out}) \subseteq [0, n]^m$,

$$\mathcal{E}_{t_1^{out}, \dots, t_m^{out}}^{ui}(t_1^{in}, \dots, t_\ell^{in}) = \max_{\substack{J_1, \dots, J_m \\ |J_1|=t_1^{out}, \dots, |J_m|=t_m^{out}}} \mathcal{E}_{J_1, \dots, J_m}^{G \circ \sigma}(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{in}}$$

for any $I_1, \dots, I_\ell$ where $|I_1| = t_1^{in}, \dots, |I_\ell| = t_\ell^{in}$.

Lemma 8. Let $n, \ell, m \in \mathbb{N}$ and $p \in [0, 1]$. Let $\mathcal{E}^G$ represent a collection of probability envelopes indexed by $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$. Let $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ be an n -share $(p, \mathcal{E}^G)$ -general RPC gadget and $\sigma \circ G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ denote the composition of G with σ , the concatenation of m independent random permutations $(\sigma_1, \dots, \sigma_m) \in \mathcal{S}_n^m$ on the m output n -sharings, as depicted in Figure 2b. Then $\sigma \circ G$ is $(p, \mathcal{E}^{\sigma \circ G})$ -general RPC with, for all $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$,

$$\mathcal{E}_{J_1, \dots, J_m}^{\sigma \circ G}(I_1, \dots, I_\ell) = \frac{1}{\prod_{i=1}^m \binom{n}{|J_i|}} \cdot \sum_{\substack{J_1^*, \dots, J_m^* \subseteq [n] \\ |J_1^*|=|J_1|, \dots, |J_m^*|=|J_m|}} \mathcal{E}_{J_1^*, \dots, J_m^*}^G(I_1, \dots, I_\ell) \quad (2)$$

In particular, $\sigma \circ G$ is $(p, \mathcal{E}^{uo})$ -UO cardinal RPC with the envelopes $\mathcal{E}^{uo}$ defined as follows. For any $(t_1^{in}, \dots, t_\ell^{in}) \in [0, n]^\ell$ and $(t_1^{out}, \dots, t_m^{out}) \in [0, n]^m$,

$$\mathcal{E}_{t_1^{out}, \dots, t_m^{out}}^{uo}(t_1^{in}, \dots, t_\ell^{in}) = \sum_{\substack{I_1, \dots, I_\ell \\ |I_1|=t_1^{in}, \dots, |I_\ell|=t_\ell^{in}}} \mathcal{E}_{J_1, \dots, J_m}^{\sigma \circ G}(I_1, \dots, I_\ell).$$

To apply an independent uniformly random permutation to the shares, we use the Knuth-Yates shuffling technique, described in Appendix D. Informally, this process consists of iteratively swapping a randomly chosen *unprocessed* share with the last *unprocessed* share in the list. The chosen share is then marked as processed, reducing the pool of candidates for the next iteration. This continues until all shares have been processed, ensuring a uniform random permutation of the entire set. The whole procedure requires generating $n - 1$ random values, each drawn uniformly from a decreasing range $[i]$, for $2 \leq i \leq n$. Additionally, it may involve up to $3(n - 1)$ read/write operations.

4.4 Comparison of Random Probing Composability Notions

In this subsection, we compare the simulation-based composability notions recalled or introduced so far, emphasizing their respective strengths and limitations in terms of security tightness and verification efficiency.

- *General RPC* offers the tightest upper bound on the adversary’s advantage. However, this comes at a steep computational cost. The associated probability envelopes are defined over $\mathcal{P}([n])^\ell \times \mathcal{P}([n])^m$, with size $2^{n(\ell+m)}$, which quickly becomes infeasible to handle for large numbers of shares, inputs, or outputs.
- *Cardinal RPC* provides a looser upper bound than *general RPC*, but significantly improves verification time and memory complexity. Envelopes are defined over $[0, n]^\ell \times [0, n]^m$, yielding a size of $(n+1)^{\ell+m}$, polynomial in the number of shares. In practice, this means that while the bound is looser, larger values of n become tractable for larger circuits.
- *Uniformly cardinal RPC* built with uniformly random permutations is designed to strike a balance between the two extremes. It preserves the tightness of general RPC while benefiting from the lower complexity of cardinal RPC for the verification of compositions.
- *Threshold RPC* provides the loosest upper bound among the four notions but has the advantage of minimal verification overhead. It is particularly suitable in large circuits with high number of shares.

One-Stage Butterfly Network. To illustrate the practical implications of these trade-offs, we consider a one-stage butterfly network, a structure common in lattice-based cryptographic algorithms. As shown in Figure 3, this circuit takes 2 masked inputs and produces 2 masked outputs, each with n shares. It is built from three basic gadgets: one scalar multiplication cMult^γ , one addition Add^γ and one subtraction (identical to the addition gadget with minus signs).

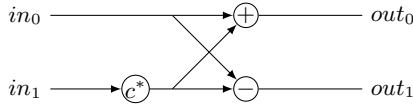


Fig. 3: 1-stage butterfly network.

We evaluate the (t, p) -threshold RPC security (see Definition 4) of this butterfly network circuit, where $t = \lfloor \frac{n}{2} \rfloor$ and using four different security analyses: (i) the general RPC envelopes; (ii) the UI cardinal RPC envelopes, assuming composition with uniformly random permutations; (iii) the cardinal RPC envelopes; and (iv) the *threshold RPC* bounds derived from the individual gadgets³. This evaluation is performed for several leakage rates, namely $p \in \{2^{-3}, 2^{-5}, 2^{-10}\}$ and different number of shares.

Initially, we observe that the circuit provides no security guarantees under the threshold RPC notion. This is because threshold RPC requires dedicated copy gadgets to securely duplicate sharings. Without them, duplicating a sharing would amount to summing two independent sets of t input shares, which exceeds the threshold and breaks the security guarantee. To enable a fair comparison, we therefore consider a modified version of the circuit (v2) in which a copy gadget (simply consisting in refreshing each branch) is inserted wherever a sharing is duplicated to feed multiple gadgets.

Figure 4 compares the threshold RPC advantages of the 1-stage butterfly network, computed under the different composition methods and circuit variants. In each subfigure, **gRPC**, **ucRPC**, **cRPC** correspond to the use of general, uniformly cardinal, and cardinal envelopes, respectively, to evaluate the global threshold RPC security. **tRPC** refers to the threshold RPC advantage, directly computed from the threshold RPC advantages of the individual gadgets. Labels **v1** to **v4** refer to: the original circuit (v1), the version with refresh gadgets at every wire split (v2), the original circuit with permutations at each copy (v3), and the version combining permutations and additional refreshes (v4).

Across all leakage rates p , we confirm that the threshold RPC composition yields weaker advantage even when additional refreshes are inserted (see, e.g. **tRPC_v2** vs. **cRPC_v1**). Then, the relative behavior of cardinal RPC, uniformly cardinal RPC and general RPC varies depending on the leakage rate:

³ For the general RPC, the UI cardinal RPC and the cardinal RPC composition scenarios, we derive the threshold RPC advantage from the final circuit envelope to provide a consistent comparison.

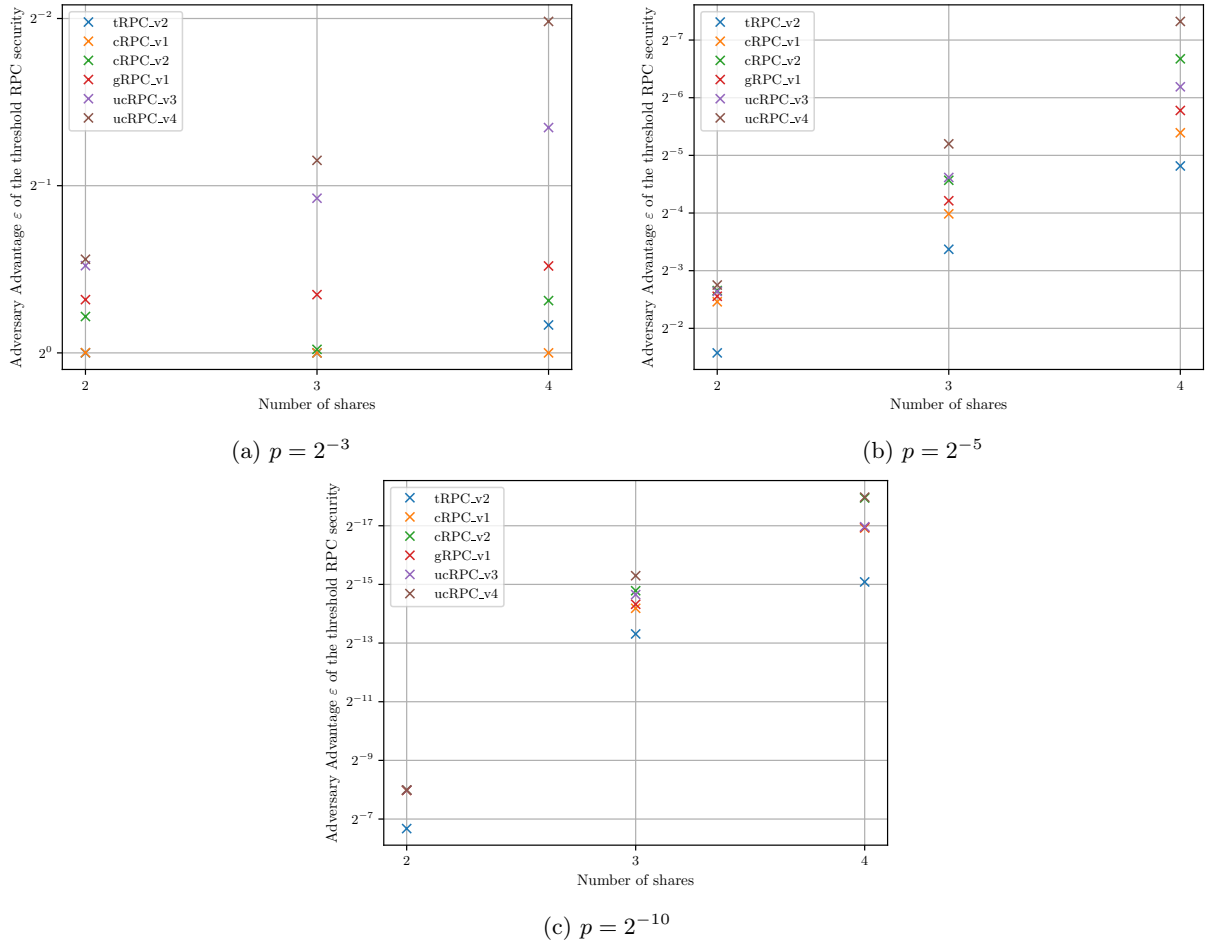


Fig. 4: Threshold RPC security of 1-stage butterfly network with respect to the number of shares n and the leakage rates p for different composition approaches.

- For low leakage rates (e.g., $p = 2^{-10}$), the security levels obtained under cardinal, general and uniformly cardinal RPC are very close, and the difference between the two circuit variants (with or without refresh) grows with the number of shares.
- At a moderate leakage rate ($p = 2^{-5}$), a small but noticeable gap begins to emerge with cardinal RPC, and the difference between the two circuit variants (with or without refresh) is more noticeable.
- When the leakage rate becomes high ($p = 2^{-3}$), the difference becomes significant with cardinal RPC and emerges between general and uniform cardinal RPC (the latter being computed on the variant with permutations). Not only does the uniformly cardinal RPC approach (with permutations) provide substantially better security guarantees than general and cardinal RPC, but the impact of using refresh gadgets (variants v2 and v4) also becomes evident, leading to improved security levels compared to the original circuit (variants v1 and v3).

These results highlight that uniformly cardinal RPC (on a variant using permutations) becomes increasingly advantageous over general and cardinal RPC (without permutations) as the leakage rate increases.

In addition, the verification of compositions based on the gadgets' general RPC envelopes does not scale efficiently to larger circuits⁴. To illustrate this limitation, a two-stage butterfly network is examined in Appendix E: an example where the general RPC composition already becomes intractable, while other methods remain practical.

5 Multiplication Gadget with Cardinal RPC Security

To build upon the work of [8], which introduced linear cardinal RPC gadgets, we aim to design a non-linear cardinal RPC multiplication gadget. Our approach builds on the proposal by [4], which is heuristically secure against horizontal attacks and is therefore presumed to offer strong security in the random probing model, despite lacking a formal analysis within that model. We first describe its generic design and demonstrate a general property on its global structure in Lemma 9. In Sections 5.1 and 5.2, we detail how we instantiate this design to facilitate evaluation and enhance its security features. Then, in Section 5.3, we analyze the trade-offs between security and complexity in our different implementations and compare our results with existing random probing multiplication schemes from the literature.

The original multiplication gadget from [4] consists of two steps. The first is the *expansion step* **MatMult**, which takes as inputs two n -sharings $\llbracket x \rrbracket, \llbracket y \rrbracket \in (\mathbb{K}^n)^2$ and outputs a matrix $(M_{i,j})_{1 \leq i,j \leq n}$ where each coefficient $M_{i,j}$ represents a cross-product $x_i \cdot y_j$ masked with random values summing to zero, ensuring that $\sum_{i,j=1}^n M_{i,j} = \sum_{i,j=1}^n x_i \cdot y_j = x \cdot y$. The second step, the *compression step*, converts the resulting n^2 -sharing of $x \cdot y$ into an n -sharing.

Algorithm 1 Two-step Multiplication Gadget

Input: two n -sharings $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$
Output: n -sharing $\llbracket z \rrbracket$ such that $z = x \cdot y$.
 $(M_{i,j})_{1 \leq i,j \leq n} \leftarrow \text{Expand}(\llbracket x \rrbracket, \llbracket y \rrbracket)$
 $\llbracket z \rrbracket \leftarrow \text{Comp}((M_{i,j})_{1 \leq i,j \leq n})$
return $\llbracket z \rrbracket$

Our multiplication gadget follows the same two-step structure. Both the expansion and compression steps are adapted variants of those in [4], while preserving their key properties. The expansion step, referred to as **Expand**, produces an n^2 -sharing of $x \cdot y$, and the compression step, referred to as **Comp**, yields an n -sharing of $x \cdot y$. The high-level design is given in Algorithm 1.

Lemma 9 establishes the cardinal RPC security of the high-level multiplication gadget described in Algorithm 1 based on the cardinal RPC security of its two components **Expand** and **Comp**. Its proof directly follows from the composition result in [8, Lemma 4].

Lemma 9. *Let $n \in \mathbb{N}$. Let $p \in [0, 1]$. We assume that **Expand** is $(p, \mathcal{E}^{\text{Expand}})$ -cardinal RPC and **Comp** is $(p, \mathcal{E}^{\text{Comp}})$ -cardinal RPC. Then the multiplication gadget described in Algorithm 1 is $(p, \mathcal{E}^{\text{mult}})$ -cardinal RPC such that for all $t_1^{in}, t_2^{in}, t^{out} \in [0, n]$, we have:*

$$\mathcal{E}_{t^{out}}^{\text{mult}}(t_1^{in}, t_2^{in}) = \sum_{k=0}^{n^2} \mathcal{E}_k^{\text{Expand}}(t_1^{in}, t_2^{in}) \cdot \mathcal{E}_{t^{out}}^{\text{Comp}}(k)$$

Remark 1. Note that we could partition the output of **Expand** (or equivalently, the input of **Comp**) into multiple sharings to obtain tighter security bounds. Specifically, instead of considering the required number of shares among n^2 shares, we would analyze α sub-cardinals among n^2/α shares, leading to a more precise estimation (for α a divisor of n^2). However, increasing α not only tightens the bounds but also significantly

⁴ Note that this scalability issue only concerns the verification of the composition. In our scripts, the computation of the gadgets' envelope for the different notions among general RPC, (uniformly) cardinal RPC and threshold RPC has almost the same complexity as we derive them all from the general RPC envelope.

Algorithm 2 MatMult from [4]

Input: n_x -sharing $\llbracket x \rrbracket$ and n_y -sharing $\llbracket y \rrbracket$

Output: $(n_x \cdot n_y)$ -sharing $M = (M_{i,j})_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}}$ such that $\sum_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}} M_{i,j} = x \cdot y$

```

if  $(n_x, n_y) = (1, 1)$  then return  $(x_1 \cdot y_1)$ 
if  $(n_x, n_y) = (1, 2)$  then return  $(x_1 \cdot y_1 \ x_1 \cdot y_2)$ 
if  $(n_x, n_y) = (2, 1)$  then return  $\begin{pmatrix} x_1 \cdot y_1 \\ x_2 \cdot y_1 \end{pmatrix}$ 
if  $(n_x, n_y) = (2, 2)$  then return  $\begin{pmatrix} x_1 \cdot y_1 & x_1 \cdot y_2 \\ x_2 \cdot y_1 & x_2 \cdot y_2 \end{pmatrix}$ 
 $\mathbf{x}^{(1)} \leftarrow (x_1, \dots, x_{\lfloor \frac{n_x}{2} \rfloor}); \mathbf{x}^{(2)} \leftarrow (x_{\lceil \frac{n_x}{2} \rceil}, \dots, x_{n_x});$ 
 $\mathbf{y}^{(1)} \leftarrow (y_1, \dots, y_{\lfloor \frac{n_y}{2} \rfloor}); \mathbf{y}^{(2)} \leftarrow (y_{\lceil \frac{n_y}{2} \rceil}, \dots, y_{n_y})$ 
 $\mathbf{x}'^{(1)} \leftarrow G_{\text{ref}}(\mathbf{x}^{(1)}); \mathbf{x}'^{(2)} \leftarrow G_{\text{ref}}(\mathbf{x}^{(2)}); \mathbf{y}'^{(1)} \leftarrow G_{\text{ref}}(\mathbf{y}^{(1)}); \mathbf{y}'^{(2)} \leftarrow G_{\text{ref}}(\mathbf{y}^{(2)})$ 
return  $\begin{pmatrix} \text{MatMult}(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}) & \text{MatMult}(\mathbf{x}^{(1)}, \mathbf{y}^{(2)}) \\ \text{MatMult}(\mathbf{x}^{(2)}, \mathbf{y}^{(1)}) & \text{MatMult}(\mathbf{x}^{(2)}, \mathbf{y}^{(2)}) \end{pmatrix}$ 

```

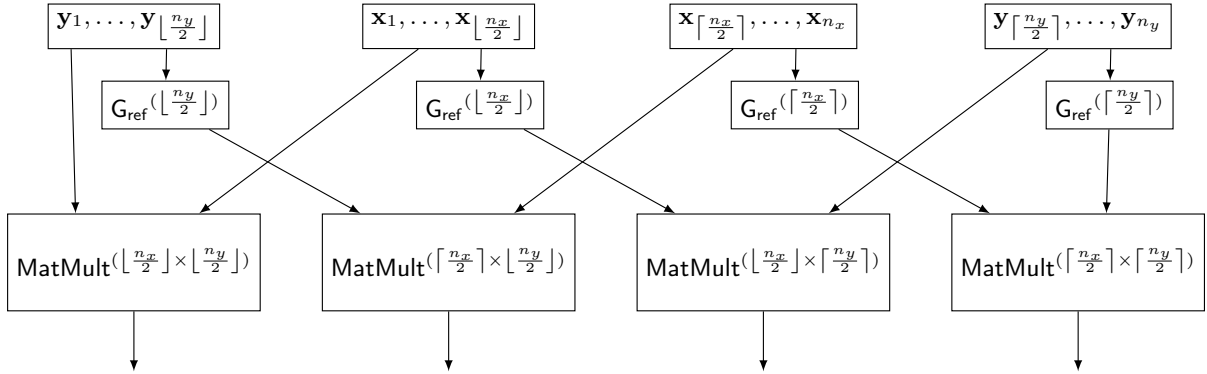


Fig. 5: Illustration of MatMult.

increases the complexity of computing the envelopes, requiring an enumeration over $(\frac{n^2}{\alpha} + 1)^\alpha$ instead of over $n^2 + 1$.

5.1 Refining MatMult: New Variants

In [4], the expansion step **Expand** is instantiated with **MatMult** which is designed as follows. Each n -sharing $\llbracket x \rrbracket$ and $\llbracket y \rrbracket$ is first split into two parts, resulting in two $\lfloor \frac{n}{2} \rfloor$ -sharings and two $\lceil \frac{n}{2} \rceil$ -sharings. Each of these sub-sharings is then duplicated into two versions: one remains unchanged (identity copy), while the other undergoes a refresh gadget, denoted G_{ref} , to introduce additional randomness. **MatMult** is then applied recursively to each of the four resulting pairs, with one instance made of a sub-sharing of x and one instance made of a sub-sharing of y , each copy being used only once. Once n reaches one or two, the algorithm directly computes the corresponding sub-products between the two 1- or 2- sharings.

We recall the full procedure in Algorithm 2, for any $n \in \mathbb{N}^*$. An illustration is also provided in Figure 5. Note that recursive calls may involve inputs of different lengths, denoted n_x and n_y , when n is not a power of two. However, starting from equal input sizes $n_x = n_y = n$, it is easy to show by induction that the invariant $|n_x - n_y| \leq 1$ holds throughout the recursion. The authors of [4] instantiate G_{ref} using the quasi-linear refresh gadget introduced in the same paper, which optimizes both computational complexity and the number of required random values.

In the following, we present three new variants of **MatMult**, designed to improve security bounds.

Variant 1: MatMultRef. We first apply transformations to **MatMult** to facilitate the computation of its cardinal RPC envelope. Specifically, we replace the quasi-linear refresh gadget with the one introduced in [8], as its cardinal RPC envelope is established. We refer to this modified version as **MatMultRef**.

Building upon this initial construction, we explore two alternative trade-offs between security and performance through the following variants:

Variant 2: MatMultSym refreshes each sub-sharing before every recursive call, leading to twice as many refresh gadgets as in **MatMultRef**, as initially suggested in [13,19]. **MatMultSym** is formally described in Appendix F.

Variant 3: UnifMatMult introduces uniformly random permutations at key locations in the circuit to improve the security bounds when duplicating shared values. Specifically, for each parallel composition of two gadgets that share the same input sharing, a random permutation is applied to the input sharing of one of them to ensure it satisfies the UI cardinal RPC property, thereby transforming the composition into a Tree-UI gadget (see Lemma 3). A formal description of **UnifMatMult** is provided in Appendix F. Note that we do not add random permutations at the output of intermediate gadgets, however, since these already satisfy the UO cardinal RPC property by construction.

Complexity. Regarding the complexity of the three variants of **MatMult**, as the number of random values used, denoted $C_{\text{rand}}(n_x, n_y)$, as well as the number of additions, denoted $C_{\text{add}}(n_x, n_y)$, and the number of multiplications $C_{\text{mult}}(n_x, n_y)$ can be computed by recursively. We note respectively C_*^1 , C_*^2 , C_*^3 the complexity of, respectively, variants **MatMultRef**, **MatMultSym** and **UnifMatMult**. We denote by $\gamma(n)$ the number of random values used in **RPrefresh** when applied to an input sharing of size n , allowing this number to depend on n . We have

$$\begin{aligned} C_{\text{rand}}^1(n_x, n_y) &= 2\gamma(n_x) + 2\gamma(n_y) \\ &\quad + C_{\text{rand}}^1(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor) + C_{\text{rand}}^1(\lfloor \frac{n_x}{2} \rfloor, \lceil \frac{n_y}{2} \rceil) \\ &\quad + C_{\text{rand}}^1(\lceil \frac{n_x}{2} \rceil, \lfloor \frac{n_y}{2} \rfloor) + C_{\text{rand}}^1(\lceil \frac{n_x}{2} \rceil, \lceil \frac{n_y}{2} \rceil) \\ C_{\text{add}}^1(n_x, n_y) &= 4\gamma(n_x) + 4\gamma(n_y) + n_x + n_y \\ &\quad + C_{\text{add}}^1(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor) + C_{\text{add}}^1(\lfloor \frac{n_x}{2} \rfloor, \lceil \frac{n_y}{2} \rceil) \\ &\quad + C_{\text{add}}^1(\lceil \frac{n_x}{2} \rceil, \lfloor \frac{n_y}{2} \rfloor) + C_{\text{add}}^1(\lceil \frac{n_x}{2} \rceil, \lceil \frac{n_y}{2} \rceil) \end{aligned}$$

where we have

$$\begin{aligned} C_{\text{rand}}^1(1, 1) &= C_{\text{rand}}^1(1, 2) = C_{\text{rand}}^1(2, 1) = C_{\text{rand}}^1(2, 2) = 0 \\ C_{\text{add}}^1(1, 1) &= C_{\text{add}}^1(1, 2) = C_{\text{add}}^1(2, 1) = C_{\text{add}}^1(2, 2) = 0 \end{aligned}$$

Furthermore, the number of multiplications used in **MatMultRef** is given by $C_{\text{mult}}^1(n_x, n_y) = n_x \cdot n_y$. For the two other variants, we have

$$\begin{aligned} C_{\text{rand}}^2(n_x, n_y) &= C_{\text{rand}}^3(n_x, n_y) = 2 \cdot C_{\text{rand}}^1(n_x, n_y) \\ C_{\text{add}}^2(n_x, n_y) &= C_{\text{add}}^3(n_x, n_y) = 2 \cdot C_{\text{add}}^1(n_x, n_y) \\ C_{\text{mult}}^2(n_x, n_y) &= C_{\text{mult}}^3(n_x, n_y) = C_{\text{mult}}^1(n_x, n_y) \end{aligned}$$

Finally, we compute the number of permutations $\text{NbP}(n_x, n_y, k)$ of size $k > 1$ used in **UnifMatMult** when called with $n_x = n_y = n$, recursively as follows.

If $k > \max(n_x, n_y)$, then:

$$\text{NbP}(n_x, n_y, k) = 0.$$

Otherwise:

$$\begin{aligned} \text{NbP}(n_x, n_y, k) = & \text{NbP}\left(\left\lfloor \frac{n_x}{2} \right\rfloor, \left\lfloor \frac{n_y}{2} \right\rfloor, k\right) + \text{NbP}\left(\left\lceil \frac{n_x}{2} \right\rceil, \left\lfloor \frac{n_y}{2} \right\rfloor, k\right) \\ & + \text{NbP}\left(\left\lfloor \frac{n_x}{2} \right\rfloor, \left\lceil \frac{n_y}{2} \right\rceil, k\right) + \text{NbP}\left(\left\lceil \frac{n_x}{2} \right\rceil, \left\lceil \frac{n_y}{2} \right\rceil, k\right) \\ & + 2 \cdot \left(\mathbf{1}_{k \in \{\lfloor \frac{n_x}{2} \rfloor, \lceil \frac{n_x}{2} \rceil\}} + \mathbf{1}_{k \in \{\lfloor \frac{n_y}{2} \rfloor, \lceil \frac{n_y}{2} \rceil\}} \right). \end{aligned}$$

Table 1 summarizes the complexity in terms of number of operations and number of random values of all the gadgets used in our three multiplication gadget variants: **MatMultAsym**, **MatMultSym**, and **UnifMatMult**, and in **TreeAdd**, with respect to the number of shares n , assumed to be a power of two.

As explained in Section 3.1, it is important to distinguish the two types of randomness involved in our gadgets. Specifically, the randomness required for generating random indexes and random permutations is fundamentally different from that used in random gates. In practice, random gates must be implemented in a *secure* manner to limit information leakage, as characterized by the leakage probability p . In contrast, the random bits used for indexes and permutations do not require any protections. Consequently, generating $\log_2(|\mathbb{K}|)$ bits for indexes or permutations is typically much less costly than a gate producing a random element of $\mathbb{K}$. Therefore, we account for these two types of randomness separately.

Regarding the count of random bits, recall that the **RPrefresh** gadget requires, at each iteration, the selection of two distinct indices: one from $[n]$ and another from $[n-1]$. These selections can be performed efficiently using approximately $\gamma(n) \cdot (\log_2 n + \log_2(n-1))$ random bits per iteration. Similarly, in **UnifMatMult**, each permutation involves $n-1$ random values, where the i -th value is sampled uniformly from $[i]$ for $2 \leq i \leq n$. Since sampling from an interval of size k typically requires around $\log_2(k)$ random bits, the overall cost corresponds to the sum of these logarithmic terms. The total number of such unprotected random bits used in both the **RPrefresh** gadgets and the input permutations of **UnifMatMult** is reported in the last column of the table.

The implementation of permutations may also incur additional read/write (R/W) operations, which are not included in the table as they are implementation-dependent. However, as a rough estimate, each permutation of size k may involve up to $3k$ R/W operations. The total number of such permutations is given by $\text{NbP}(n, n, k)$.

Table 1: Complexity of gadgets as a function of n .

Gadget	#Randoms (field)	#Additions	#Mult.	#Random (bits)
RPrefresh	$\gamma(n)$	$2\gamma(n) + n$	0	$\gamma(n) \cdot (\log_2(n) + \log_2(n-1))$
Add	$2\gamma(n)$	$4\gamma(n) + 3n$	0	$2\gamma(n) \cdot (\log_2(n) + \log_2(n-1))$
MatMultAsym	$\sum_{\ell=1}^{\log_2(n)-1} 4^\ell \gamma(\frac{n}{2^\ell})$	$\sum_{\ell=1}^{\log_2(n)-1} 4^\ell (2\gamma(\frac{n}{2^\ell}) + \frac{n}{2^\ell})$	n^2	$\sum_{\ell=1}^{\log_2(n)-1} 4^\ell \gamma(\frac{n}{2^\ell}) (\log_2(\frac{n}{2^\ell}) + \log_2(\frac{n}{2^\ell} - 1))$
MatMultSym	$\sum_{\ell=1}^{\log_2(n)-1} 8^\ell \gamma(\frac{n}{2^\ell})$	$\sum_{\ell=1}^{\log_2(n)-1} 8^\ell (2\gamma(\frac{n}{2^\ell}) + \frac{n}{2^\ell})$	n^2	$\sum_{\ell=1}^{\log_2(n)-1} 8^\ell \gamma(\frac{n}{2^\ell}) (\log_2(\frac{n}{2^\ell}) + \log_2(\frac{n}{2^\ell} - 1))$
UnifMatMult	$\sum_{\ell=1}^{\log_2(n)-1} 8^\ell \gamma(\frac{n}{2^\ell})$	$\sum_{\ell=1}^{\log_2(n)-1} 8^\ell (2\gamma(\frac{n}{2^\ell}) + \frac{n}{2^\ell})$	n^2	$\sum_{\ell=1}^{\log_2(n)-1} 8^\ell \gamma(\frac{n}{2^\ell}) (\log_2(\frac{n}{2^\ell}) + \log_2(\frac{n}{2^\ell} - 1)) + 4^\ell \sum_{k=2}^{n/2^\ell} \log_2(k)$
TreeAdd	$2\gamma(n-1)$	$(n-1) \cdot (4\gamma(n) + 3n)$	0	0

Security. Regarding the cardinal RPC security of the three **MatMult** variants, the corresponding envelopes can be computed recursively. They are provided, along with a detailed description of the computation method, in Appendix F. All scripts are open-source at

<https://github.com/CryptoExperts/AC25-cardinal-rp-compiler>.

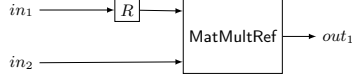


Fig. 6: A recursive call of **MatMultRef**.

Below, we briefly outline the approach used for **MatMultRef**; the computations for **MatMultSym** and **UnifMatMult** follow a similar structure.

Assuming we are not in the base case, **MatMultRef** gadget makes four recursive calls to itself, each involving a sub-sharing of x and a sub-sharing of y . In each call, one of the sub-sharings is refreshed beforehand, as shown in Figure 6. Given the cardinal RPC envelopes for the refresh gadget and for the recursive call to **MatMultRef**, we can derive the cardinal RPC envelope of the overall structure in Figure 6 by applying [8, Lemma 4]. This provides us with four recursive cardinal RPC envelopes. To obtain the global cardinal RPC envelope, we must take the union of the required shares across the four recursive calls. However, this step is non-trivial since each sub-sharing appears in two of the recursive calls, meaning that a single share may be required by multiple calls. Consequently, we must account for the probability of all possible cases when constructing our envelopes.

5.2 Refining Comp: a New Variant

The original **Comp** from [4] does not follow a recursive structure, making it difficult to efficiently compute the envelopes of the compression step. To address this issue, we propose a new functionally equivalent **Comp** gadget, denoted **TreeAdd**, for which we are able to compute the envelopes efficiently. The choice of a tree of additions then naturally arises for its efficiency and its recursive structure.

Algorithm 3 describes **TreeAdd**, which implements a tree structure of $(n, 2\text{-to-1})$ addition gadgets Add^γ (as defined in Algorithm 6) to compress the n n -sharings composing M into a single n -sharing.

Algorithm 3 **TreeAdd**

Input : $(M_{i,j})_{\substack{1 \leq i \leq k \\ 1 \leq j \leq n}}$ k n -sharings.

Output : An n -sharing of $\sum_{i=1}^k \sum_{j=1}^n M_{i,j}$.

```

if  $k = 1$  then return  $(M_{1,j})_{1 \leq j \leq n}$ 
if  $k = 2$  then return  $\text{Add}^\gamma((M_{1,j})_{1 \leq j \leq n}, (M_{2,j})_{1 \leq j \leq n})$ 
 $\llbracket z_1 \rrbracket \leftarrow \text{TreeAdd}((M_{i,j})_{\substack{1 \leq i \leq \lfloor \frac{k}{2} \rfloor \\ 1 \leq j \leq n}}); \llbracket z_2 \rrbracket \leftarrow \text{TreeAdd}((M_{i,j})_{\substack{\lceil \frac{k}{2} \rceil \leq i \leq n \\ 1 \leq j \leq n}})$ 
return  $\text{Add}^\gamma(\llbracket z_1 \rrbracket, \llbracket z_2 \rrbracket)$ 

```

The authors of [8] established the cardinal RPC security of Add^γ based on the cardinal RPC security of the underlying refresh gadget RRefresh^γ . In particular, they demonstrated how to compute $\mathcal{E}^{\text{add}}$ from the envelope of the refresh gadget. Building on these results, the envelopes of the cardinal RPC security $\mathcal{E}^{\text{TA},k}$ of **TreeAdd** can be computed inductively from $\mathcal{E}^{\text{TA}, \lfloor \frac{k}{2} \rfloor}$, $\mathcal{E}^{\text{TA}, \lceil \frac{k}{2} \rceil}$ and $\mathcal{E}^{\text{add}}$. This approach is formalized in Lemma 10, whose proof is directly derived from the composition lemma in [8, Lemma 4].

Lemma 10. *Let $k, n \in \mathbb{N}^*$, $k \leq n$. Let $p \in [0, 1]$ the leakage rate. Assume **TreeAdd** with $\lfloor \frac{k}{2} \rfloor \cdot n$ (resp. $\lceil \frac{k}{2} \rceil \cdot n$) input shares is $(p, \mathcal{E}^{\text{TA}, \lfloor \frac{k}{2} \rfloor})$ -cardinal RPC (resp. $(p, \mathcal{E}^{\text{TA}, \lceil \frac{k}{2} \rceil})$ -cardinal RPC) and Add^γ is $(p, \mathcal{E}^{\text{add}})$ -cardinal RPC. Then **TreeAdd** with $k \cdot n$ input shares is $(p, \mathcal{E}^{\text{TA},k})$ -cardinal RPC where, for all $t^{in} \in [0, k \cdot n]$, all*

$t^{out} \in [0, n]$, and all $k > 2$:

$$\mathcal{E}_{t^{out}}^{TA,k}(t^{in}) = \sum_{i_1=\max(0, t^{in}-\lceil \frac{k}{2} \rceil \cdot n)}^{\min(t^{in}, \lfloor \frac{k}{2} \rfloor \cdot n)} \sum_{\ell_1=0}^n \sum_{\ell_2=0}^n \mathcal{E}_{\ell_1}^{TA, \lfloor \frac{k}{2} \rfloor}(i_1) \cdot \mathcal{E}_{\ell_2}^{TA, \lceil \frac{k}{2} \rceil}(t^{in} - i_1) \cdot \mathcal{E}_{t^{out}}^{add}(\ell_1, \ell_2) \quad (3)$$

with

$$\mathcal{E}^{TA,(2)} = \mathcal{E}^{add} \quad \text{and} \quad \mathcal{E}_{t^{out}}^{TA,(1)}(t^{in}) = \begin{cases} 0 & \text{if } t^{in} \neq t^{out} \\ 1 & \text{otherwise} \end{cases}$$

5.3 Results and Comparison

Building on the results from the previous two subsections and Lemma 9, we can derive the cardinal RPC property of our multiplication gadget, denoted **CardSecMult**, built from Algorithm 1. This construction instantiates **Expand** with one of the three gadgets **MatMultRef**, **MatMultSym** or **UnifMatMult**, and **Comp** with **TreeAdd**. From the cardinal RPC property, we further derive the threshold RPC advantage for **CardSecMult** using [8, Lemma 1].

Figure 7 illustrates the impact of varying the number γ of iterations of the underlying refresh gadget from [8], on the simulation failure probability ε in the (p, t, ε) -threshold RPC model. We consider several scenarios with $n = 4$ or $n = 8$ shares and a leakage probability p among 2^{-12} and 2^{-6} , always with $t = \frac{n}{2}$. The legend distinguishes between three instantiations of **CardSecMult**: with **MatMultRef** (*Asym*), **MatMultSym** (*Sym*), and **UnifMatMult** (*Uni*). Dotted lines indicate a lower bound on the simulation failure probability achievable in each case, given by the probability to observe strictly more than t input shares.

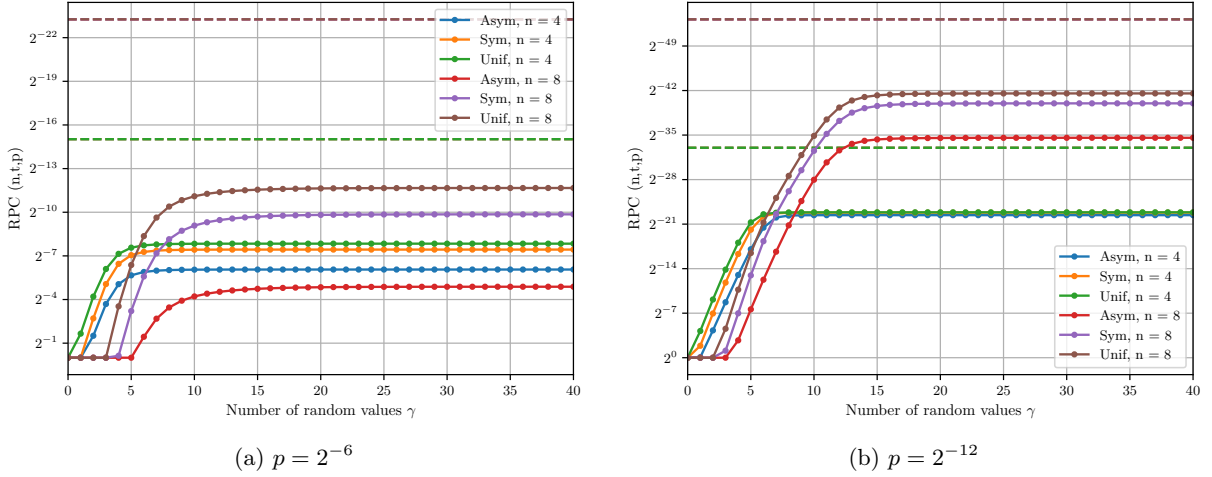


Fig. 7: Simulation failure probability ε of the (p, t, ε) -threshold RPC security for $n \in \{4, 8\}$, $t = \frac{n}{2}$, $p \in \{2^{-12}, 2^{-6}\}$.

For each version of **CardSecMult**, the simulation failure probability decreases as γ increases, until it reaches a threshold beyond which further gains become marginal. This threshold value of γ , which depends on both n and p , hence offers a good trade-off between performance and security. Among the three variants, **CardSecMult** instantiated with **UnifMatMult** consistently achieves the lowest failure probability, thanks to its

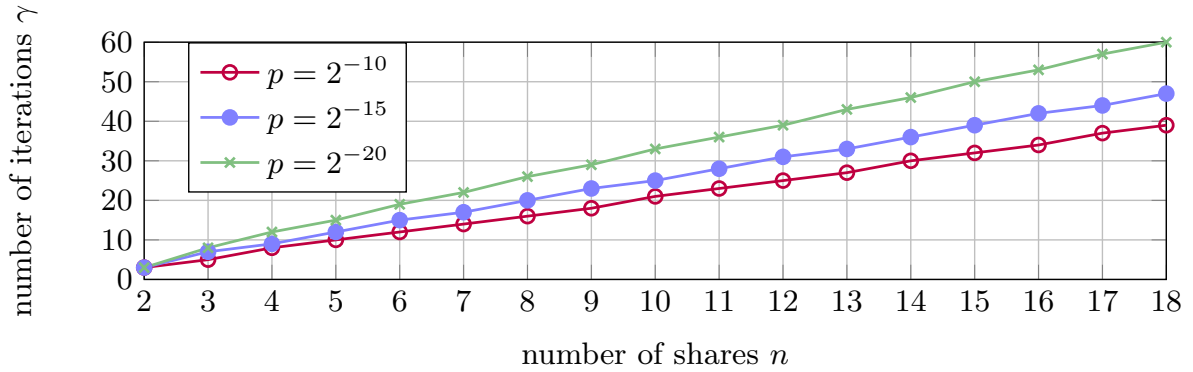


Fig. 8: Number of γ values to reach the threshold for a number of shares $n = 2$ to 18 and $p = 2^{-15}$.

additional refresh steps and the use of permutations. This advantage becomes increasingly noticeable as the wire leakage probability p grows.

Note that in the **MatMultRef** (Asym) implementation (based on the original **Expand** algorithm from [4]), the security level tends to decrease as the number of shares increases, particularly under high leakage probabilities (e.g. $p = 2^{-6}$). This can be attributed to the increased reuse of certain shares when n grows, as some of them are not refreshed before being involved in recursive calls. This observation highlights the importance of refreshing all propagation paths, as implemented in **MatMultSym** (Sym), and subsequently in **UnifMatMult** (Uni).

Remark 2. Following Remark 1, the cardinal RPC envelopes used to compute the security advantage in Figure 7 split the output of variants of **MatMult** into four parts and similarly divide the inputs of **TreeAdd** by four (i.e. $\alpha = 4$). This is feasible when n is a power of two, and thus 4 divides n^2 . While this optimization is applied in Figure 7, it will not be used in the following sections and benchmarks, as it significantly increases the verification complexity and can not be used with any n .

Fine-tuning of the RPrefresh gadget. We adopt a strategy to finely tune the value of γ (the number of randoms) used in the RPrefresh gadget. Specifically, we adjust γ based on the number of shares involved in each recursive invocation of the **UnifMatMult** algorithm. As discussed earlier, Figure 7 highlights the existence of a threshold γ that offers an optimal trade-off between security and computational complexity. Accordingly, for a given number of shares n and leakage rate p , our goal is to identify this threshold value of γ . To do so, we fix a threshold δ and exhaustively test all values γ until the random probing security advantage ε_γ of **UnifMatMult** is such that

$$|\log_2(\varepsilon_\gamma) - \log_2(\varepsilon_{\gamma+1})| < \delta .$$

By choosing a small value of δ (concretely we used $\delta = 0.05$ in our experiments), we can approximate this threshold γ . We apply this strategy for a range of number of shares n . In this way, at each recursive call of **UnifMatMult**, we use the threshold value γ identified for the recursive parameters n_x and n_y . These values are generally smaller than the threshold γ required for the full **UnifMatMult** instance, which improves complexity while merely preserving the same level of security. This adaptive choice of γ allows us to achieve better security-performance trade-offs. The specific number of iterations γ used for the refresh gadget is detailed in Figure 8.

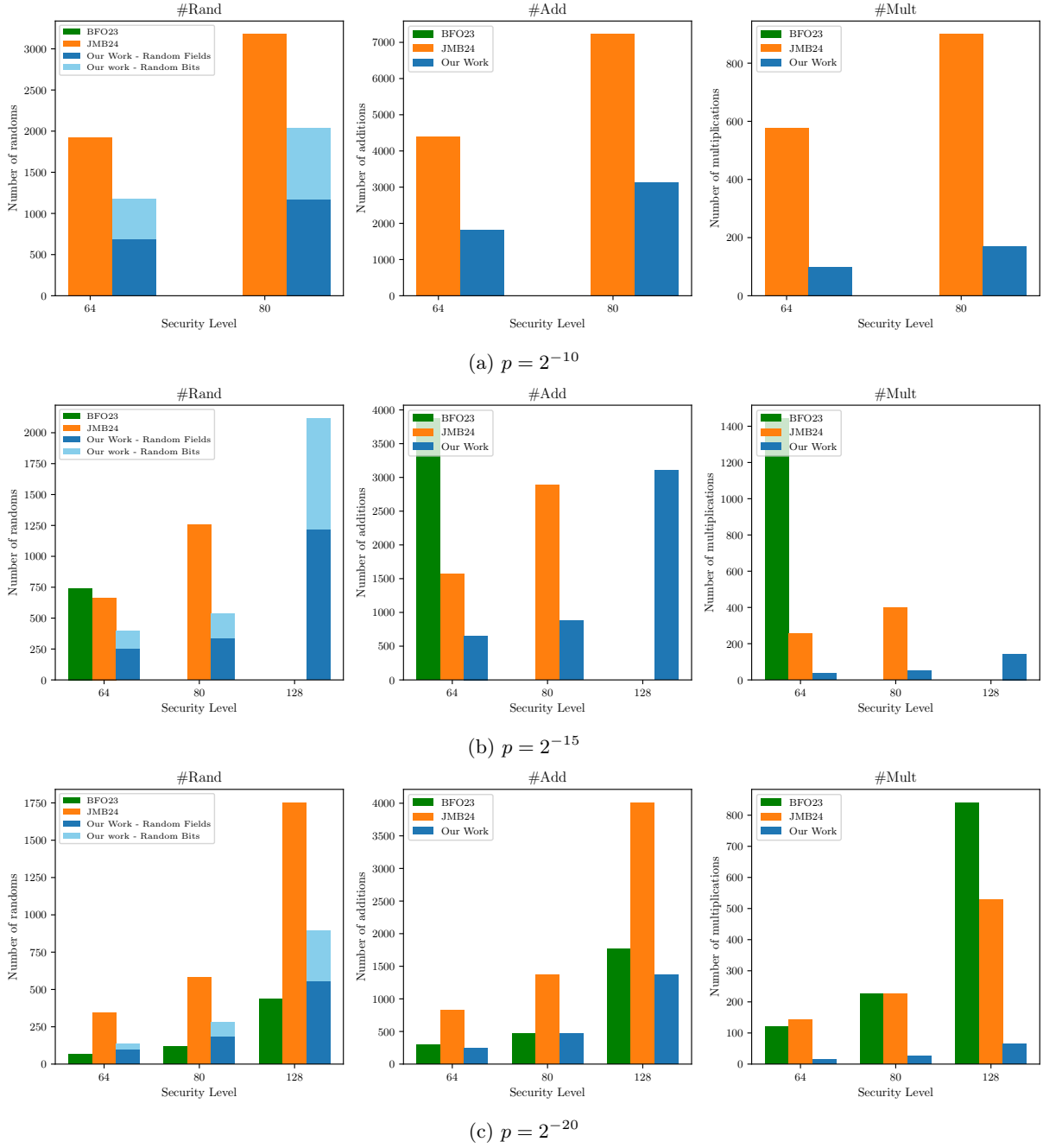


Fig.9: Complexity comparison of three multiplication gadgets with random probing security of $\varepsilon = 2^{-64}, 2^{-80}, 2^{-128}$.

Comparison with the state of the art. Hereafter, we compare our results of security and complexity with the two other state-of-the-art multiplication gadgets for which the random probing security has been previously established⁵:

⁵ We omit comparisons with earlier approaches, such as the line of work on random probing expansion, as their trade-offs are expected to be significantly less favorable.

1. The multiplication gadget from [11], referred to as BFO23, is based on the ISW multiplication gadget [18] which follows the same two-step structure as Algorithm 1. For two n -sharing $\llbracket x \rrbracket, \llbracket y \rrbracket$, **Expand** computes for any $i, j \in [n], i \neq j$ the cross-products $x_i \cdot y_j$ and $x_j \cdot y_i$, add them together with a random value $r_{i,j}$ in one share and store the random value in another share of the n^2 -sharing. If $i = j$, then it computes the cross-product $x_i \cdot y_i$ and store it in a share. The **Compress** step consists in a tree of additions which outputs an n -sharing.
2. The multiplication gadget proposed in [19], referred to as JMB24, adopts the same two-step structure described in Algorithm 1. For the **Expand** phase, it matches our second variant, **MatMultSym**, incorporating a refresh of each sub-sharing before every recursive call to ensure stronger security guarantees. For the **Comp** phase, it retains the original design introduced in [4].

Since both of these works provide analytical bounds on random probing security, we can directly compare our results with theirs. Indeed, a bound on the random probing security of our gadgets can be derived from their cardinal RPC security. In fact, if our multiplication gadget is $(p, \mathcal{E})$ cardinal RPC then it is (p, ε) random probing secure with

$$\varepsilon = \sum_{\substack{t_1^{\text{in}}, t_2^{\text{in}} \in [0, n] \\ t_1^{\text{in}} = n \text{ or } t_2^{\text{in}} = n}} \mathcal{E}_0(t_1^{\text{in}}, t_2^{\text{in}}).$$

Building on this equality, we compare in Figure 9 the complexity of the different multiplication designs for a leakage rate $p \in \{2^{-10}, 2^{-15}, 2^{-20}\}$ and a target random probing security level $\varepsilon \in \{2^{-64}, 2^{-80}, 2^{-128}\}$. For the randomness complexity of our gadget, we further account for the random bits used in selecting random share indices during the refresh, as well as those used for the random permutations (highlighted in light blue). As previously discussed, this form of randomness is likely to be less costly in practice than the randomness required for implementing random gates.

BF023 [11] is absent from several scenarios, as it fails to achieve the selected security levels for some of the tested leakage rates, regardless of the number of shares $n \in \mathbb{N}$. Similarly, the analytical formula provided in JMB24 [19] is only valid up to $n = 30$ shares, which is insufficient to reach some target security levels when the leakage rate is low. Consequently, some scenarios include both comparisons, others only JMB24, and a few rely exclusively on our own results.

As illustrated in the figure, our proposal generally achieves significantly better complexity than existing state-of-the-art approaches, except in a few scenarios with low leakage rates and low target security levels, where the required amount of randomness is comparable to that of previous works.

6 Compiler and Application to AES

6.1 Description of the Compiler

Our proposed circuit compiler features the following gadgets:

- *Addition and multiplication by a constant*: Add^γ and cMult^γ from [8], described in Algorithm 6 and 7;
- *Multiplication*: The **UnifMatMult** gadget presented in Section 5;
- *Addition of a constant*: The linear gadget cAdd^γ adds a constant $c \in \mathbb{K}$ by first refreshing the inputs, then adding c to the first share. It is formally described in Algorithm 8 and its cardinal RPC security is provided in Lemma 11 (see Appendix C.2);
- *Squaring (characteristic 2 only)*: In the case where $\mathbb{K} = \mathbb{F}_{2^k}$ with $k \in \mathbb{N}^*$ (as for the application to AES), we define a squaring gadget Sq^γ , which takes as input an n -sharing, refreshes it, and squares each share component-wise. It is formalized in Algorithm 9 in Appendix C.2, and its cardinal RPC security can be computed similarly to that of cMult^γ .

Moreover, our compiler inserts random permutations and refresh gadgets in targeted regions of the circuit to deal with copies. Specifically, we consider two situations:

1. *Fan-out 2 setting*: When an n -sharing serves as input to exactly two subsequent gadgets, we simply apply a permutation before the input of one of the two gadgets. This approach preserves the *UI-cardinal RPC* property, as established in Lemma 3.
2. *Fan-out greater than 2 setting*: A dedicated procedure is required to safely reuse the same sharing across multiple gadgets. Without appropriate safeguards, repeated use of the same sharing increases the probability of leakage from each individual share, thereby significantly weakening the overall random probing security. To address this pitfall, whenever a sharing $\llbracket x \rrbracket^{(0)}$ is used more than twice (we cannot technically avoid the use of two copies of a single sharing), we use a refresh gadget to securely duplicate the data. Specifically, the sharing $\llbracket x \rrbracket^{(0)}$ is used directly in the first gadget, after which it is refreshed to produce a new independent sharing $\llbracket x \rrbracket^{(1)}$. Like in the fan-out 2 setting, a permutation is applied before the first gadget or before the refreshing. If no further duplications are required, the refreshed copy can also be used directly in the second and the third gadgets, with a permutation inserted in one of the branches (as in the fan-out 2 setting). Otherwise, the procedure continues recursively: one copy is used directly, and the other is refreshed again to produce $\llbracket x \rrbracket^{(2)}$, with a permutation applied to one of the inputs, and so on.

Figure 10 illustrates the procedures for a fan-out 3.

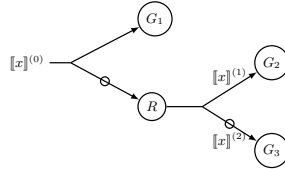


Fig. 10: Illustration of the duplication of a sharing $\llbracket x \rrbracket^{(0)}$ across three gadgets G_1 , G_2 , and G_3 . The black circles on the edges indicate the permutations applied to the shares.

6.2 Composition Strategy for the Compiled Circuits

We now describe the composition strategy to evaluate the random probing security of compiled circuits. Ideally, this would involve computing the cardinal RPC envelope of the entire compiled circuit. However, this approach is often impractical, particularly in the case of circuits with numerous input and output sharings, each composed of a substantial number of shares. The complexity of such an analysis grows at least as $(n+1)^{\ell+m}$, where ℓ and m denote the number of n -shared inputs and outputs, respectively.

This is why we adopt a *block-wise approach* for computing the random probing security of larger circuits. We start by partitioning our larger circuit into blocks of gadgets for which the global cardinal RPC envelopes can be computed within a reasonable amount of time. Subsequently, we use [8, Lemma 1] to infer the threshold RPC security of each block from its cardinal RPC security, which in turn enables us to upper bound the overall random probing security of the compiled circuit.

To ensure efficient computation of the block-level cardinal RPC security, we impose constraints on the size of each block. Specifically, given ℓ input sharings and m output sharings with n shares, we require that $n^{\ell+m} < 2^\delta$, where δ is a chosen threshold value (we use $\delta = 30$ in the application to AES presented below). In this paper, we do not focus on optimizing the block partitioning strategy to maximize the global random probing security, this is left as a direction for future work. Instead, we rely on manual clustering of the compiled circuits, as exemplified hereafter in the case of AES.

6.3 Application to AES

AES is a block cipher, which operates on blocks of 16 bytes. Each byte $b \in \mathbb{F}_{256}$ is a sensible variable of the computation which need to be masked and processed as a sharing $\llbracket b \rrbracket$. We describe hereafter a masked AES

implementation using our compiler. We start by describing the circuits of the four building blocks of an AES implementation. Namely, we describe the circuits of **AddRoundKey**, **SubBytes**, **MixColumns** and **ShiftRows**.

AddRoundKey. This operation consists of adding a 16-byte round key block to the current 16-byte AES state. The masked circuit is constructed by applying 16 addition gadgets, each operating on a shared byte of the state and the corresponding shared byte from the round key. As a result, we obtain 16 envelopes for the **AddRoundKey** step, each one associated to one shared byte of the state and one shared byte of the round key.

Subbytes. This operation is the only non-linear part of the AES. It consists in applying an s-box to each byte of the state. This s-box has a circuit representation which is composed of two parts. The *exponentiation part* maps the input byte $b \in \mathbb{F}_{2^8}$ to b^{254} (which is the multiplicative inverse for every $n \neq 0$ and 0 for $b = 0$). This step can be realized through an addition chain to obtain 254, for instance $(1 \rightarrow 2 \rightarrow 3 \rightarrow 12 \rightarrow 15 \rightarrow 240 \rightarrow 252 \rightarrow 254)$. Figure 11 describes the exponentiation circuit used in our compiler, where the small empty gate are the permutations.

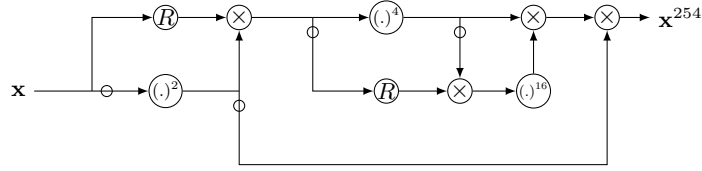


Fig. 11: Exponentiation circuit.

Remark 3. Compared to the rules enforced by the compiler, we manually insert two additional refresh gadgets on the first input variable of the first two multiplication gadgets. This choice was motivated by the observation that refreshing this specific input significantly improves the security level of the block and, consequently, provides a better security–complexity trade-off.

The *affine part* of the s-box consists in the application of the function $\text{Affine} : \mathbb{F}_{2^8} \rightarrow \mathbb{F}_{2^8}$ defined as:

$$\text{Affine}(b) = (((((((207b)^2 + 22b)^2 + b)^2 + 73b)^2 + 204b)^2 + 168b)^2 + 238b)^2 + 5b + 99 .$$

While the above function is used for AES encryption, the inverse function involved for an AES decryption satisfies:

$$\text{Affine}^{-1}(b) = (((((((147b)^2 + 146b)^2 + 190b)^2 + 41b)^2 + 73b)^2 + 139b)^2 + 79b)^2 + 5b + 5 .$$

The circuit for the Affine and Affine^{-1} functions is depicted in Figure 12.

For AES encryption, the **Subbytes** operation applies the exponentiation part followed by the Affine to each shared byte composing the state. For AES decryption, the inverse **Subbytes** operation applies the Affine^{-1} function followed by the exponentiation part to each shared byte composing the state. In both cases, we end up with 16 envelopes for the (inverse) **SubBytes** operation, each one associated to one shared byte of the state.

Remark 4. Since the **AddRoundKey** and **SubBytes** follow one another, we can compose their cardinal RPC envelopes to enhance the threshold RPC security results by building bigger blocks. We will use this optimisation to end up with 16 envelopes for the 16 computations: **AddRoundKey** $\rightarrow$ **SubBytes**.

ShiftRows. This operation performs a permutation of the row of the state. It affects only the wiring of the circuit but do not involve any gadgets.

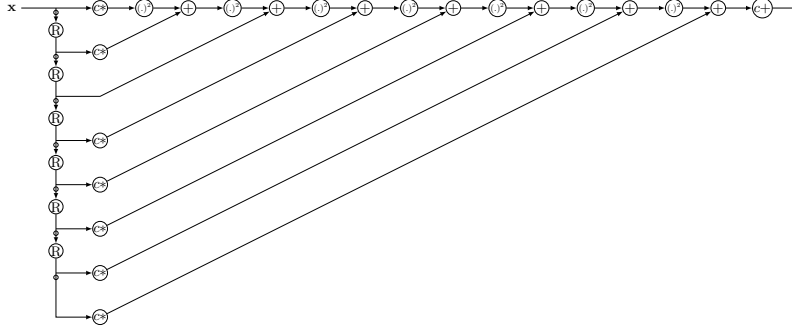


Fig. 12: Circuit for Affine and Affine⁻¹.

MixColumns. This linear transformation takes in input a state $B = (b_{i,j})_{0 \leq i,j \leq 3}$ and computes an output state $B' = (b'_{i,j})_{0 \leq i,j \leq 3}$ where, for all $j \in [0, 3]$:

$$\begin{aligned} b'_{0,j} &= 02 \cdot b_{0,j} \oplus 03 \cdot b_{1,j} \oplus b_{2,j} \oplus b_{3,j} \\ b'_{1,j} &= b_{0,j} \oplus 02 \cdot b_{1,j} \oplus 03 \cdot b_{2,j} \oplus b_{3,j} \\ b'_{2,j} &= b_{0,j} \oplus b_{1,j} \oplus 02 \cdot b_{2,j} \oplus 03 \cdot b_{3,j} \\ b'_{3,j} &= 03 \cdot b_{0,j} \oplus b_{1,j} \oplus b_{2,j} \oplus 02 \cdot b_{3,j} \end{aligned}$$

Figure 13 gives the circuit for the MixColumns operation applied to one column of the state (out of 4). We were not able to compute the overall envelopes of the MixColumns block as its complexity is too big. However, we can split the MixColumns circuit into three sub-circuits: (i & ii) The blue-framed circuits perform all the operations of MixColumns, except for the final addition that produces the four n -sharings. This final step is handled separately in two distinct blocks, which simplifies the computation of this part of the process and (iii) the red-framed circuit corresponds only to 4 additions gadgets in parallel, that we can merge to the next block (namely an AddRoundKey $\rightarrow$ SubBytes block). This way, we can add this part to a bigger block and avoid going through the threshold RPC security.

RPC security of the overall AES. We end up with

- 16 blocks of AddRoundKey $\rightarrow$ SubBytes with known cardinal RPC security envelopes $\mathcal{E}^1$.
- $8 \times 16 = 128$ blocks of MixColumns $\rightarrow$ AddRoundKey $\rightarrow$ SubBytes with known cardinal RPC security envelopes $\mathcal{E}^2$.
- 16 blocks of MixColumns $\rightarrow$ AddRoundKey $\rightarrow$ SubBytes $\rightarrow$ AddRoundKey with known cardinal RPC security envelopes $\mathcal{E}^3$.
- $9 \times 8 = 72$ blocks for the first part of MixColumns with known cardinal RPC security envelopes $\mathcal{E}^4$.

From these cardinal RPC envelopes, we derive the threshold RPC security for each block. We note ε_t^i the t -threshold RPC security derived from the envelopes $\mathcal{E}^i$. Finally, we take $\varepsilon = \min_t \max(\varepsilon_t^1, \varepsilon_t^2, \varepsilon_t^3, \varepsilon_t^4)$. The AES implementation is then $(p, 232 \cdot \varepsilon)$ random probing secure.

6.4 Results and Comparison

Figure 14 illustrates the security-complexity trade-offs of the resulting masked AES in the cardinal RPC framework and compares them with prior random probing secure AES compilers. Specifically, for leakage rates $p \in \{2^{-16}, 2^{-20}\}$, our construction achieves random probing security levels $\varepsilon \in \{2^{-64}, 2^{-80}, 2^{-128}\}$. We

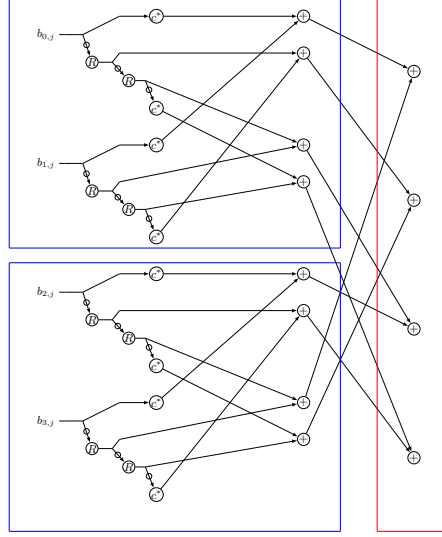


Fig. 13: Circuit for the computation of one column of MixColumns

also report the randomness usage (with light blue for the random bits dedicated to the permutations, that can be revealed) and the counts of field additions and multiplications required in each of these settings, and compare with the following compilers that provide concrete random probing security bounds:

- **Expansion Strategy** [6,9,10]: This approach introduces a compiler equipped with addition, copy, re-fresh, and multiplication gadgets. Its application to AES has been studied in [6,10,5], resulting in complexities several orders of magnitude higher than those obtained by our method. For example, [5, Table 4] reports that achieving 128-bit security at a leakage rate of 2^{-30} requires approximately $4.33 \cdot 10^9$ gates. Due to this significant gap, Figure 14 does not include results for the expansion strategy.
- **BFO23** [11]: This compiler also uses addition, copy, refresh, and multiplication gadgets. Notably, it inserts a refresh gadget between each operation to ensure composability. The authors also provide an analytical formula to estimate the random probing security of their circuits. We applied this compiler to our masked AES implementation for comparison. For $p = 2^{-16}$, the achieved security level never exceeds 44 bits (i.e. the failure probability remains above 2^{-44}) for any number of shares $n \in \mathbb{N}^*$, making meaningful comparisons with our results infeasible. For $p = 2^{-20}$, our compiler achieves lower complexities (or similar for random complexities) for all three target security levels.
- **JMB24** [19]: This work presents a masked AES implementation with random probing security levels expressed as functions of n and p . As shown in Figure 14, our compiler consistently outperforms JMB24 in both complexity and randomness usage for each leakage probability and target security level. It also offers a strictly stronger guarantee: our construction ensures a security level ε in the standard simulation-based random probing security model, whereas JMB24 only upper bounds the mutual information between the leakage and a single byte of the secret key. Such a mutual information bound only limits average leakage and does not yield simulation-based security, whereas simulation-based security (when the distance between the simulated leakage and the actual leakage is statistical) both implies the corresponding mutual information bound and composes well within larger protocols while preserving their security.

Beyond these results, we also compare the combination of our gadgets under the threshold RPC framework with their combination under the block-level cardinal RPC setting. At $p = 2^{-20}$, Figure 14(b) shows our block-level cardinal RPC compiler already exceeds the 64-bit target (with exactly $\varepsilon = 2^{-67}$) with $n = 8$ shares. In the same setting (p, n) , combining directly the threshold RPC advantages of our gadgets (copies included) would only yield $\varepsilon = 2^{-61}$. This gap highlights the benefit of (block-level) cardinal RPC.

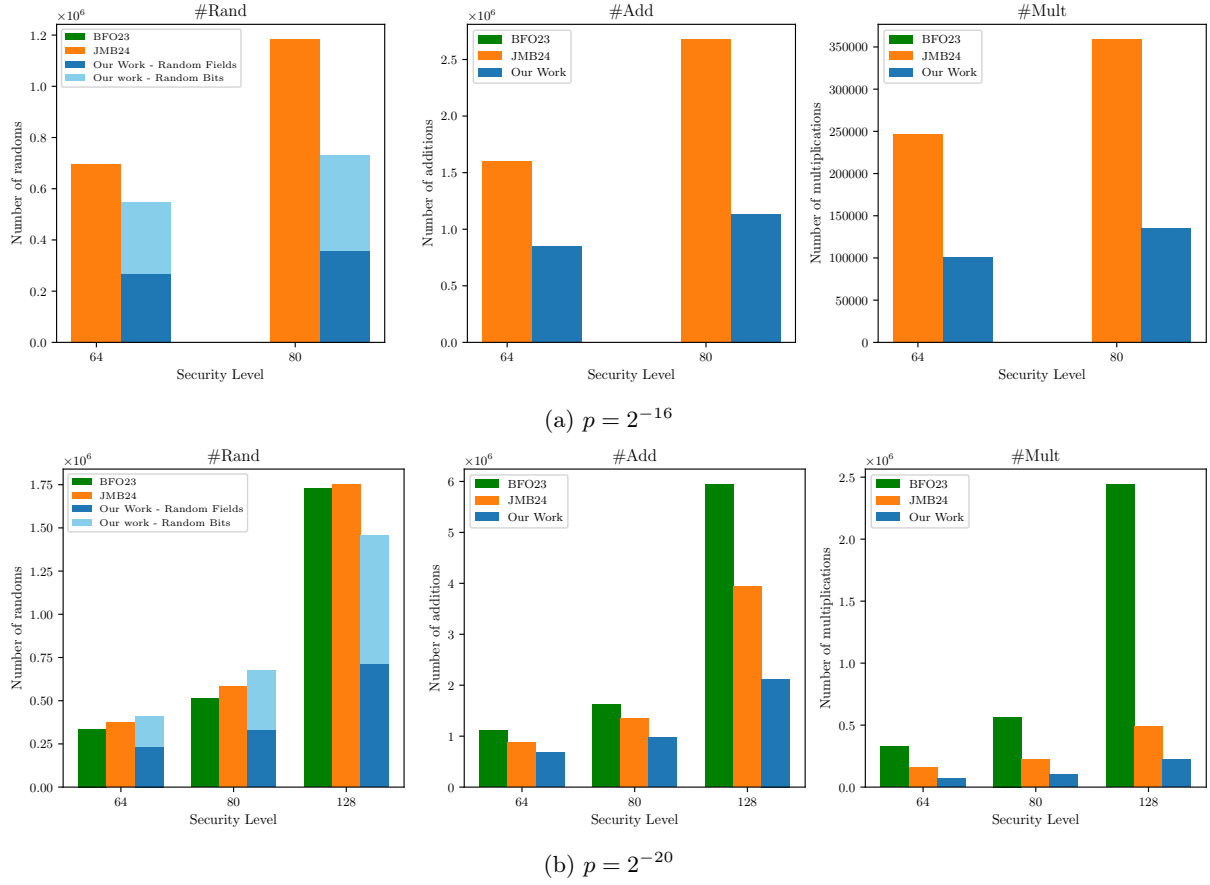


Fig. 14: Number of randoms, additions and multiplications for three security levels.

Acknowledgments. This work was funded by the European Union (ERC, AMaskZONE, 101077506) and Bpifrance (France national quantum strategy, RESQUE project). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the granting authority can be held responsible for them.

References

1. Miklós Ajtai. Secure computation with information leaking to an adversary. In Lance Fortnow and Salil P. Vadhan, editors, *43rd Annual ACM Symposium on Theory of Computing*, pages 715–724, San Jose, CA, USA, June 6–8, 2011. ACM Press.
2. Prabhanjan Ananth, Yuval Ishai, and Amit Sahai. Private circuits: A modular approach. In Hovav Shacham and Alexandra Boldyreva, editors, *Advances in Cryptology – CRYPTO 2018, Part III*, volume 10993 of *Lecture Notes in Computer Science*, pages 427–455, Santa Barbara, CA, USA, August 19–23, 2018. Springer, Cham, Switzerland.
3. Marcin Andrychowicz, Stefan Dziembowski, and Sebastian Faust. Circuit compilers with $O(1/\log(n))$ leakage rate. In Marc Fischlin and Jean-Sébastien Coron, editors, *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 586–615, Vienna, Austria, May 8–12, 2016. Springer Berlin Heidelberg, Germany.
4. Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In Benedikt Gierlichs and Axel Y. Poschmann, editors, *Cryptographic Hardware and Embedded Systems – CHES 2016*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39, Santa Barbara, CA, USA, August 17–19, 2016. Springer Berlin Heidelberg, Germany.
5. Sonia Belaïd, Gaëtan Cassiers, Camille Mutschler, Matthieu Rivain, Thomas Roche, François-Xavier Standaert, and Abdul Rahman Taleb. Sok: A methodology to achieve provable side-channel security in real-world implementations. *IACR Commun. Cryptol.*, 2(1):4, 2025.
6. Sonia Belaïd, Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Abdul Rahman Taleb. Random probing security: Verification, composition, expansion and new constructions. In Daniele Micciancio and Thomas Ristenpart, editors, *Advances in Cryptology – CRYPTO 2020, Part I*, volume 12170 of *Lecture Notes in Computer Science*, pages 339–368, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.
7. Sonia Belaïd, Darius Mercadier, Matthieu Rivain, and Abdul Rahman Taleb. IronMask: Versatile verification of masking security. In *2022 IEEE Symposium on Security and Privacy*, pages 142–160, San Francisco, CA, USA, May 22–26, 2022. IEEE Computer Society Press.
8. Sonia Belaïd, Matthieu Rivain, and Mélissa Rossi. New techniques for random probing security and application to raccoon signature scheme. In Serge Fehr and Pierre-Alain Fouque, editors, *Advances in Cryptology – EUROCRYPT 2025*, pages 94–123, Cham, 2025. Springer Nature Switzerland.
9. Sonia Belaïd, Matthieu Rivain, and Abdul Rahman Taleb. On the power of expansion: More efficient constructions in the random probing model. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021, Part II*, volume 12697 of *Lecture Notes in Computer Science*, pages 313–343, Zagreb, Croatia, October 17–21, 2021. Springer, Cham, Switzerland.
10. Sonia Belaïd, Matthieu Rivain, Abdul Rahman Taleb, and Damien Vergnaud. Dynamic random probing expansion with quasi linear asymptotic complexity. In Mehdi Tibouchi and Huaxiong Wang, editors, *Advances in Cryptology – ASIACRYPT 2021, Part II*, volume 13091 of *Lecture Notes in Computer Science*, pages 157–188, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.
11. Francesco Berti, Sebastian Faust, and Maximilian Ortl. Provable secure parallel gadgets. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(4):420–459, 2023.
12. Gaëtan Cassiers, Sebastian Faust, Maximilian Ortl, and François-Xavier Standaert. Towards tight random probing security. In Tal Malkin and Chris Peikert, editors, *Advances in Cryptology – CRYPTO 2021, Part III*, volume 12827 of *Lecture Notes in Computer Science*, pages 185–214, Virtual Event, August 16–20, 2021. Springer, Cham, Switzerland.
13. Gaëtan Cassiers and François-Xavier Standaert. Towards globally optimized masking: From low randomness to low noise rate. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(2):162–198, 2019.
14. Rafaël del Pino, Shuichi Katsumata, Thomas Prest, and Mélissa Rossi. Raccoon: A masking-friendly signature proven in the probing model. In Leonid Reyzin and Douglas Stebila, editors, *Advances in Cryptology – CRYPTO 2024, Part I*, volume 14920 of *Lecture Notes in Computer Science*, pages 409–444, Santa Barbara, CA, USA, August 18–22, 2024. Springer, Cham, Switzerland.
15. Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In Phong Q. Nguyen and Elisabeth Oswald, editors, *Advances in Cryptology – EUROCRYPT 2014*, volume 8441 of *Lecture Notes in Computer Science*, pages 423–440, Copenhagen, Denmark, May 11–15, 2014. Springer Berlin Heidelberg, Germany.
16. Stefan Dziembowski, Sebastian Faust, and Karol Zebrowski. Simple refreshing in the noisy leakage model. In Steven D. Galbraith and Shiho Moriai, editors, *Advances in Cryptology – ASIACRYPT 2019, Part III*, volume

- 11923 of *Lecture Notes in Computer Science*, pages 315–344, Kobe, Japan, December 8–12, 2019. Springer, Cham, Switzerland.
17. Dahmun Goudarzi, Antoine Joux, and Matthieu Rivain. How to securely compute with noisy leakage in quasilinear complexity. In Thomas Peyrin and Steven Galbraith, editors, *Advances in Cryptology – ASIACRYPT 2018, Part II*, volume 11273 of *Lecture Notes in Computer Science*, pages 547–574, Brisbane, Queensland, Australia, December 2–6, 2018. Springer, Cham, Switzerland.
 18. Yuval Ishai, Amit Sahai, and David Wagner. Private circuits: Securing hardware against probing attacks. In Dan Boneh, editor, *Advances in Cryptology – CRYPTO 2003*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481, Santa Barbara, CA, USA, August 17–21, 2003. Springer Berlin Heidelberg, Germany.
 19. Vahid Jahandideh, Bart Mennink, and Lejla Batina. An algebraic approach for evaluating random probing security with application to AES. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024(4):657–689, 2024.
 20. Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In Thomas Johansson and Phong Q. Nguyen, editors, *Advances in Cryptology – EUROCRYPT 2013*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159, Athens, Greece, May 26–30, 2013. Springer Berlin Heidelberg, Germany.

A Recall of Random Probing Partitioning Definition

Definition 10 (Partition ZeroEncoding from [8]). For $P = (n_1, \dots, n_\ell) \in \mathcal{P}(n)$ an integer partition, we define $\text{PartitionZeroEnc}(P)$ as a particular type of distribution producing a sharing $\llbracket z \rrbracket = \llbracket z_1 \rrbracket \parallel \dots \parallel \llbracket z_\ell \rrbracket$ where $\llbracket z_i \rrbracket$ is a fresh n_i -sharing of 0 for every $i \in [1, \ell]$. In other words,

$$\text{PartitionZeroEnc}(n_1, \dots, n_\ell) = \underbrace{z_1, \dots, z_{n_1}}_{\sum z_i = 0}, \underbrace{z_{n_1+1}, \dots, z_{n_1+n_2}}_{\sum z_i = 0}, \dots, \underbrace{z_{n-n_\ell+1}, \dots, z_n}_{\sum z_i = 0}.$$

Definition 11 (Random Probing Partitioning from [8]). Let $n \in \mathbb{N}$ and $\mathcal{P}(n)$ be the set of partitions of n . Let $\mathcal{E}$ be a distribution envelope over $\mathcal{P}(n)$. An n -share zero-encoding gadget $\text{ZeroEnc} : \emptyset \rightarrow \mathbb{K}^n$ is $(p, \mathcal{E})$ -random-probing-partitioned (RPP) for some $p \in [0, 1]$ if there exists a deterministic algorithm ϕ and a PPT simulator Sim such that the outputs of the random experiment 2 satisfy:

1. $P \lesssim \mathcal{E}$, and
2. $\llbracket z \rrbracket$ and $\llbracket w \rrbracket + \llbracket u \rrbracket$ are identically distributed.

Experiment 2: Random Probing Partition

$\mathcal{W} \leftarrow \text{LeakingWires}(\text{ZeroEnc}, p)$ $\rho \leftarrow \$$ $\llbracket z \rrbracket \leftarrow \text{ZeroEnc}(\rho)$ $\text{leak} \leftarrow \text{AssignWires}(\text{ZeroEnc}, \rho, \mathcal{W}, \emptyset)$ $\llbracket w \rrbracket \leftarrow \text{Sim}(\mathcal{W}, \text{leak})$ $(P, \sigma) := \phi(\mathcal{W})$ $\llbracket u \rrbracket \leftarrow \sigma(\text{PartitionZeroEnc}(P))$ $\text{return } (P, \llbracket z \rrbracket, \llbracket w \rrbracket, \llbracket u \rrbracket)$	Drawing the leaking wires. Drawing a random tape. Drawing $\llbracket z \rrbracket$ an encoding of 0. Computing the leaked values. The leakages are given to the simulator. It outputs an encoding of 0 whose randomness is "weakened" due to the leakage. Computing a partition and a permutation corresponding to the leaking wires. Drawing an encoding of 0 corresponding to the "remaining" randomness to compensate the randomness that was given in the leakage.

B Proofs of Section 4

Proof (Proof of Lemma 2). Let $n, \ell, m \in \mathbb{N}$ and let $G : (\mathbb{K}^n)^\ell \rightarrow (\mathbb{K}^n)^m$ be an n -share gadget. Let $p \in [0, 1]$.

On the one hand, we suppose that G is *uniformly*-($p, \mathcal{E}^u$)-cardinal random probing composable. We show that G is $(p, \mathcal{E}^{ui})$ -UI-cardinal random probing composable and $(p, \mathcal{E}^{uo})$ -UO-cardinal random probing composable and that

$$\mathcal{E}^u = \mathcal{E}^{ui} = \mathcal{E}^{uo}$$

Let $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and $(J'_1, \dots, J'_m) \in \mathcal{P}([n])^m$ such that

$$(|J_1|, \dots, |J_m|) = (|J'_1|, \dots, |J'_m|)$$

and let $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(I'_1, \dots, I'_\ell) \in \mathcal{P}([n])^\ell$ such that

$$(|I_1|, \dots, |I_\ell|) = (|I'_1|, \dots, |I'_\ell|)$$

By the uniformly cardinal RPC property of G , we know that G is $(p, \mathcal{E}^g)$ -general RPC with

$$\mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \mathcal{E}_{J'_1, \dots, J'_m}^g(I'_1, \dots, I'_\ell) \quad (4)$$

and with

$$\mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = \mathcal{E}_{J'_1, \dots, J'_m}^g(I_1, \dots, I_\ell) \quad (5)$$

Therefore, by Equation 4, G is $(p, \mathcal{E}^{ui})$ -UI-cardinal random probing composable such that for all $(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \in [0, n]^\ell$ and for all $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$ we have

$$\begin{aligned} \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^{ui}(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) &= \max_{\substack{J'_1, \dots, J'_m \\ |J'_1| = t_1^{\text{out}}, \dots, |J'_m| = t_m^{\text{out}}}} \mathcal{E}_{J'_1, \dots, J'_m}^g(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{\text{in}}} \\ &\stackrel{5}{=} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{\text{in}}} \\ &= \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^u(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \end{aligned}$$

where $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ with $|J_1| = t_1^{\text{out}}, \dots, |J_m| = t_m^{\text{out}}$ and $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ with $|I_1| = t_1^{\text{in}}, \dots, |I_\ell| = t_\ell^{\text{in}}$. Furthermore, by Equation 5, G is $(p, \mathcal{E}^{uo})$ -UO-cardinal random probing composable such that

$$\begin{aligned} \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^{uo}(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) &= \sum_{\substack{I'_1, \dots, I'_\ell \\ |I'_1| = t_1^{\text{in}}, \dots, |I'_\ell| = t_\ell^{\text{in}}}} \mathcal{E}_{J_1, \dots, J_m}^g(I'_1, \dots, I'_\ell) \\ &\stackrel{4}{=} \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{\text{in}}} \\ &= \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^u(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \end{aligned}$$

where $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ with $|J_1| = t_1^{\text{out}}, \dots, |J_m| = t_m^{\text{out}}$ and $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ with $|I_1| = t_1^{\text{in}}, \dots, |I_\ell| = t_\ell^{\text{in}}$. Finally we have

$$\mathcal{E}^u = \mathcal{E}^{ui} = \mathcal{E}^{uo}.$$

On the other hand, we suppose that G is $(p, \mathcal{E}^{ui})$ -UI-cardinal random probing composable and $(p, \mathcal{E}^{uo})$ -UO-cardinal random probing composable with $\mathcal{E}^u := \mathcal{E}^{ui} = \mathcal{E}^{uo}$. Then we show that G is uniformly- $(p, \mathcal{E}^u)$ -cardinal random probing composable. Let $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$ and $(J'_1, \dots, J'_m) \in \mathcal{P}([n])^m$ such that

$$(|J_1|, \dots, |J_m|) = (|J'_1|, \dots, |J'_m|)$$

and let $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(I'_1, \dots, I'_\ell) \in \mathcal{P}([n])^\ell$ such that

$$(|I_1|, \dots, |I_\ell|) = (|I'_1|, \dots, |I'_\ell|)$$

By the UI-cardinal random probing composability we know that G is $(p, \mathcal{E}^g)$ -general RPC with

$$\mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) \stackrel{UI}{=} \mathcal{E}_{J'_1, \dots, J'_m}^g(I'_1, \dots, I'_\ell) \quad (6)$$

Moreover, by the UO-cardinal random probing composability, we know that

$$\sum_{\substack{I_1^* \subseteq [n], \dots, I_\ell^* \subseteq [n] \\ |I_1^*| = |I_1|, \dots, |I_\ell^*| = |I_\ell|}} \mathcal{E}_{J_1, \dots, J_m}^g(I_1^*, \dots, I_\ell^*) = \sum_{\substack{I_1^* \subseteq [n], \dots, I_\ell^* \subseteq [n] \\ |I_1^*| = |I_1|, \dots, |I_\ell^*| = |I_\ell|}} \mathcal{E}_{J'_1, \dots, J'_m}^g(I_1^*, \dots, I_\ell^*)$$

In particular, by Equation 6, we obtain

$$c.\mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) = c.\mathcal{E}_{J'_1, \dots, J'_m}^g(I'_1, \dots, I'_\ell)$$

where $c := |\{I_1^* \subseteq [n], \dots, I_\ell^* \subseteq [n] \text{ such that } |I_1^*| = |I_1|, \dots, |I_\ell^*| = |I_\ell|\}|$. Therefore, dividing by c , we show that G is *uniformly-($p, \mathcal{E}$)-cardinal random probing composable* such that for all $(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \in [0, n]^\ell$ and $(t_1^{\text{out}}, \dots, t_m^{\text{out}}) \in [0, n]^m$ we have

$$\begin{aligned} \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) &= \mathcal{E}_{J_1, \dots, J_m}^g(I_1, \dots, I_\ell) \prod_{\alpha=1}^{\ell} \binom{n}{t_\alpha^{\text{in}}} \\ &\stackrel{UI}{=} \sum_{\substack{I'_1, \dots, I'_\ell \\ |I'_1| = t_1^{\text{in}}, \dots, |I'_\ell| = t_\ell^{\text{in}}}} \mathcal{E}_{J_1, \dots, J_m}^g(I'_1, \dots, I'_\ell) \\ &= \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^{ui}(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \\ &= \mathcal{E}_{t_1^{\text{out}}, \dots, t_m^{\text{out}}}^u(t_1^{\text{in}}, \dots, t_\ell^{\text{in}}) \end{aligned}$$

Finally, G is *uniformly-($p, \mathcal{E}^u$)-cardinal random probing composable*. This concludes the proof. $\square$

Proof (Proof of Lemma 3). Let $\llbracket \mathbf{x} \rrbracket \in \mathbb{K}^n$, $J_1 \subseteq [n]$, $J_2 \subseteq [n]$, and $\mathcal{W} = \text{LeakingWires}(G, p)$. We split $\mathcal{W}$ into two parts: the set $\mathcal{W}_1$ containing the wires of G_1 and the set $\mathcal{W}_2$ containing the wires in G_2 . We then build two simulators $(\text{Sim}_1, \text{Sim}_2)$ from the simulators $(\text{Sim}_1^{G_1}, \text{Sim}_2^{G_1})$ and $(\text{Sim}_1^{G_2}, \text{Sim}_2^{G_2})$ of the cardinal RPC property of G_1 and G_2 (see Definition 7).

$$\text{Sim}_1(G, \mathcal{W}, J_1, J_2) := \text{Sim}_1^{G_1}(G_1, \mathcal{W}_1, J_1) \cup \text{Sim}_1^{G_2}(G_2, \mathcal{W}_2, J_2).$$

Let $I_1 \leftarrow \text{Sim}_1^{G_1}(G_1, \mathcal{W}_1, J_1)$ and $I_2 \leftarrow \text{Sim}_1^{G_2}(G_2, \mathcal{W}_2, J_2)$ denote the resulting sets. Therefore, the resulting input set for $\text{Sim}_1(G, \mathcal{W}, J_1, J_2)$ is $I = I_1 \cup I_2$. We then define the second simulator as follows:

$$\text{Sim}_2(\llbracket \mathbf{x} \rrbracket) := (out_1^1 || out_2^1, out_2^2 || out_2^2)$$

where $(out_1^1, out_2^1) \leftarrow \text{Sim}_2^{G_1}(\llbracket \mathbf{x} \rrbracket|_I)$ and $(out_2^2, out_2^2) \leftarrow \text{Sim}_2^{G_2}(\llbracket \mathbf{x} \rrbracket|_I)$.

Since out_1^1 is identically distributed to the internal leakage of $\mathcal{W}_1$ in G_1 (because $I_1 \subseteq I$) and out_2^2 is identically distributed to the internal leakage of $\mathcal{W}_2$ in G_2 (because $I_2 \subseteq I$), it follows that $out_1 := (out_1^1 || out_2^1)$ is identically distributed to the internal leakage $\mathcal{W}$ in G . Similarly, $out_2 := (out_2^1 || out_2^2)$ is identically distributed to the output leakage of G . This ensures that the second condition of the cardinal RPC property is satisfied.

Regarding the first condition, we have for all $t^{\text{in}} \in [0, n]$ and for all $(t_1^{\text{out}}, t_2^{\text{out}}) \in [0, n]^2$:

$$\begin{aligned} &\mathbb{P}(|I| = t^{\text{in}} \mid |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}) \\ &= \mathbb{P}(|I_1 \cup I_2| = t^{\text{in}} \mid |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}) \\ &= \sum_{i_1=0}^{t^{\text{in}}} \sum_{i_2=t^{\text{in}}-i_1}^{t^{\text{in}}} \mathbb{P}(|I_1 \cup I_2| = t^{\text{in}} \mid |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}, |I_1| = i_1, |I_2| = i_2) \\ &\quad \cdot \mathbb{P}(|I_1| = i_1, |I_2| = i_2 \mid |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}) \end{aligned}$$

The two events $|I_1| = i_1$ and $|I_2| = i_2$ are independent with their respective conditions on the output cardinals, therefore we have:

$$\begin{aligned} &\mathbb{P}(|I| = t^{\text{in}} \mid |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}) \\ &\leq \sum_{i_1=0}^{t^{\text{in}}} \sum_{i_2=t^{\text{in}}-i_1}^{t^{\text{in}}} \mathbb{P}(|I_1 \cup I_2| = t^{\text{in}} \mid |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}, |I_1| = i_1, |I_2| = i_2) \\ &\quad \cdot \mathcal{E}_{t_1^{\text{out}}}^1(i_1) \cdot \mathcal{E}_{t_2^{\text{out}}}^{u2}(i_2) \end{aligned}$$

Since G_2 is UI cardinal RPC, each set of input shares required for the simulation of G_2 is equiprobable for equal cardinals. Therefore, we can compute the probability of the cardinality of $|I_1 \cup I_2|$ only from the cardinalities $|I_1|$ and $|I_2|$. Specifically $|I_1 \cap I_2| = |I_1| + |I_2| - |I_1 \cup I_2|$ follows a hypergeometric law. We thus have

$$\mathbb{P}(|I_1 \cap I_2| = i_1 + i_2 - t^{\text{in}} \mid |I_1| = i_1, |I_2| = i_2, |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}) = \frac{\binom{i_1}{i_1+i_2-t^{\text{in}}} \cdot \binom{n-i_1}{t^{\text{in}}-i_1}}{\binom{n}{i_2}}$$

and we obtain

$$\begin{aligned} & \mathbb{P}(|I| = t^{\text{in}} \mid |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}) \\ & \leq \sum_{i_1=0}^{t^{\text{in}}} \sum_{i_2=t^{\text{in}}-i_1}^{t^{\text{in}}} \mathbb{P}(|I_1 \cap I_2| = i_1 + i_2 - t^{\text{in}} \mid |J_1| = t_1^{\text{out}}, |J_2| = t_2^{\text{out}}, |I_1| = i_1, |I_2| = i_2) \\ & \quad \cdot \mathcal{E}_{t_1^{\text{out}}}^1(i_1) \cdot \mathcal{E}_{t_2^{\text{out}}}^{ui2}(i_2) \\ & \leq \sum_{i_1=0}^{t^{\text{in}}} \sum_{i_2=t^{\text{in}}-i_1}^{t^{\text{in}}} \frac{\binom{i_1}{i_1+i_2-t^{\text{in}}} \cdot \binom{n-i_1}{t^{\text{in}}-i_1}}{\binom{n}{i_2}} \cdot \mathcal{E}_{t_1^{\text{out}}}^1(i_1) \cdot \mathcal{E}_{t_2^{\text{out}}}^{ui2}(i_2) \end{aligned}$$

which concludes the proof. $\square$

Proof (Proof of Lemma 5). We apply [12, Theorem 1] to give us the general RPC envelopes of the parallel composition of both refresh gadget RPRefresh , we note this parallel composition as gadget G_1 . We note G_2 the addition gadget which consists simply in adding the shares of each secret together (i.e. the last loop **for** in Algorithm 6). We compute the general RPC envelopes of G_2 . Let $(I'_1, I'_2) \subseteq [n]^2$ and $J \subseteq [n]$. Since the outputs shares are not masked by randoms in G_2 , we need at least the $|J|$ input shares of each secrets leaked by J . Then we can complete with the leakage on the input shares. This gives us the general RPC envelopes of G_2

$$\mathcal{E}_J(I'_1, I'_2) = \begin{cases} 0 & \text{if } J \not\subseteq I'_1 \text{ or } J \not\subseteq I'_2 \\ p^{|I'_1|-|J|} \cdot (1-p)^{n-|I'_1|} \cdot p^{|I'_2|-|J|} \cdot (1-p)^{n-|I'_2|} & \text{otherwise} \end{cases}$$

Then we apply [12, Theorem 2] to give us the general RPC envelopes of the sequential composition of G_1 and G_2 , which gives us the general RPC envelopes of Add^γ . $\square$

Proof (Proof of Lemma 6). We note G_2 the addition gadget which consist simply to multiply the shares by the constant c (i.e. the last loop **for** in Algorithm 7). We compute the general RPC envelopes of G_2 . Let $I_1 \subseteq [n]$ and $J \subseteq [n]$. Since the output shares are not masked by randoms in G_2 , we need at least the $|J|$ input shares leaked by J . Then we can complete with the leakage on the input shares. This gives us the general RPC envelopes of G_2

$$\mathcal{E}_J(I_1) = \begin{cases} 0 & \text{if } J \not\subseteq I_1 \\ p^{|I_1|-|J|} \cdot (1-p)^{n-|I_1|} & \text{otherwise} \end{cases}$$

Then the overall gadget is a sequential composition of the RPRefresh gadget and the gadget G_2 . By [12, Theorem 2], we obtain the general RPC envelopes of cMult^γ . $\square$

Proof (Proof of Lemma 7). First, we establish the general RPC envelopes of the concatenation of permutations σ . Specifically, by the definition of a uniform random permutation, for all $(I_1, \dots, I_\ell) \subseteq [n]^\ell$ and $(J_1, \dots, J_\ell) \subseteq [n]^\ell$, we have

$$\mathcal{E}_{J_1, \dots, J_\ell}^\sigma(I_1, \dots, I_\ell) = \frac{1}{\prod_{i=1}^\ell \binom{n}{|I_i|}} \cdot \mathbf{1}_{(|J_1|=|I_1|) \wedge \dots \wedge (|J_\ell|=|I_\ell|)}.$$

Next, we apply the upper bound for the general RPC envelopes of the sequential composition of two gadgets from [12, Theorem 2]. This gives the following expression for all $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$:

$$\begin{aligned}\mathcal{E}_{J_1, \dots, J_m}^{G \circ \sigma}(I_1, \dots, I_\ell) &= \sum_{I_1^*, \dots, I_\ell^* \subseteq [n]} \mathcal{E}_{I_1^*, \dots, I_\ell^*}^\sigma(I_1, \dots, I_\ell) \cdot \mathcal{E}_{J_1, \dots, J_m}^G(I_1^*, \dots, I_\ell^*) \\ &= \sum_{\substack{I_1^*, \dots, I_\ell^* \subseteq [n] \\ |I_1^*| = |I_1|, \dots, |I_\ell^*| = |I_\ell|}} \frac{1}{\prod_{i=1}^\ell \binom{n}{|I_i|}} \cdot \mathcal{E}_{J_1, \dots, J_m}^G(I_1^*, \dots, I_\ell^*)\end{aligned}$$

which proves Equation 1. Finally, the expression for $\mathcal{E}^{ui}$ is directly derived from Definition 8. $\square$

Proof (Proof of Lemma 8). First, we establish the general RPC envelopes of the concatenation of permutations σ . Specifically, by the definition of a uniform random permutation, for all $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(J_1, \dots, J_\ell) \in \mathcal{P}([n])^\ell$, we have

$$\mathcal{E}_{J_1, \dots, J_\ell}^\sigma(I_1, \dots, I_\ell) = \frac{1}{\prod_{i=1}^\ell \binom{n}{|I_i|}} \cdot \mathbf{1}_{(|J_1| = |I_1|) \wedge \dots \wedge (|J_\ell| = |I_\ell|)}.$$

Next, we apply the upper bound for the general RPC envelopes of the sequential composition of two gadgets from [12, Theorem 2]. This gives the following expression for all $(I_1, \dots, I_\ell) \in \mathcal{P}([n])^\ell$ and $(J_1, \dots, J_m) \in \mathcal{P}([n])^m$:

$$\begin{aligned}\mathcal{E}_{J_1, \dots, J_m}^{\sigma \circ G}(I_1, \dots, I_\ell) &= \sum_{J_1^*, \dots, J_m^* \subseteq [n]} \mathcal{E}_{J_1^*, \dots, J_m^*}^G(I_1, \dots, I_\ell) \cdot \mathcal{E}_{J_1, \dots, J_m}^\sigma(J_1^*, \dots, J_m^*) \\ &= \sum_{\substack{J_1^*, \dots, J_m^* \subseteq [n] \\ |J_1^*| = |J_1|, \dots, |J_m^*| = |J_m|}} \mathcal{E}_{J_1^*, \dots, J_m^*}^G(I_1, \dots, I_\ell) \cdot \frac{1}{\prod_{i=1}^m \binom{n}{|J_i|}}\end{aligned}$$

which proves Equation 2. Finally, the expression for $\mathcal{E}^{uo}$ is directly derived from Definition 9. $\square$

Proof (Proof of Lemma 4). Let $I, J \subseteq [n]$ denote the sets of required input shares and leaked output shares, respectively. We decompose the set of required shares I into two subsets. The first subset, denoted by K , corresponds to the input shares required for the simulation of the output shares J and the leakage of the gadget $\text{RPZeroEnc}_+^{n, \gamma}$. The second subset is $I \setminus K$, representing the shares that leak directly at the input. From the RPP (Random Probing Partition) property of the gadget $\text{RPZeroEnc}_+^{n, \gamma}$, proven in [8], its output shares can be seen, by the random probing attacker, as divided into independent uniformly random 0-sharings. From the uniform properties of $\text{RPZeroEnc}_+^{n, \gamma}$, these 0-sharings are fully determined by a partition $P = (c_1, \dots, c_\ell)$ of n , where each c_i represents the size of a 0-sharing. Given such a partition P , we consider tuples $(k_1, \dots, k_\ell)$ such that $0 \leq k_i \leq c_i$ and:

$$\sum_{i=1}^\ell k_i = |J|.$$

These tuples represent the number of shares in each 0-sharing at the output of $\text{RPZeroEnc}_+^{n, \gamma}$ that matches the leaked output shares in J . Now, in order for some input shares to be required for simulation, they must be associated with fully leaked 0-sharings. That is, for each i such that $k_i = c_i$, all the output shares in that 0-sharing are included in J , and the corresponding input shares must be included in K . To ensure that exactly $|K|$ such input shares are required, we impose the condition:

$$\sum_{i=1}^\ell k_i \cdot \mathbf{1}_{(k_i = c_i)} = |K|.$$

The probability of such a configuration occurring is determined by the envelopes $\mathcal{E}^\gamma$ of $\text{RPZeroEnc}_+^{n,\gamma}$ and the probability of selecting the tuple $(k_1, \dots, k_\ell)$ given P, J , and K . This is given by:

$$\frac{\prod_{i=1}^{\ell} \binom{c_i}{k_i}}{\binom{|J|}{|K|} \cdot \binom{n}{|J|}}$$

Finally, the remaining required shares in $I \setminus K$ must leak independently at the input, each with probability p . The probability that these shares leak, while the input shares not in I must remain un-leaked, is given by:

$$p^{|I|-|K|} \cdot (1-p)^{n-|I|},$$

which concludes the proof. $\square$

C Gadgets

C.1 Linear gadgets from [8] are UO cardinal RPC

Algorithm 4 RPZeroEnc^γ from [8]

Output : $\llbracket z \rrbracket$ such that $\sum_{i=1}^n z_i = 0$
 $\llbracket z \rrbracket = (0, \dots, 0)$
for $i = 1$ **to** γ **do**
 $i_1, i_2 \xleftarrow{\$} [1, n]$
 $r \xleftarrow{\$} \mathbb{K}$
 $z_{i_1} \leftarrow z_{i_1} + r$
 $z_{i_2} \leftarrow z_{i_2} - r$
end for
return $\llbracket z \rrbracket$

Algorithm 5 RPRefresh^γ from [8]

Input : $\llbracket a \rrbracket = (a_1, \dots, a_n) \in \mathbb{K}^n$
Output : $\llbracket s \rrbracket = (s_1, \dots, s_n) \in \mathbb{K}^n$ such that $\sum_{i=1}^n s_i = \sum_{i=1}^n a_i$
 $\llbracket z \rrbracket \leftarrow \text{RPZeroEnc}^\gamma$
for $i = 1$ **to** n **do**
 $s_i \leftarrow a_i + z_i$
end for
return $\llbracket s \rrbracket$

Algorithm 6 Add^γ

Input : $\llbracket a \rrbracket, \llbracket b \rrbracket$ two n -sharing.
Output : $\llbracket c \rrbracket$ a n -sharing of $a + b$.
 $\llbracket a' \rrbracket \leftarrow \text{RPRefresh}^\gamma(\llbracket a \rrbracket)$
 $\llbracket b' \rrbracket \leftarrow \text{RPRefresh}^\gamma(\llbracket b \rrbracket)$
for $i = 1$ **to** n **do**
 $c_i \leftarrow a'_i + b'_i$
end for
return $\llbracket c \rrbracket$

Algorithm 7 cMult^γ

Input : $\llbracket a \rrbracket$ a n -sharing, c a constant.
Output : $\llbracket b \rrbracket$ a n -sharing of $c \cdot a$.
 $\llbracket a' \rrbracket \leftarrow \text{RPRefresh}^\gamma(\llbracket a \rrbracket)$
for $i = 1$ **to** n **do**
 $b_i \leftarrow c \cdot a'_i$
end for
return $\llbracket b \rrbracket$

Proof (Proof of Corollary 2). Let $J, J' \subseteq [n]$ such that $|J| = |J'|$. Let $(t_1^i, t_2^i) \in [0, n]^2$, we have

$$\sum_{\substack{I_1 \subseteq [n] \\ |I_1|=t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2|=t_2^i}} \mathcal{E}_J^g(I_1, I_2) = \sum_{\substack{I_1 \subseteq [n] \\ |I_1|=t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2|=t_2^i}} \sum_{I'_1 \subseteq [n]} \sum_{I'_2 \subseteq [n]} \mathcal{E}_{I'_1}^{g_1}(I_1) \cdot \mathcal{E}_{I'_2}^{g_1}(I_2) \cdot \mathcal{E}_J(I'_1, I'_2)$$

These sums are finite and we can rearrange them

$$\sum_{\substack{I_1 \subseteq [n] \\ |I_1|=t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2|=t_2^i}} \mathcal{E}_J^g(I_1, I_2) = \sum_{\substack{I'_1 \subseteq [n] \\ J \subseteq I'_1}} \sum_{\substack{I'_2 \subseteq [n] \\ J \subseteq I'_2}} \mathcal{E}_J(I'_1, I'_2) \left(\sum_{\substack{I_1 \subseteq [n] \\ |I_1|=t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2|=t_2^i}} \mathcal{E}_{I'_1}^{g_1}(I_1) \cdot \mathcal{E}_{I'_2}^{g_1}(I_2) \right)$$

Now, by the UO cardinal RPC security of RPRefresh^γ , we know that for $(I_1^*, I_2^*) \subseteq [n]^2$ such that $|I_1^*| = |I_1^*|$ and $|I_2^*| = |I_2^*|$, we have

$$\sum_{\substack{I_1 \subseteq [n] \\ |I_1|=t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2|=t_2^i}} \mathcal{E}_{I_1^*}^{g_1}(I_1) \cdot \mathcal{E}_{I_2^*}^{g_1}(I_2) = \sum_{\substack{I_1 \subseteq [n] \\ |I_1|=t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2|=t_2^i}} \mathcal{E}_{I_1^*}^{g_1}(I_1) \cdot \mathcal{E}_{I_2^*}^{g_1}(I_2)$$

Furthermore, if $J' \subseteq I_1^*$ and $J' \subseteq I_2^*$ we have also

$$\mathcal{E}_J(I_1', I_2') = \mathcal{E}_{J'}(I_1^*, I_2^*)$$

Finally, because $|J| = |J'|$, for any cardinality $\ell \in [0, n]$, we have

$$\#\{I' \subseteq [n] \mid |I'| = \ell \text{ and } J \subseteq I'\} = \#\{I^* \subseteq [n] \mid |I^*| = \ell \text{ and } J' \subseteq I^*\}$$

Therefore we have

$$\sum_{\substack{I_1 \subseteq [n] \\ |I_1| = t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2| = t_2^i}} \mathcal{E}_J^g(I_1, I_2) = \sum_{\substack{I_1^* \subseteq [n] \\ J' \subseteq I_1^*}} \sum_{\substack{I_2^* \subseteq [n] \\ J' \subseteq I_2^*}} \mathcal{E}_{J'}(I_1^*, I_2^*) \left(\sum_{\substack{I_1 \subseteq [n] \\ |I_1| = t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2| = t_2^i}} \mathcal{E}_{I_1^*}^{g_1}(I_1) \cdot \mathcal{E}_{I_2^*}^{g_1}(I_2) \right)$$

Eventually, we obtain the results

$$\sum_{\substack{I_1 \subseteq [n] \\ |I_1| = t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2| = t_2^i}} \mathcal{E}_J^g(I_1, I_2) = \sum_{\substack{I_1 \subseteq [n] \\ |I_1| = t_1^i}} \sum_{\substack{I_2 \subseteq [n] \\ |I_2| = t_2^i}} \mathcal{E}_{J'}^g(I_1, I_2)$$

Hence, Add^γ is UO cardinal RPC. This conclude the proof. $\square$

Proof (Proof of Corollary 3). Let $J, J' \subseteq [n]$ such that $|J| = |J'|$. Let $t_1^i \in [0, n]$, we have

$$\sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_J^g(I) = \sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \sum_{I_1 \subseteq [n]} \mathcal{E}_{I_1}^{g_1}(I) \cdot \mathcal{E}_J(I_1)$$

These sums are finite and we can rearrange them

$$\sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_J^g(I) = \sum_{I_1 \subseteq [n]} \mathcal{E}_J(I_1) \cdot \left(\sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_{I_1}^{g_1}(I) \right)$$

Now by the UO cardinal RPC of RPRefresh^γ , we know that for $I_1' \subseteq [n]$ such that $|I_1| = |I_1'|$, we have

$$\sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_{I_1}^{g_1}(I) = \sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_{I_1'}^{g_1}(I)$$

Furthermore, if $J \subseteq I_1$ and $J' \subseteq I_1'$, we have also

$$\mathcal{E}_J(I_1) = \mathcal{E}_{J'}(I_1')$$

Finally, because $|J| = |J'|$, for any cardinality $\ell \in [0, n]$, we have

$$\#\{I_1 \subseteq [n] \mid |I_1| = \ell \text{ and } J \subseteq I_1\} = \#\{I_1' \subseteq [n] \mid |I_1'| = \ell \text{ and } J' \subseteq I_1'\}$$

Therefore we have

$$\sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_J^g(I) = \sum_{I_1' \subseteq [n]} \mathcal{E}_{J'}(I_1') \cdot \left(\sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_{I_1'}^{g_1}(I) \right)$$

Eventually, we obtain the results

$$\sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_J^g(I) = \sum_{\substack{I \subseteq [n] \\ |I| = t_1^i}} \mathcal{E}_{J'}^g(I)$$

Hence, cMult^γ is UO cardinal RPC. This concludes the proof. $\square$

C.2 Constant Addition Gadget

Algorithm 8 cAdd^γ

Input : A n -sharing $\llbracket x \rrbracket = (x_1, \dots, x_n)$, a constant $c \in \mathbb{K}$
Output : A n -sharing $\llbracket z \rrbracket$ where $z = x + c$.

$\llbracket y \rrbracket \leftarrow \text{RPRefresh}^\gamma(\llbracket x \rrbracket)$
 $z_1 \leftarrow y_1 + c$
return $(z_1, y_2, \dots, y_n)$

Algorithm 9 Sq^γ

Input : A n -sharing $\llbracket x \rrbracket$ of x
Output : A n -sharing $\llbracket z \rrbracket$ of $z = x^2$

$\llbracket y \rrbracket \leftarrow \text{RPRefresh}^\gamma(\llbracket x \rrbracket)$
for $i \in [1, n]$ **do**
 $z_i \leftarrow y_i^2$
end for
return $\llbracket z \rrbracket$

The following lemma provides the cardinal RPC security of the cAdd^γ gadget.

Lemma 11. *Let $n \in \mathbb{N}$. Let $\text{Refresh} : \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^R)_{t \in [0, n]})$ -cardinal RPC for some collection of envelopes $(\mathcal{E}_t^R)_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, cAdd^γ is $(p, (\mathcal{E}_t^{\text{cadd}})_{t \in [0, n]})$ where for all $t^{\text{in}} \in [0, n]$ and all $t^{\text{out}} \in [0, n]$ we have*

$$\mathcal{E}_{t^{\text{out}}}^{\text{cadd}}(t^{\text{in}}) = \begin{cases} \mathcal{E}_{t^{\text{out}}}^R(t^{\text{in}}) & \text{if } t^{\text{in}} = n \\ \max(p \cdot \mathcal{E}_{t^{\text{out}}+1}^R(t^{\text{in}}) + (1-p) \cdot \mathcal{E}_{t^{\text{out}}}^R(t^{\text{in}}), \mathcal{E}_{t^{\text{out}}}^R(t^{\text{in}})) & \text{otherwise} \end{cases}$$

D Knuth-Yates Shuffling

Algorithm 10 Knuth-Yates shuffling

Input : A n -sharing $(x_1, \dots, x_n)$

Output : A uniform permutation $(\sigma(x_1), \dots, \sigma(x_n))$ of the n -sharing of x , with $\sigma \xleftarrow{\$} S_n$

for $i \leftarrow n$ **down to** 2 **do**
 $j \xleftarrow{\$} \{1, \dots, i\}$
 $t \leftarrow x_i$
 $x_i \leftarrow x_j$
 $x_j \leftarrow t$
end for
return $(x_1, \dots, x_n)$

E 2-Stage Butterfly Network

We now consider a two-stage butterfly network, built on the same gadget. As shown in Figure 15, this circuit takes 4 masked inputs and produces 4 masked outputs, each with n shares. We perform the same tests than on the one-stage butterfly network, but this time without the general RPC notions which does not scale.

The conclusions align with those drawn for the one-stage butterfly network, though the differences between variants (v1 and v3 versus v2 and v4, which include additional refreshes) are more pronounced. Overall, the results further highlight the benefits of using the composition method grounded in the uniformly cardinal RPC framework.

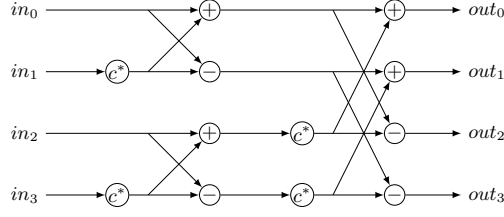


Fig. 15: 2-stages butterfly network.

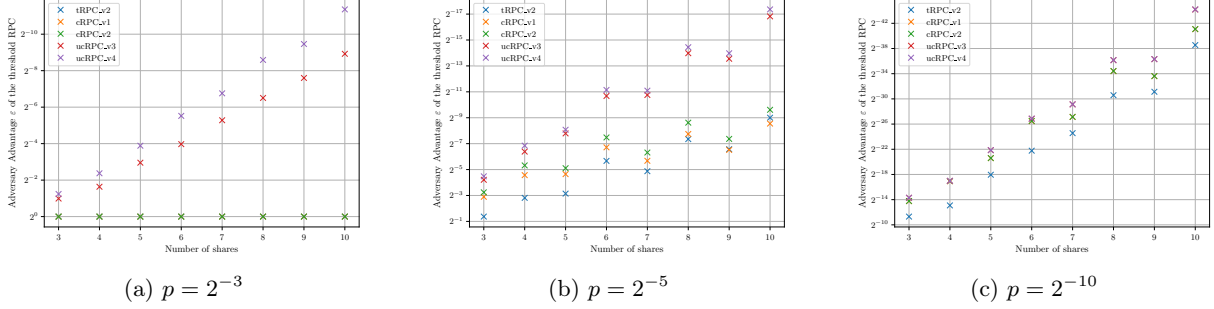


Fig. 16: Threshold RPC security of 2-stage butterfly network with respect to the number of shares n and the leakage rates p for different composition approaches.

F Proofs of Section 5

Lemma 12. *Let $n_x, n_y \in \mathbb{N}$. For any $n \in \mathbb{N}$, let $G_{\text{ref}} : \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^{G_{\text{ref}}(n)})_{t \in [0, n]})$ -cardinal RPC for some collection of envelopes $(\mathcal{E}_t^{G_{\text{ref}}(n)})_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, gadget $\text{MatMultRef} : \mathbb{K}^{n_x} \times \mathbb{K}^{n_y} \rightarrow \mathbb{K}^{n_x \cdot n_y}$ is $(p, (\mathcal{E}_t^{\text{MMR}(n_x, n_y)})_{t \in [0, n_x \cdot n_y]})$ -cardinal RPC with an envelope that can be computed by recurrence.*

Proof (Proof of Lemma 12). Using the result [8, Lemma 4], we can compute the envelope of $\text{MatMultRef}(n_x, n_y)$ by recurrence. In fact we compose the cardinal RPC envelopes $\mathcal{E}^{G_{\text{ref}}}$ of the refresh gadget G_{ref} and the cardinal RPC envelopes $\mathcal{E}^{\text{MMR}}$ of $\text{MatMultRef}(m_1, m_2)$, where $m_1 \in \{\lfloor \frac{n_x}{2} \rfloor, \lceil \frac{n_x}{2} \rceil\}$ and $m_2 \in \{\lfloor \frac{n_y}{2} \rfloor, \lceil \frac{n_y}{2} \rceil\}$. We terminate with the base cases $\text{MatMultRef}(m_1, m_2)$, where $m_1 \in \{1, 2\}$ and $m_2 \in \{1, 2\}$ for which we can assess the cardinal RPC envelopes manually. Throughout the proof, we have

$$\begin{aligned} 0 \leq j_1 &\leq \left\lfloor \frac{n_x}{2} \right\rfloor \cdot \left\lfloor \frac{n_y}{2} \right\rfloor & 0 \leq j_2 &\leq \left\lceil \frac{n_x}{2} \right\rceil \cdot \left\lfloor \frac{n_y}{2} \right\rfloor \\ 0 \leq j_3 &\leq \left\lfloor \frac{n_x}{2} \right\rfloor \cdot \left\lceil \frac{n_y}{2} \right\rceil & 0 \leq j_4 &\leq \left\lceil \frac{n_x}{2} \right\rceil \cdot \left\lceil \frac{n_y}{2} \right\rceil \end{aligned}$$

We develop the formulas hereafter. For all $t^{\text{out}} \in [0, n_x \cdot n_y]$ we have:

$$\begin{aligned} \forall \quad 0 \leq t^{\text{in}, x} < \left\lfloor \frac{n_x}{2} \right\rfloor, 0 \leq t^{\text{in}, y} < \left\lfloor \frac{n_y}{2} \right\rfloor, \quad \mathcal{E}_{t^{\text{out}}}^{\text{MMR}(n_x, n_y)}(t^{\text{in}, x}, t^{\text{in}, y}) = \\ \max_{j_1 + j_2 + j_3 + j_4 = t^{\text{out}}} \sum_{i_1^{x1}=0}^{t^{\text{in}, x}} \sum_{i_1^{x2}=0}^{i_1^{x1}} \sum_{i_1^{y1}=0}^{t^{\text{in}, y}} \sum_{i_1^{y2}=0}^{i_1^{y1}} \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i_1^{x1}, i_1^{y1}) \\ \cdot \mathcal{E}_{j_2}^{G_1}(i_1^{x2}, i_1^{y1} - i_1^{y1}) \cdot \mathcal{E}_{j_3}^{G_2}(i_1^{x1} - i_1^{x1}, i_1^{y2}) \cdot \mathcal{E}_{j_4}^{G_3}(t^{\text{in}, x} - i_1^{x1} - i_1^{x2}, t^{\text{in}, y} - i_1^{y1} - i_1^{y2}) \end{aligned}$$

$$\begin{aligned}
& \forall \left\lfloor \frac{n_x}{2} \right\rfloor \leq t^{\text{in},x} \leq n_x, 0 \leq t^{\text{in},y} < \left\lfloor \frac{n_y}{2} \right\rfloor, \quad \mathcal{E}_{t^{\text{out}}}^{\text{MMR}(n_x, n_y)}(t^{\text{in},x}, t^{\text{in},y}) = \\
& \max_{\substack{j_1, j_2, j_3, j_4 \\ j_1 + j_2 + j_3 + j_4 = t^{\text{out}}}} \sum_{\substack{i^{x1} = t^{\text{in},x} - \lfloor \frac{n_x}{2} \rfloor \\ t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) < \lceil \frac{n_x}{2} \rceil}}^{n_x} \sum_{\substack{i_1^{x1} = \max(0, i^{x1} - \lfloor \frac{n_x}{2} \rfloor) \\ i_1^{x2} = 0}}^{\min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)} \sum_{i_1^{y2} = 0}^{t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)} \\
& \sum_{i^{y1} = 0}^{t^{\text{in},y}} \sum_{i_1^{y1} = 0}^{i^{y1}} \sum_{i_1^{y2} = 0}^{t^{\text{in},y} - i^{y1}} \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i^{x1}, i_1^{y1}) \cdot \mathcal{E}_{j_2}^{G_1}(i^{x2}, i^{y1} - i_1^{y1}) \cdot \\
& \mathcal{E}_{j_3}^{G_2}(i^{x1} - i_1^{x1}, i_1^{y2}) \cdot \mathcal{E}_{j_4}^{G_3}(t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) - i_1^{x2}, t^{\text{in},y} - i^{y1} - i_1^{y2}) \\
& + \sum_{\substack{i^{x1} = t^{\text{in},x} - \lfloor \frac{n_x}{2} \rfloor \\ t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) \geq \lceil \frac{n_x}{2} \rceil}}^{n_x} \sum_{\substack{i_1^{x1} = \max(0, i^{x1} - \lfloor \frac{n_x}{2} \rfloor) \\ i^{x2} = \lceil \frac{n_x}{2} \rceil}}^{\min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)} \sum_{i^{x2} = \lceil \frac{n_x}{2} \rceil}^{n_x} \sum_{i_1^{x2} = i^{x2} - \lceil \frac{n_x}{2} \rceil}^{\lceil \frac{n_x}{2} \rceil} \\
& \sum_{i^{y1} = 0}^{t^{\text{in},y}} \sum_{i_1^{y1} = 0}^{i^{y1}} \sum_{i_1^{y2} = 0}^{t^{\text{in},y} - i^{y1}} \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i^{x1}, i_1^{y1}) \cdot \mathcal{E}_{j_2}^{G_1}(i^{x2}, i^{y1} - i_1^{y1}) \cdot \\
& \mathcal{E}_{j_3}^{G_2}(i^{x1} - i_1^{x1}, i_1^{y2}) \cdot \mathcal{E}_{j_4}^{G_3}(i^{x2} - i_1^{x2}, t^{\text{in},y} - i^{y1} - i_1^{y2})
\end{aligned}$$

$$\begin{aligned}
& \forall \quad 0 \leq t^{\text{in},x} < \left\lfloor \frac{n_x}{2} \right\rfloor, \left\lfloor \frac{n_y}{2} \right\rfloor \leq t^{\text{in},y} \leq n_y, \quad \mathcal{E}_{t^{\text{out}}}^{\text{MMR}(n_x, n_y)}(t^{\text{in},x}, t^{\text{in},y}) = \\
& \max_{\substack{j_1, j_2, j_3, j_4 \\ j_1 + j_2 + j_3 + j_4 = t^{\text{out}}}} \sum_{i^{x1} = 0}^{t^{\text{in},x}} \sum_{i_1^{x1} = 0}^{i^{x1}} \sum_{i_1^{x2} = 0}^{t^{\text{in},x} - i^{x1}} \\
& \sum_{\substack{i^{y1} = t^{\text{in},y} - \lfloor \frac{n_y}{2} \rfloor \\ t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) < \lceil \frac{n_y}{2} \rceil}}^{n_y} \sum_{\substack{i_1^{y1} = \max(0, i^{y1} - \lfloor \frac{n_y}{2} \rfloor) \\ i_1^{y2} = 0}}^{\min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)} \sum_{i_1^{y2} = 0}^{t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)} \\
& \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i^{x1}, i_1^{y1}) \cdot \mathcal{E}_{j_2}^{G_1}(i^{x2}, i^{y1} - i_1^{y1}) \cdot \mathcal{E}_{j_3}^{G_2}(i^{x1} - i_1^{x1}, i_1^{y2}) \\
& \cdot \mathcal{E}_{j_4}^{G_3}(t^{\text{in},x} - i^{x1} - i_1^{x2}, t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) - i_1^{y2}) \\
& + \sum_{i^{x1} = 0}^{t^{\text{in},x}} \sum_{i_1^{x1} = 0}^{i^{x1}} \sum_{i_1^{x2} = 0}^{t^{\text{in},x} - i^{x1}} \\
& \sum_{\substack{i^{y1} = t^{\text{in},y} - \lfloor \frac{n_y}{2} \rfloor \\ t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) \geq \lceil \frac{n_y}{2} \rceil}}^{n_y} \sum_{\substack{i_1^{y1} = \max(0, i^{y1} - \lfloor \frac{n_y}{2} \rfloor) \\ i^{y2} = \lceil \frac{n_y}{2} \rceil}}^{\min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)} \sum_{i^{y2} = \lceil \frac{n_y}{2} \rceil}^{n_y} \sum_{i_1^{y2} = i^{y2} - \lceil \frac{n_y}{2} \rceil}^{\lceil \frac{n_y}{2} \rceil} \\
& \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i^{x1}, i_1^{y1}) \cdot \mathcal{E}_{j_2}^{G_1}(i^{x2}, i^{y1} - i_1^{y1}) \cdot \mathcal{E}_{j_3}^{G_2}(i^{x1} - i_1^{x1}, i_1^{y2}) \\
& \cdot \mathcal{E}_{j_4}^{G_3}(t^{\text{in},x} - i^{x1} - i_1^{x2}, i^{y2} - i_1^{y2})
\end{aligned}$$

$$\begin{aligned}
& \forall \left\lfloor \frac{n_x}{2} \right\rfloor \leq t^{\text{in},x} \leq n_x, \left\lfloor \frac{n_y}{2} \right\rfloor \leq t^{\text{in},y} \leq n_y, \mathcal{E}_{t^{\text{out}}}^{\text{MMR}(n_x, n_y)}(t^{\text{in},x}, t^{\text{in},y}) = \\
& \max_{\substack{j_1, j_2, j_3, j_4 \\ j_1 + j_2 + j_3 + j_4 = t^{\text{out}}}} \sum_{\substack{i^{x1} = t^{\text{in},x} - \lfloor \frac{n_x}{2} \rfloor \\ t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) < \lceil \frac{n_x}{2} \rceil}}^{\sum_{i^{x1} = \max(0, i^{x1} - \lfloor \frac{n_x}{2} \rfloor)}^{\min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)}} \sum_{i_1^{x2}=0}^{t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)} \\
& \sum_{\substack{i^{y1} = t^{\text{in},y} - \lfloor \frac{n_y}{2} \rfloor \\ t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) < \lceil \frac{n_y}{2} \rceil}}^{\sum_{i_1^{y2}=0}^{\min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)}} \sum_{i_1^{y2}=0}^{t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)} \\
& \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i_1^{x1}, i_1^{y1}) \cdot \mathcal{E}_{j_2}^{G_1}(i_1^{x2}, i^{y1} - i_1^{y1}) \cdot \mathcal{E}_{j_3}^{G_2}(i^{x1} - i_1^{x1}, i_1^{y2}) \\
& \cdot \mathcal{E}_{j_4}^{G_3}(t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) - i_1^{x2}, t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) - i_1^{y2}) \\
& + \sum_{\substack{i^{x1} = t^{\text{in},x} - \lfloor \frac{n_x}{2} \rfloor \\ t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) < \lceil \frac{n_x}{2} \rceil}}^{\sum_{i_1^{x1} = \max(0, i^{x1} - \lfloor \frac{n_x}{2} \rfloor)}^{\min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)}} \sum_{i_1^{x2}=0}^{t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)} \\
& \sum_{\substack{i^{y1} = t^{\text{in},y} - \lfloor \frac{n_y}{2} \rfloor \\ t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) \geq \lceil \frac{n_y}{2} \rceil}}^{\sum_{i_1^{y1} = \max(0, i^{y1} - \lfloor \frac{n_y}{2} \rfloor)}^{\min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)}} \sum_{i^{y2} = \lceil \frac{n_y}{2} \rceil}^{n_y} \sum_{i_1^{y2} = i^{y2} - \lceil \frac{n_y}{2} \rceil}^{\lceil \frac{n_y}{2} \rceil} \\
& \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i_1^{x1}, i_1^{y1}) \cdot \mathcal{E}_{j_2}^{G_1}(i_1^{x2}, i^{y1} - i_1^{y1}) \cdot \mathcal{E}_{j_3}^{G_2}(i^{x1} - i_1^{x1}, i_1^{y2}) \\
& \cdot \mathcal{E}_{j_4}^{G_3}(t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) - i_1^{x2}, i^{y2} - i_1^{y2}) \\
& + \sum_{\substack{i^{x1} = t^{\text{in},x} - \lfloor \frac{n_x}{2} \rfloor \\ t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) \geq \lceil \frac{n_x}{2} \rceil}}^{\sum_{i_1^{x1} = \max(0, i^{x1} - \lfloor \frac{n_x}{2} \rfloor)}^{\min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)}} \sum_{i^{x2} = \lceil \frac{n_x}{2} \rceil}^{n_x} \sum_{i_1^{x2} = i^{x2} - \lceil \frac{n_x}{2} \rceil}^{\lceil \frac{n_x}{2} \rceil} \\
& \sum_{\substack{i^{y1} = t^{\text{in},y} - \lfloor \frac{n_y}{2} \rfloor \\ t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) < \lceil \frac{n_y}{2} \rceil}}^{\sum_{i_1^{y1} = \max(0, i^{y1} - \lfloor \frac{n_y}{2} \rfloor)}^{\min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)}} \sum_{i_1^{y2}=0}^{t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)} \\
& \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i_1^{x1}, i_1^{y1}) \cdot \mathcal{E}_{j_2}^{G_1}(i_1^{x2}, i^{y1} - i_1^{y1}) \cdot \mathcal{E}_{j_3}^{G_2}(i^{x1} - i_1^{x1}, i_1^{y2}) \\
& \cdot \mathcal{E}_{j_4}^{G_3}(i^{x2} - i_1^{x2}, t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) - i_1^{y2}) \\
& + \sum_{\substack{i^{x1} = t^{\text{in},x} - \lfloor \frac{n_x}{2} \rfloor \\ t^{\text{in},x} - \min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor) \geq \lceil \frac{n_x}{2} \rceil}}^{\sum_{i_1^{x1} = \max(0, i^{x1} - \lfloor \frac{n_x}{2} \rfloor)}^{\min(i^{x1}, \lfloor \frac{n_x}{2} \rfloor)}} \sum_{i^{x2} = \lceil \frac{n_x}{2} \rceil}^{n_x} \sum_{i_1^{x2} = i^{x2} - \lceil \frac{n_x}{2} \rceil}^{\lceil \frac{n_x}{2} \rceil} \\
& \sum_{\substack{i^{y1} = t^{\text{in},y} - \lfloor \frac{n_y}{2} \rfloor \\ t^{\text{in},y} - \min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor) \geq \lceil \frac{n_y}{2} \rceil}}^{\sum_{i_1^{y1} = \max(0, i^{y1} - \lfloor \frac{n_y}{2} \rfloor)}^{\min(i^{y1}, \lfloor \frac{n_y}{2} \rfloor)}} \sum_{i^{y2} = \lceil \frac{n_y}{2} \rceil}^{n_y} \sum_{i_1^{y2} = i^{y2} - \lceil \frac{n_y}{2} \rceil}^{\lceil \frac{n_y}{2} \rceil} \\
& \mathcal{E}_{j_1}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(i_1^{x1}, i_1^{y1}) \cdot \mathcal{E}_{j_2}^{G_1}(i_1^{x2}, i^{y1} - i_1^{y1}) \cdot \mathcal{E}_{j_3}^{G_2}(i^{x1} - i_1^{x1}, i_1^{y2}) \\
& \cdot \mathcal{E}_{j_4}^{G_3}(i^{x2} - i_1^{x2}, i^{y2} - i_1^{y2})
\end{aligned}$$

where the envelopes $\mathcal{E}^{G_1}$, $\mathcal{E}^{G_2}$ and $\mathcal{E}^{G_3}$ are on the form:

$$\begin{aligned}\mathcal{E}_{j_2}^{G_1}(i_1^{x_2}, i_2^{y_1}) &= \sum_{\ell=0}^{\lfloor \frac{n_y}{2} \rfloor} \mathcal{E}_{j_2}^{\text{MMR}(\lceil \frac{n_x}{2} \rceil, \lfloor \frac{n_y}{2} \rfloor)}(i_1^{x_2}, \ell) \cdot \mathcal{E}_{\ell}^{\text{Gref}(\lfloor \frac{n_y}{2} \rfloor)}(i_2^{y_1}) \\ \mathcal{E}_{j_3}^{G_2}(i_2^{x_1}, i_1^{y_2}) &= \sum_{\ell=0}^{\lfloor \frac{n_x}{2} \rfloor} \mathcal{E}_{j_3}^{\text{MMR}(\lfloor \frac{n_x}{2} \rfloor, \lceil \frac{n_y}{2} \rceil)}(\ell, i_1^{y_2}) \cdot \mathcal{E}_{\ell}^{\text{Gref}(\lfloor \frac{n_x}{2} \rfloor)}(i_2^{x_1}) \\ \mathcal{E}_{j_4}^{G_3}(i_2^{x_2}, i_2^{y_2}) &= \sum_{\ell_1=0}^{\lfloor \frac{n_x}{2} \rfloor} \sum_{\ell_2=0}^{\lfloor \frac{n_y}{2} \rfloor} \mathcal{E}_{j_4}^{\text{MMR}(\lceil \frac{n_x}{2} \rceil, \lceil \frac{n_y}{2} \rceil)}(\ell_1, \ell_2) \cdot \mathcal{E}_{\ell_1}^{\text{Gref}(\lceil \frac{n_x}{2} \rceil)}(i_2^{x_2}) \cdot \mathcal{E}_{\ell_2}^{\text{Gref}(\lceil \frac{n_y}{2} \rceil)}(i_2^{y_2})\end{aligned}$$

for any $i_2^{x_1} \in [0, \lfloor \frac{n_x}{2} \rfloor]$, $i_2^{y_1} \in [0, \lfloor \frac{n_y}{2} \rfloor]$, $(i_1^{x_2}, i_2^{x_2}) \in [0, \lceil \frac{n_x}{2} \rceil]$, $(i_1^{y_2}, i_2^{y_2}) \in [0, \lceil \frac{n_y}{2} \rceil]$. *Initialisation.* We compute manually the cardinal RPC envelopes of $\text{MatMultRef}(n_x, n_y)$ for $n_x \in \{1, 2\}$, $n_y \in \{1, 2\}$. We start with $n_x = 1$ and $n_y = 1$.

For $t^{\text{out}} = 0$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1
0	$(1-p)^2$	$p \cdot (1-p)$
1	$p \cdot (1-p)$	p^2

For $t^{\text{out}} = 1$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1
0	0	0
1	0	1

For $n_x = 1$ and $n_y = 2$, we have :

For $t^{\text{out}} = 0$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1	2
0	$(1-p)^3$	$2 \cdot p \cdot (1-p)^2$	$p^2 \cdot (1-p)$
1	$p \cdot (1-p)^2$	$2 \cdot p^2 \cdot (1-p)$	p^3

For $t^{\text{out}} = 1$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1	2
0	0	0	0
1	0	$1-p$	p

For $t^{\text{out}} = 2$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1	2
0	0	0	0
1	0	0	1

For $n_x = 2$ and $n_y = 1$, the array is the transpose of the above array. Eventually, for $n_x = 2$ and $n_y = 2$, if we note $v = (1-p)^3$ and $w = 1-p$, we have :

For $t^{\text{out}} = 0$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1	2
0	v^4	$2 \cdot w \cdot v^3$	$v^2 \cdot w^2$
1	$2 \cdot w \cdot v^3$	$4 \cdot w^2 \cdot v^2$	$2 \cdot w^3 \cdot v$
2	$w^2 \cdot v^2$	$2 \cdot w^3 \cdot v$	w^4

For $t^{\text{out}} = 1$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1	2
0	0	0	0
1	0	v^2	$w \cdot v$
2	0	$w \cdot v$	w^2

For $t^{\text{out}} = 2$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1	2
0	0	0	0
1	0	0	v
2	0	v	1

For $t^{\text{out}} = 3$ or $t^{\text{out}} = 4$:

$t^{\text{in},y} \backslash t^{\text{in},x}$	0	1	2
0	0	0	0
1	0	0	0
2	0	0	1

□

MatMultSym is formally described in Algorithm 11.

Algorithm 11 MatMultSym

Input: n_x -sharing $\llbracket x \rrbracket$ and n_y -sharing $\llbracket y \rrbracket$
Output: $(n_x \cdot n_y)$ -sharing $\mathbf{M} = (M_{i,j})_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}}$ such that $\sum_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}} M_{i,j} = x \cdot y$

```

if  $n_x = 1$  and  $n_y = 1$  then
     $\mathbf{M} \leftarrow (x_1 \cdot y_1)$ 
else if  $n_x = 1$  and  $n_y = 2$  then
     $\mathbf{M} \leftarrow (x_1 \cdot y_1 \ x_1 \cdot y_2)$ 
else if  $n_x = 2$  and  $n_y = 1$  then
     $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 \\ x_2 \cdot y_1 \end{pmatrix}$ 
else if  $n_x = 2$  and  $n_y = 2$  then
     $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 & x_1 \cdot y_2 \\ x_2 \cdot y_1 & x_2 \cdot y_2 \end{pmatrix}$ 
else
     $\mathbf{x}^{(1)} \leftarrow (x_1, \dots, x_{\lfloor \frac{n_x}{2} \rfloor})$ ,  $\mathbf{x}^{(2)} \leftarrow (x_{\lceil \frac{n_x}{2} \rceil}, \dots, x_{n_x})$ 
     $\mathbf{y}^{(1)} \leftarrow (y_1, \dots, y_{\lfloor \frac{n_y}{2} \rfloor})$ ,  $\mathbf{y}^{(2)} \leftarrow (y_{\lceil \frac{n_y}{2} \rceil}, \dots, y_{n_y})$ 
     $\mathbf{x}_1^{(1)} \leftarrow \text{RPRefresh}(\mathbf{x}^{(1)})$ ,  $\mathbf{x}_1^{(2)} \leftarrow \text{RPRefresh}(\mathbf{x}^{(2)})$ 
     $\mathbf{y}_1^{(1)} \leftarrow \text{RPRefresh}(\mathbf{y}^{(1)})$ ,  $\mathbf{y}_1^{(2)} \leftarrow \text{RPRefresh}(\mathbf{y}^{(2)})$ 
     $\mathbf{M}^{(1,1)} \leftarrow \text{UnifMatMult}(\mathbf{x}_1^{(1)}, \mathbf{y}_1^{(1)})$ 
     $\mathbf{x}_2^{(1)} \leftarrow \text{RPRefresh}(\mathbf{x}^{(1)})$ ,  $\mathbf{y}_2^{(1)} \leftarrow \text{RPRefresh}(\mathbf{y}^{(1)})$ 
     $\mathbf{M}^{(1,2)} \leftarrow \text{UnifMatMult}(\mathbf{x}_2^{(1)}, \mathbf{y}_1^{(2)})$ 
     $\mathbf{M}^{(2,1)} \leftarrow \text{UnifMatMult}(\mathbf{x}_1^{(2)}, \mathbf{y}_2^{(1)})$ 
     $\mathbf{x}_2^{(2)} \leftarrow \text{RPRefresh}(\mathbf{x}^{(2)})$ ,  $\mathbf{y}_2^{(2)} \leftarrow \text{RPRefresh}(\mathbf{y}^{(2)})$ 
     $\mathbf{M}^{(2,2)} \leftarrow \text{UnifMatMult}(\mathbf{x}_2^{(2)}, \mathbf{y}_2^{(2)})$ 
     $\mathbf{M} = \begin{pmatrix} \mathbf{M}^{(1,1)} & \mathbf{M}^{(1,2)} \\ \mathbf{M}^{(2,1)} & \mathbf{M}^{(2,2)} \end{pmatrix}$ 
end if
return  $\mathbf{M}$ 

```

Lemma 13. Let $n_x, n_y \in \mathbb{N}$. For any $n \in \mathbb{N}$, let $\mathbf{G}_{\text{ref}} : \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^{\mathbf{G}_{\text{ref}}(n)})_{t \in [0, n]})$ -cardinal RPC for some collection of envelopes $(\mathcal{E}_t^{\mathbf{G}_{\text{ref}}(n)})_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, gadget $\text{MatMultSym} : \mathbb{K}^{n_x} \times \mathbb{K}^{n_y} \rightarrow \mathbb{K}^{n_x \cdot n_y}$ is $(p, (\mathcal{E}_t^{\text{MMS}(n_x, n_y)})_{t \in [0, n_x \cdot n_y]})$ -cardinal RPC with an envelope that can be computed by recurrence.

Proof (Proof of Lemma 13). The proof is similar to the proof of Lemma 12. The envelopes of the base case with $n_x \in \{1, 2\}$ and $n_y \in \{1, 2\}$ are exactly the same. As for the induction step, the only differences is that we replace $\mathcal{E}_{t_{\text{out}}}^{\text{MMR}}, \mathcal{E}_{t_{\text{out}}}^{G_1}, \mathcal{E}_{t_{\text{out}}}^{G_2}, \mathcal{E}_{t_{\text{out}}}^{G_3}$ by $\mathcal{E}_{t_{\text{out}}}^{G, (m_1, m_2)}$ where $m_1 \in \{\lfloor \frac{n_x}{2} \rfloor, \lceil \frac{n_x}{2} \rceil\}$ and $m_2 \in \{\lfloor \frac{n_y}{2} \rfloor, \lceil \frac{n_y}{2} \rceil\}$ (according to the considered case) where for all $t_1^{\text{in}} \in [0, m_1], t_2^{\text{in}} \in [0, m_2]$ we have

$$\mathcal{E}_{t_{\text{out}}}^{G, (m_1, m_2)}(t_1^{\text{in}}, t_2^{\text{in}}) = \sum_{\ell_1=0}^{m_1} \sum_{\ell_2=0}^{m_2} \mathcal{E}_{t_{\text{out}}}^{\text{MMR}(m_1, m_2)}(\ell_1, \ell_2) \cdot \mathcal{E}_{\ell_1}^{\mathbf{G}_{\text{ref}}(m_1)}(t_1^{\text{in}}) \cdot \mathcal{E}_{\ell_2}^{\mathbf{G}_{\text{ref}}(m_2)}(t_2^{\text{in}})$$

□

UnifMatMult is formally described in Algorithm 12.

Algorithm 12 UnifMatMult

Input: n_x -sharing $\llbracket x \rrbracket$ and n_y -sharing $\llbracket y \rrbracket$

Output: $(n_x \cdot n_y)$ -sharing $\mathbf{M} = (M_{i,j})_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}}$ such that $\sum_{\substack{1 \leq i \leq n_x \\ 1 \leq j \leq n_y}} M_{i,j} = x \cdot y$

```

if  $n_x = 1$  and  $n_y = 1$  then
   $\mathbf{M} \leftarrow (x_1 \cdot y_1)$ 
else if  $n_x = 1$  and  $n_y = 2$  then
   $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 & x_1 \cdot y_2 \end{pmatrix}$ 
else if  $n_x = 2$  and  $n_y = 1$  then
   $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 \\ x_2 \cdot y_1 \end{pmatrix}$ 
else if  $n_x = 2$  and  $n_y = 2$  then
   $\mathbf{M} \leftarrow \begin{pmatrix} x_1 \cdot y_1 & x_1 \cdot y_2 \\ x_2 \cdot y_1 & x_2 \cdot y_2 \end{pmatrix}$ 
else
   $\mathbf{x}^{(1)} \leftarrow (x_1, \dots, x_{\lfloor \frac{n_x}{2} \rfloor})$ ,  $\mathbf{x}^{(2)} \leftarrow (x_{\lceil \frac{n_x}{2} \rceil}, \dots, x_{n_x})$ 
   $\mathbf{y}^{(1)} \leftarrow (y_1, \dots, y_{\lfloor \frac{n_y}{2} \rfloor})$ ,  $\mathbf{y}^{(2)} \leftarrow (y_{\lceil \frac{n_y}{2} \rceil}, \dots, y_{n_y})$ 

   $\sigma_1, \sigma_2, \sigma_3, \sigma_4 \xleftarrow{\$} \mathcal{S}_*$ 
   $\mathbf{x}_1^{(1)} \leftarrow \sigma_1(\mathbf{x}^{(1)})$ ,  $\mathbf{x}_1^{(2)} \leftarrow \sigma_2(\mathbf{x}^{(2)})$ 
   $\mathbf{y}_1^{(1)} \leftarrow \sigma_3(\mathbf{y}^{(1)})$ ,  $\mathbf{y}_1^{(2)} \leftarrow \sigma_4(\mathbf{y}^{(2)})$ 
   $\mathbf{x}_1^{(1)} \leftarrow \text{RPRfresh}(\mathbf{x}_1^{(1)})$ ,  $\mathbf{x}_1^{(2)} \leftarrow \text{RPRfresh}(\mathbf{x}_1^{(2)})$ 
   $\mathbf{y}_1^{(1)} \leftarrow \text{RPRfresh}(\mathbf{y}_1^{(1)})$ ,  $\mathbf{y}_1^{(2)} \leftarrow \text{RPRfresh}(\mathbf{y}_1^{(2)})$ 
   $\mathbf{M}^{(1,1)} \leftarrow \text{UnifMatMult}(\mathbf{x}_1^{(1)}, \mathbf{y}_1^{(1)})$ 
   $\mathbf{x}_2^{(1)} \leftarrow \text{RPRfresh}(\mathbf{x}_1^{(1)})$ ,  $\mathbf{y}_2^{(1)} \leftarrow \text{RPRfresh}(\mathbf{y}_1^{(1)})$ 
   $\mathbf{M}^{(1,2)} \leftarrow \text{UnifMatMult}(\mathbf{x}_2^{(1)}, \mathbf{y}_1^{(2)})$ 
   $\mathbf{M}^{(2,1)} \leftarrow \text{UnifMatMult}(\mathbf{x}_1^{(2)}, \mathbf{y}_2^{(1)})$ 
   $\mathbf{x}_2^{(2)} \leftarrow \text{RPRfresh}(\mathbf{x}_1^{(2)})$ ,  $\mathbf{y}_2^{(2)} \leftarrow \text{RPRfresh}(\mathbf{y}_2^{(1)})$ 
   $\mathbf{M}^{(2,2)} \leftarrow \text{UnifMatMult}(\mathbf{x}_2^{(2)}, \mathbf{y}_2^{(2)})$ 
   $\mathbf{M} = \begin{pmatrix} \mathbf{M}^{(1,1)} & \mathbf{M}^{(1,2)} \\ \mathbf{M}^{(2,1)} & \mathbf{M}^{(2,2)} \end{pmatrix}$ 
end if
return  $\mathbf{M}$ 

```

Lemma 14. Let $n_x, n_y \in \mathbb{N}$. For any $n \in \mathbb{N}$, let $\mathbf{G}_{\text{ref}} : \mathbb{K}^n \rightarrow \mathbb{K}^n$ be an n -share refresh gadget satisfying $(p, (\mathcal{E}_t^{\mathbf{G}_{\text{ref}}(n)})_{t \in [0, n]})$ -cardinal RPC for some collection of envelopes $(\mathcal{E}_t^{\mathbf{G}_{\text{ref}}(n)})_{t \in [0, n]}$ and some probability $p \in [0, 1]$. Then, gadget $\text{UnifMatMult} : \mathbb{K}^{n_x} \times \mathbb{K}^{n_y} \rightarrow \mathbb{K}^{n_x \cdot n_y}$ is $(p, (\mathcal{E}_t^{\text{UMM}(n_x, n_y)})_{t \in [0, n_x \cdot n_y]})$ -cardinal RPC with an envelope that can be computed by recurrence.

Proof (Proof of Lemma 14). Using the result [8, Lemma 4], we can compute the envelope of $\text{UnifMatMult}(n_x, n_y)$ by recurrence. In fact we compose the cardinal RPC envelopes $\mathcal{E}^{\mathbf{G}_{\text{ref}}}$ of the refresh gadget $\mathbf{G}_{\text{ref}}$ and the cardinal RPC envelopes $\mathcal{E}^{\text{UMM}}$ of $\text{UnifMatMult}(m_1, m_2)$, where $m_1 \in \{\lfloor \frac{n_x}{2} \rfloor, \lceil \frac{n_x}{2} \rceil\}$ and $m_2 \in \{\lfloor \frac{n_y}{2} \rfloor, \lceil \frac{n_y}{2} \rceil\}$. We terminate with the base cases $\text{UnifMatMult}(m_1, m_2)$, where $m_1 \in \{1, 2\}$ and $m_2 \in \{1, 2\}$ for which we can assess the cardinal RPC envelopes manually. Throughout the proof, we have

$$\begin{aligned}
 0 \leq j_1 &\leq \left\lfloor \frac{n_x}{2} \right\rfloor \cdot \left\lfloor \frac{n_y}{2} \right\rfloor & 0 \leq j_2 &\leq \left\lceil \frac{n_x}{2} \right\rceil \cdot \left\lfloor \frac{n_y}{2} \right\rfloor \\
 0 \leq j_3 &\leq \left\lfloor \frac{n_x}{2} \right\rfloor \cdot \left\lceil \frac{n_y}{2} \right\rceil & 0 \leq j_4 &\leq \left\lceil \frac{n_x}{2} \right\rceil \cdot \left\lceil \frac{n_y}{2} \right\rceil
 \end{aligned}$$

We develop the formulas hereafter. For all $t^{\text{out}} \in [0, n_x \cdot n_y]$ we have:

$$\begin{aligned}
\mathcal{E}_{t^{\text{out}}}^{\text{UMM}(n_x, n_y)}(t^{\text{in}, x}, t^{\text{in}, y}) = & \sum_{\substack{j_1, j_2, j_3, j_4 \\ j_1 + j_2 + j_3 + j_4 = t^{\text{out}}}} \frac{1}{\#\{(j_1, j_2, j_3, j_4) s.t. j_1 + j_2 + j_3 + j_4 = t^{\text{out}}\}} \\
& \sum_{i^{x_1} = \max(0, t^{\text{in}, x} - \lfloor \frac{n_x}{2} \rfloor)}^{\min(t^{\text{in}, x}, \lfloor \frac{n_x}{2} \rfloor)} \sum_{i_1^{x_1} = 0}^{i^{x_1}} \sum_{i_2^{x_1} = i^{x_1} - i_1^{x_1}}^{t^{\text{in}, x} - i^{x_1}} \sum_{i_1^{x_2} = 0}^{t^{\text{in}, x} - i^{x_1}} \sum_{i_2^{x_2} = t^{\text{in}, x} - i^{x_1} - i_1^{x_2}}^{t^{\text{in}, x} - i^{x_1} - i_1^{x_2}} \\
& \sum_{i^{y_1} = \max(0, t^{\text{in}, y} - \lfloor \frac{n_y}{2} \rfloor)}^{\min(t^{\text{in}, y}, \lfloor \frac{n_y}{2} \rfloor)} \sum_{i_1^{y_1} = 0}^{i^{y_1}} \sum_{i_2^{y_1} = i^{y_1} - i_1^{y_1}}^{t^{\text{in}, y} - i^{y_1}} \sum_{i_1^{y_2} = 0}^{t^{\text{in}, y} - i^{y_1}} \sum_{i_2^{y_2} = t^{\text{in}, y} - i^{y_1} - i_1^{y_2}}^{t^{\text{in}, y} - i^{y_1} - i_1^{y_2}} \\
& \frac{\binom{i_1^{x_1}}{i_1^{x_1} + i_2^{x_1} - i^{x_1}} \cdot \binom{\lfloor \frac{n_x}{2} \rfloor - i_1^{x_1}}{i^{x_1} - i_1^{x_1}}}{\binom{\lfloor \frac{n_x}{2} \rfloor}{i_2^{x_1}}} \cdot \frac{\binom{i_1^{x_2}}{i_1^{x_2} + i_2^{x_2} - t^{\text{in}, x} + i^{x_1}} \cdot \binom{\lfloor \frac{n_x}{2} \rfloor - i_1^{x_2}}{t^{\text{in}, x} - i^{x_1} - i_1^{x_2}}}{\binom{\lfloor \frac{n_x}{2} \rfloor}{i_2^{x_2}}} \\
& \frac{\binom{i_1^{y_1}}{i_1^{y_1} + i_2^{y_1} - i^{y_1}} \cdot \binom{\lfloor \frac{n_y}{2} \rfloor - i_1^{y_1}}{i^{y_1} - i_1^{y_1}}}{\binom{\lfloor \frac{n_y}{2} \rfloor}{i_2^{y_1}}} \cdot \frac{\binom{i_1^{y_2}}{i_1^{y_2} + i_2^{y_2} - t^{\text{in}, y} + i^{y_1}} \cdot \binom{\lfloor \frac{n_y}{2} \rfloor - i_1^{y_2}}{t^{\text{in}, y} - i^{y_1} - i_1^{y_2}}}{\binom{\lfloor \frac{n_y}{2} \rfloor}{i_2^{y_2}}} \\
& \mathcal{E}_{j_1}^1(i_1^{x_1}, i_1^{y_1}) \cdot \mathcal{E}_{j_2}^2(i_1^{x_2}, i_2^{y_1}) \cdot \mathcal{E}_{j_3}^3(i_2^{x_1}, i_1^{y_2}) \mathcal{E}_{j_4}^4(i_2^{x_2}, i_2^{y_2})
\end{aligned}$$

where the envelopes $\mathcal{E}^1, \mathcal{E}^2, \mathcal{E}^3$ and $\mathcal{E}^4$ are on the form :

$$\begin{aligned}
\mathcal{E}_{j_1}^1(i_1^{x_1}, i_1^{y_1}) &= \sum_{\ell_1=0}^{\lfloor \frac{n_x}{2} \rfloor} \sum_{\ell_2=0}^{\lfloor \frac{n_y}{2} \rfloor} \mathcal{E}_{j_1}^{\text{UMM}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(\ell_1, \ell_2) \cdot \mathcal{E}_{\ell_1}^{\text{Gref}(\lfloor \frac{n_x}{2} \rfloor)}(i_1^{x_1}) \cdot \mathcal{E}_{\ell_2}^{\text{Gref}(\lfloor \frac{n_y}{2} \rfloor)}(i_1^{y_1}) \\
\mathcal{E}_{j_2}^2(i_1^{x_2}, i_2^{y_1}) &= \sum_{\ell_1=0}^{\lfloor \frac{n_x}{2} \rfloor} \sum_{\ell_2=0}^{\lfloor \frac{n_y}{2} \rfloor} \mathcal{E}_{j_2}^{\text{UMM}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(\ell_1, \ell_2) \cdot \mathcal{E}_{\ell_1}^{\text{Gref}(\lfloor \frac{n_x}{2} \rfloor)}(i_1^{x_2}) \cdot \mathcal{E}_{\ell_2}^{\text{Gref}(\lfloor \frac{n_y}{2} \rfloor)}(i_2^{y_1}) \\
\mathcal{E}_{j_3}^3(i_2^{x_1}, i_1^{y_2}) &= \sum_{\ell_1=0}^{\lfloor \frac{n_x}{2} \rfloor} \sum_{\ell_2=0}^{\lfloor \frac{n_y}{2} \rfloor} \mathcal{E}_{j_3}^{\text{UMM}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(\ell_1, \ell_2) \cdot \mathcal{E}_{\ell_1}^{\text{Gref}(\lfloor \frac{n_x}{2} \rfloor)}(i_2^{x_1}) \cdot \mathcal{E}_{\ell_2}^{\text{Gref}(\lfloor \frac{n_y}{2} \rfloor)}(i_1^{y_2}) \\
\mathcal{E}_{j_4}^4(i_2^{x_2}, i_2^{y_2}) &= \sum_{\ell_1=0}^{\lfloor \frac{n_x}{2} \rfloor} \sum_{\ell_2=0}^{\lfloor \frac{n_y}{2} \rfloor} \mathcal{E}_{j_4}^{\text{UMM}(\lfloor \frac{n_x}{2} \rfloor, \lfloor \frac{n_y}{2} \rfloor)}(\ell_1, \ell_2) \cdot \mathcal{E}_{\ell_1}^{\text{Gref}(\lfloor \frac{n_x}{2} \rfloor)}(i_2^{x_2}) \cdot \mathcal{E}_{\ell_2}^{\text{Gref}(\lfloor \frac{n_y}{2} \rfloor)}(i_2^{y_2})
\end{aligned}$$

Initialisation. The cardinal RPC envelopes for $(n_x, n_y) \in \{(1, 1), (1, 2), (2, 1), (2, 2)\}$ are the same than in proof of Lemma 12. □

Résumé

Si les schémas cryptographiques offrent une résistance théorique contre des adversaires limités à l'observation d'entrées et de sorties, leurs implémentations physiques restent vulnérables aux *attaques par canaux auxiliaires*, qui exploitent les fuites physiques du dispositif exécutant l'algorithme (telles que le temps d'exécution, la consommation de courant ou les émissions électromagnétiques) pour retrouver leurs secrets.

Une contre-mesure classique consiste à *masquer* les données sensibles, en les divisant en n parts de sorte que tout ensemble de $n - 1$ parts reste indépendant du secret. Comme chaque part fuit une fonction bruitée de sa valeur, la recombinaison du secret devient exponentiellement difficile à mesure que n augmente. Dans les schémas de masquage booléen ou arithmétique, les opérations linéaires sont naturellement dupliquées sur chaque part, tandis que les opérations non linéaires nécessitent de l'aléa supplémentaire pour éviter que les fuites issues d'opérations individuelles ne révèlent plusieurs parts à la fois.

Pour raisonner sur la sécurité de ces implémentations, la communauté s'appuie sur des *modèles de fuite* décrivant la connaissance accessible à l'attaquant. Parmi eux, le modèle *noisy* est reconnu pour refléter fidèlement le comportement physique des dispositifs embarqués, toutefois, sa complexité le rend difficile à exploiter dans la construction de preuves. À l'inverse, le modèle *random probing*, dans lequel chaque variable fuit avec une probabilité p , admet une réduction fine vers le modèle *noisy*, tout en restant adapté à la construction de preuves théoriques.

Ce manuscrit présente mes contributions, obtenues en collaboration avec mes co-auteurs du monde académique et de l'industrie, à la sécurité du masquage dans le modèle *random probing*. Après avoir formalisé les notions associées, nous avons proposé une méthodologie pratique d'évaluation de la sécurité de petits circuits (qui sert de base à plusieurs vérificateurs automatiques) ainsi qu'une chaîne de transformations d'implémentations sûres dans le modèle *random probing* en des implémentations sûres sur des microcontrôleurs.

Pour franchir l'obstacle de l'échelle, nous avons adopté une approche modulaire par composition : à partir de petits circuits masqués (ou *gadgets*) satisfaisant des propriétés de composabilité, nous obtenons des garanties de sécurité globales. Dans ce cadre, nous avons formalisé la notion de *threshold random probing composability* qui, lorsqu'elle est vérifiée par les gadgets constituant le circuit, permet d'établir la sécurité globale dans le modèle *random probing*. Nous avons également introduit des variantes plus complexes, *cardinal random probing composability* et *uniformly cardinal random probing composability*, offrant des bornes de sécurité plus précises.

Enfin, nous avons conçu des gadgets compatibles avec ces nouvelles notions de composition. Une première stratégie permet d'atteindre des niveaux de sécurité arbitrairement élevés, au prix d'une complexité importante en nombre d'opérations et en quantité d'aléa. Une seconde approche génère des gadgets plus efficaces, dont le niveau de sécurité peut être évalué jusqu'à un certain stade (dépendant du nombre d'entrées, de sorties et de parts n). Une étude de cas sur l'AES illustre concrètement les avantages de ces nouvelles notions.