SYSTEMATIC
PARIS REGION SYSTEMS & ICT CLUSTER

VERISICC
Deliverable L2.2
Formal methods for the verification of countermeasures

CALL: FUI25

NAME OF THIS PROJECT: VERISICC

Leader of this deliverable

- Partner: CryptoExperts
- Contact name: Sonia Belaïd
- Contact information: sonia.belaid@cryptoexperts.com

Leader of the project

- Company: CryptoExperts
- Contact name: Sonia Belaïd
- Contact information: sonia.belaid@cryptoexperts.com

Partners

- SMEs: CryptoExperts and NinjaLab
- Big Business: IDEMIA
- Public Institutions: INRIA, ANSSI, and Université du Luxembourg

Version : 1.0

Sommaire

1. Introduction

1.1. Context

As explained in previous deliverables, verification tools have been developed by the cryptographic community to verify the probing security of concrete (software) implementations. Among them, `maskVerif` [BBD+15] and `CheckMasks` [Cor18] both take a (software) implementation in input and a security order t and output either a formal proof that the implementation is secure in the t-probing model or potential sets of up to t intermediate variables that might leak in the t-probing model. While `CheckMasks` can take advantage of the structure of the algorithm it verifies with ad-hoc rules, `maskVerif` applies successively and repeatedly the same set of three rules to considered tuples (i.e., sets of t or more intermediate variables):

rule 1: check whether all the expressions in the tuple contain all the shares of at least one input;

rule 2: for every tuple element, check whether a random r additively masks a sub-expression ε (which does not involve r) and appears nowhere else in the other expressions and tuple elements; in this case replace the sum of the so-called sub-expression and r by r, namely $\varepsilon + r \leftarrow r$;

rule 3: apply mathematical simplifications on the tuple.

In this task (2.2), the consortium worked on improving the formal verification of implementations. In particular, two ideas were developed in the project's proposal: providing a more accurate verification by limiting as much as possible false negatives, (i.e., the tuples which could not be proven to be secure whereas they are) and improving the high complexity of the verification. While the consortium contributed in these two directions, it also worked on extending the verification model. So far, implementations were verified in the probing model with a few software extensions. New contributions allow the verification in the probing model with significant physical defaults and the consortium is currently working on integrating devices characterization to increase the accuracy of the verification result. In parallel, the consortium is working on verification in a more practical model, namely the *random probing model*. The latter is meant to reflect better the reality of embedded devices but makes security proofs more complicated to build.

1.2. Publications of the consortium

Two papers were published to provide the *exact* (i.e., no false negative) verification of standard circuits (with a set of common but limited gadgets) in the generic t-probing model and in an extended t-register probing model:

- **Sonia Belaïd**, Dahmun Goudarzi, **Matthieu Rivain**: *Tight Private Circuits: Achieving Probing Security with the Least Refreshing*. ASIACRYPT (2) 2018: 343-372 [BGR18]
- **Sonia Belaïd**, Pierre-Evariste Dagand, Darius Mercadier, **Matthieu Rivain**, and Raphaël Wintersdorff: *Tornado: Automatic Generation of Probing-Secure Masked Bitsliced Implementations*. To appear in the proceedings of EUROCRYPT 2020

`maskVerif` was also improved to ensure exact verification, to reduce the verification complexity, and to extend the verification towards the probing model with glitches:

- Gilles Barthe, **Sonia Belaïd**, Gaëtan Cassiers, Pierre-Alain Fouque, **Benjamin Grégoire**, François-Xavier Standaert: *maskVerif: Automated Verification of Higher-Order Masking in Presence of Physical Defaults*. ESORICS (1) 2019: 300-318 [BBC+19]

1.3. Organization

We organize the deliverable according to three contributions. In Section 3, we describe new verification methods that lead to exact security proofs without false negative. In particular, two different methods are provided to achieve such results in a generic probing model with a limited set of gadgets, and in the probing model with fixed order t for a wider range of implementations. Then in Section 4, we describe the contributions on the verification complexity. Finally in Section 5, we display the new techniques that were developed to provide verification in extended leakage models.

2. Technical background

In this section, we only recall the security notions that we use hereafter. However, we do not recall prior masking-related notions which can be found in the previous deliverables for the interested reader.

We first recall the *t-probing security* originally introduced in [ISW03] as formalized through a concrete security game in [BGR18]. It is based on two experiments described in Figure 1 from [BGR18] in which an adversary $\mathcal{A}$, modeled as a probabilistic algorithm, outputs a set of t probes $\mathcal{P}$ and n inputs $x_1, \ldots, x_n$ in a set $\mathbb{K}$. In the first experiment, ExpReal, the inputs are encoded and given as inputs to the shared circuit C. The experiment then outputs a random evaluation of the chosen probes $(v_1, \ldots, v_t)$. In the second experiment, ExpSim, the simulator outputs a simulation of the evaluation $C([x_1], \ldots, [x_n])_{\mathcal{P}}$ without the input sharings. It wins the game if and only if the distributions of both experiments are identical.

$\underline{\mathsf{ExpReal}(\mathcal{A}, C)}$:
1. $(\mathcal{P}, x_1, \ldots, x_n) \leftarrow \mathcal{A}()$
2. $[x_1] \leftarrow \mathsf{Enc}(x_1), \ldots, [x_n] \leftarrow \mathsf{Enc}(x_n)$
3. $(v_1, \ldots, v_t) \leftarrow C([x_1], \ldots, [x_n])_{\mathcal{P}}$
4. Return $(v_1, \ldots, v_t)$

$\underline{\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)}$:
1. $(\mathcal{P}, x_1, \ldots, x_n) \leftarrow \mathcal{A}()$
2. $(v_1, \ldots, v_t) \leftarrow \mathcal{S}(\mathcal{P})$
3. Return $(v_1, \ldots, v_t)$

Figure 1: t-probing security game from [BGR18].

Definition 2.1 *(from [BGR18]) A shared circuit C is t-probing secure if and only if for every adversary $\mathcal{A}$, there exists a simulator $\mathcal{S}$ that wins the t-probing security game defined in Figure 1, i.e., the random experiments* $\mathsf{ExpReal}(\mathcal{A}, C)$ *and* $\mathsf{ExpSim}(\mathcal{A}, \mathcal{S}, C)$ *output identical distributions.*

In [BGR18], the notion of t-probing security was defined for a Boolean circuit, with $\mathbb{K} = \mathbb{F}_2$, that is with $x_1, \ldots, x_n \in \mathbb{F}_2$ and $v_1, \ldots, v_t \in \mathbb{F}_2$. We further refer to this specialized notion as *t-bit probing security*.

While the notion of t-bit probing security is relevant in a hardware scenario, in the reality of masked software embedded devices, variables are manipulated in registers which contain several bits that leak all together. To capture this model, in this paper, we extend the verification to what we

call the t-*register probing model* in which the targeted circuit manipulates variables on registers of size s for some $s \in \mathbb{N}^+$ and the adversary is able to choose t probes as registers containing values in $\mathcal{V} = \mathbb{F}_2^s$. Notice that the t-bit probing model can be seen as a specialization of the t-register probing model with $s = 1$.

3. Exact verification in the probing model - no false negative

3.1. Verification from the high-level description at a generic order

CryptoExperts developed a new verification method which aims to verify the probing security of a shared Boolean circuit at any masking order t. It takes as input a list of instructions that describes a shared circuit made of specific multiplication, addition and refresh *gadgets* and outputs either a probing security proof or a probing attack. To that end, a security reduction is made through a sequence of four equivalent games. In each of them, an adversary chooses a set of probes (indices pointing to wires in the shared circuit) in the target circuit C, and a simulator wins the game if it successfully simulates the distribution of the tuple of variables carried by the corresponding wires without knowledge of the secret inputs.

Game 0 corresponds to the t-probing security definition: the adversary can choose t probes in a $t+1$-shared circuit, on whichever wires she wishes. In Game 1, the adversary is restricted to only probe gadget inputs: one probe on an addition or refresh gadget becomes one probe on one input share, one probe on a multiplication gadget becomes one probe on each of the input sharings. In Game 2, the circuit C is replaced by another circuit C' that has a multiplicative depth of one, through a transformation called **Flatten**, illustrated in [BGR18]. In a nutshell, each output of a multiplication or refresh gadget in the original circuit gives rise to a new input with a fresh sharing in C'. Finally, in Game 3, the adversary is only allowed to probe pairs of inputs of multiplication gadgets. The transition between these games is mainly made possible by an important property of the selected refresh and multiplication gadgets: in addition to being t-probing secure, they are t-*strong non interfering* (t-SNI for short) [BBD$^+$16]. Satisfying the latter means that t probed variables in their circuit description can be simulated with less than t_1 shares of each input, where $t_1 \leq t$ denotes the number of internal probes, i.e., which are not placed on output shares.

Game 3 can be interpreted as a linear algebra problem. In the flattened circuit, the inputs of multiplication gadgets are linear combinations of the circuit inputs. These can be modeled as Boolean vectors that we call *operand vectors*, with ones at indexes of involved inputs. From the definition of Game 3, the $2t$ probes made by the adversary all target these operand vectors for chosen shares. These probes can be distributed into $t+1$ matrices $M_0, \ldots, M_t$, where $t+1$ corresponds to the (tight) number of shares, such that for each probe targeting the share i of an operand vector $\mathbf{v}$, with i in $\{0, \ldots, t\}$, $\mathbf{v}$ is added as a row to matrix M_i. Deciding whether a circuit is t-probing secure can then be reduced to verifying whether $\langle M_0^T \rangle \cap \cdots \cap \langle M_t^T \rangle = \emptyset$ (where $\langle \cdot \rangle$ denotes the column space of a matrix). The latter can be solved algorithmically with the following high-level algorithm for a circuit with m multiplications:

For each operand vector $\mathbf{w}$,

1. Create a set $\mathcal{G}_1$ with all the multiplications for which $\mathbf{w}$ is one of the operand vectors.
2. Create a set $\mathcal{O}_1$ with the co-operand vectors of $\mathbf{w}$ in the multiplications in $\mathcal{G}_1$.
3. Stop if $\mathbf{w} \in \langle \mathcal{O}_1 \rangle$ ($\mathcal{O}_1$'s linear span), that is if $\mathbf{w}$ can be written as a linear combination of Boolean vectors from $\mathcal{O}_1$.
4. For i from 2 to m, create new sets $\mathcal{G}_i$ and $\mathcal{O}_i$ by adding to $\mathcal{G}_{i-1}$ multiplications that involve an operand $\mathbf{w}'$ verifying $\mathbf{w}' \in (\mathbf{w} \oplus \langle \mathcal{O}_{i-1} \rangle)$, and adding to $\mathcal{O}_{i-1}$ the other operand vectors of these multiplications. Stop whenever $i = m$ or $\mathcal{G}_i = \mathcal{G}_{i-1}$ or $\mathbf{w} \in \langle \mathcal{O}_i \rangle$.

If this algorithm stops when $\mathbf{w} \in \langle \mathcal{O}_i \rangle$ for some i, then there is a probing attack on $\mathbf{w}$, i.e., from a certain t, the attacker can recover information on $\mathbf{x} \cdot \mathbf{w}$ (where $\mathbf{x}$ denote the vector of plain inputs), with only t probes on the $(t+1)$-shared circuit. In the other two scenarios, the circuit is proven to be t-probing secure for any value of t.

3.2. Verification of implementations at a fixed order

So far, the verification of concrete implementations at a fixed order tolerated false negative. Namely, whenever a tuple could not be proven with the three rules introduced in [BBD+15], this tuple was considered as a potential flaw and a manual assessment was required. While in the chosen experiments, the number of false negative was very slow, depending on the implementation, it could become prohibitive to analyze manually a significant number of tuples.

Therefore, an additional automatic verification can be performed on potential flawed tuples. Namely, by computing the exact distribution of the tuple, it is possible to determine whether the latter depends on the secret inputs or not. This improvement was brought up in [BBC+19] and integrated into `maskVerif` by Inria. Basically, whenever verification fails, i.e. a potentially flawed tuple is detected, the joint distribution of this tuple is computed, so as to verify exactly whether this tuple is an attack for the weakest security notion considered. This step is exact, therefore all false negatives are removed.

While this step solves the issue of accuracy in the verification of concrete implementations, it is still complex and requires some improvement not to slow down the complexity in case many tuples cannot be verified by previous rules.

4. Optimization of the research complexity

In this section, we describe the progress of our new tools or updated versions of our tools in terms of research complexity.

4.1. Verification from the high-level description at a generic order

The verification of a high-level description at a generic order is usually more efficient than the verification of concrete implementations as it only goes through gadgets and not through every single instructions.

We display the timings obtained from the most recent tool built by the consortium in the context of the project: tightPROVE$^+$. Table 1 contains the results of tightPROVE$^+$ for some masking friendly primitives submitted to the lightweight competition organized by NIST, as well as for implementations of AES and PRESENT for comparison. We display the output of our algorithm for each circuit, along with the size of the registers used and the time it takes for tightPROVE$^+$ to output the results. Table 2 provides additional information about the implementations that are not secure in the register probing model. This includes the size of the registers, the time it takes to find the first attack, the time it takes to find all the operands that can be retrieved, then the least attack order, the optimal number of refresh gadgets needed to make the implementation secure in the register probing model, and finally the time tightPROVE$^+$ takes to verify that the refreshed implementation is indeed secure.

All calculations were made on an iMac with an intel Core i7 processor (4 GHz) and 16 GB of DDR3 RAM (1600 MHz), with parallel computing on its 8 CPUs.

Table 1: Results of tightPROVE$^+$ on all the implementations.

submissions	primitive	time (bitslice)	bit probing security	register size	time (n-slice)	register probing security
block ciphers						
GIFT-COFB, HYENA, SUNDAE-GIFT	GIFT-128	55 H 40 min	✓	32	2 H 15 min	✓
Pyjamask	Pyjamask-128	30 min	✓	32	6 min	✓
SKINNY, ROMULUS	SKINNY-128-256	10 H	✓	-	-	-
Spook	Clyde-128	10 min	✓	32	32 s	✗
permutations						
ACE	ACE	54 H 30 min	✓	32	10 min	✗
ASCON	p^{12}	1 H 45 min	✓	64	1 H 13 min	✓
Elephant	SPONGENT-π[160](1 round)	6 s	✓	-	-	-
Elephant	SPONGENT-π[160](10 rounds)	20 min 40 s	✓	-	-	-
Gimli	Gimli-36	22 H 45 min	✓	32	1 H 10 min	✗
ORANGE, PHOTON-BEETLE	PHOTON-256	2 H	✓	-	-	-
Xoodyak	Xoodoo[12]	2 H 50 min	✓	32	4 H 5 min	✓
others						
Subterranean	blank(8)	17 min	✓	-	-	-

Table 2: Complementary information on flawed implementations.

primitive	register size	first attack	all operands	least attack order	refresh gadgets needed	refreshed circuit
ACE	32	10 min	25 min	1	384	70 H
Clyde-128	32	32 s	2 min 10 s	2	6	3 min 10 s
Gimli-36	32	1 H 10 min	66 H 20 min	2	≤ 120	8 H 50 min

tightPROVE$^+$ places refresh gadgets for the considered implementations of ACE, Clyde and Gimli. For the two first primitives, there is exactly one subcircuit which is responsible for the identified register probing attacks, which can be fixed by adding only one refresh gadget. This gives us a lower bound for the optimal number of refresh gadgets, and since tightPROVE$^+$ does not find any further attack after the addition of refresh gadgets, it is also an upper bound. Gimli, however, is made of 6 subsequent identical subcircuits that are subject to register probing attacks, but the method uses 20 refresh gadgets per subcircuits to make the implementation secure. We can thus only conclude that we have an upper bound of 120 for the optimal number of gadgets, and that it is a multiple of 6, but in the current method, we cannot ascertain that it is optimal without setting up an exhaustive search.

4.2. Verification of implementations at a fixed order

To verify that an observation set O (a tuple of expressions) is independent from some secret, the key idea is to apply successive transformation on O into O', preserving its distribution, until a termination condition Test is able to syntactically decide the independence. The Test function depends on the property:

- For probing security, we check if the tuple is independent from the initial mapping by checking syntactically if the secret inputs do not appear in O'.
- For non-interference, we check if for each input parameter a, at most t shares $a[i]$ occur in the tuple O'.
- For strong non-interference, the condition is similar: for each parameter a, at most $|O_{in}|$ shares $a[i]$ should occur in O'.

The transformation of O is based on optimistic sampling rule: if $r \notin e$ then r and $e + r$ follow the same distribution, as well as O and O' where r is replaced by $e + r$ ($O\{r \leftarrow e + r\}$). The condition $r \notin e$ (i.e r is not a variable of e) ensures that the distributions of r and e are independent. The critical step is to select a substitution that will guarantee that the method terminates. Take for example $O = (r, x + r)$. If we replace r by $x + r$, we obtain after simplification $(x + r, r)$ on which we could apply the same transformation again and again.

4.2.1. Verification of single observation set

The Check verification algorithm is summarized in Figure 2: it takes as input an observation set represented as a tuple O of expressions. If Test is satisfied then Check succeeds. Otherwise, it uses the Select procedure to perform a transformation of O into O'. To guarantee termination, the algorithm first attempts to check if O can be rewritten (modulo associativity and commutativity of $+$)

as $C[e+r]$ where $C[\cdot]$ is a context, and $r \notin e \cup C$, i.e. r does not occur in e and C). If it is the case, we apply the optimistic sampling rule and get $C[e+r]\{r \leftarrow e+r\} = C[e+(e+r)] = C[r]$. Notice that in that case the size of $C[r]$ is less than the size of O (i.e the size of the resulting O' decreases).

If the algorithm cannot exhibit such a context, it tries to apply the general optimistic sampling rule (removing the condition $r \notin C$). The resulting expression is the simplification of $O\{r \leftarrow e+r\}$. For the simplification, we basically use the ring laws but the distributivity makes harder the detection of new simplifications. Notice that this time the size of the resulting $O' = O\{r \leftarrow e+r\}$ does not necessarily decrease. To ensure termination, we add a set R of random variables on which the general rule has already been used. The application of the rule is conditioned by the fact that $r \notin R$. The termination of the Check algorithm is ensured since either R increases or the size of O decreases (lexicographic order).

When more than one r allow to apply the rules (i.e for the selection of the context), we define the multiplicative depth of a random variable and we rewrite in increasing order of multiplicative depth. For instance, in the expression $r + (r' + e) \times e'$ we assign multiplicative depth 0 to r and 1 to r'.

We can prove that our new algorithm always terminates and is sound, i.e. it can detect all the attacks in our models. Note that considering only the first rule (first `if` statement of Select) makes our algorithm equivalent to the one of [BBD+15]. When we apply both rules (the two `if` statements of Select), our algorithm is equivalent to the one of [BDK+10], inspired from Gaussian elimination: contrary to this last one, we do not require the expressions to be linear. An additional advantage is the absence of false negatives when all the expressions are linear (completeness), it is no more the case if we remove the second `if` in Select.

Both algorithms return the list B of optimistic sampling rules that have been applied: successive transformations in the exploration algorithm can be replayed.

4.2.2. Exploration

The exploration algorithm ensures that the verification algorithm analyzes all the possible sets of at most t intermediate variables. However, rather than verifying each set separately, the exploration algorithm recursively checks larger sets, as in [BBD+15]. The idea behind the exploration algorithm is that if a tuple O is probabilistic non-interferent then all sub-tuples of O are. We present a very high level description of the algorithm to highlight the main differences with [BBD+15].

The algorithm CheckAll is presented in figure 2. Let X be the set of all tuples that need to be checked. If X is empty all tuples are trivially checked and the algorithm returns true. Else, it first tries to simplify as much as possible the set X by applying the simple optimistic sampling rule, as in the first `if` of Select. This point is really crucial because it allows to share simplifications between all tuples in X and was not done in [BBD+15]. Then, the algorithm chooses an element O in X and tries to check it. If O is successfully verified, the result B is a list of optimistic sampling transformations that can be applied to prove independence of O. Next, the algorithm selects all the elements of X that can be checked using the transformation B using Extend[1]. At this point all elements in X_0 are known to satisfy the desired properties. Finally the algorithm needs to check the remaining tuples $X \setminus X_0$.

Initially, X represents the set of all tuples of t elements that can be generated within the set of m possible observations. Its size is $\binom{m}{t}$. A naive implementation would thus be exponential in

[1] Missing tuples with Extend does not impact the correctness of the algorithm.

Verification algorithm

proc Select(R, O) =
  if $\exists r, e, C \mid O = C[e + r] \wedge r \notin e \cup C$ **then**
    return $(R, (e, r), C[r])$;
  if $\exists r, e, C \mid O = C[e + r] \wedge r \notin e \cup R$ **then**
    O' = Simplify($O\{r \leftarrow e + r\}$);
    return $(R \cup \{r\}, (e, r), O')$;
  else fail ;

proc Check(R, B, O) =
  if Test(O) **then return** B;
  (R', b, O') = Select(R, O);
  Check($R', B :: b, O'$);

Exploration algorithm

proc Replay(B, O) =
  if $B = []$ **then return** Test(Simplify(O))
  if $B = (e, r) :: B'$ **then**
    Replay($B', O\{r \leftarrow e + r\}$)

proc Extend(B, X) =
  $\{O \in X \mid$
    Replay$(B, O)\}$

proc OptSampling(X) =
  if $\exists r, e, C_X \mid X = C_X[e + r] \wedge r \notin e \cup C_X$ **then**
    OptSampling($C_X[r]$);
  else return X;

proc CheckAll(X) =
  if $X = \emptyset$ **return** true;
  X = OptSampling(X);
  O = Choose(X);
  B = Check($\emptyset, [], O$);
  X_0 = Extend(B, X);
  CheckAll($X \setminus X_0$);

Figure 2: Verification algorithm for probing security

t and it is crucial to have efficient data structures to represent X and to implement the functions OptSampling, Extend, and $X \setminus X_0$. We use the data structures presented in [BBD+15] (worklist base space splitting).

Moreover, we use a representation of expressions as imperative graphs. This allows to detect easily if the simplest version of optimistic sampling rule can be applied (used in the first conditional of Select and OptSampling), and to compute efficiently the resulting expression.

5. Verification in extended models

5.1. Probing model with physical defaults

The abstractions in the tools described above still do not prevent the risks due to specific physical defaults that may happen when trying to implement masking in hardware or software devices. In fact, many masking schemes that are secure in the abstract probing model become insecure (or at least less secure) when concretely implemented.

This is usually due to physical defaults which contradict the independence assumption required for secure masking. For instance, glitches are a form of physical default occurring when the information does not propagate simultaneously throughout execution. They introduce dependencies between the leakage of an instruction and of its predecessors (in the sense of dataflow analysis). These dependencies may cause hardware implementations proved secure in the probing model to be practically vulnerable against side-channel attacks [MPG05]. *Transitions* are another example

of physical default which more typically happen in software implementations, where the value in a register is overwritten by another value and leads the leakages to depend on both [CGP+12].

As a consequence, it is necessary to develop models and verification methods for proving security in presence of physical defaults. Bloem et al. [BGI+18] and Faust et al. [FGP+18] independently extended the probing model in order to capture physical defaults such as glitches — we will next denote this model as the probing model with glitches. In addition, Bloem et al. proposed an automated method based on an estimation of Fourier coefficients for proving that an implementation is secure in their model. They also used their verification method on a set of examples, including the S-Boxes of the AES and Keccak. Due to the computational cost of their approach, the tool primarily applies to the first-order setting, where secrets are split into two shares. Moreover, their method does not consider strong non-interference, which is key to verify complete implementations. By contrast, Faust et al. provided a hand-made analysis of some masking gadgets and proved their strong non-interference with glitches for arbitrary number of shares (at the cost of higher randomness requirements), and discuss simple conditions under which the composability rules from [BBD+16] apply to implementations with glitches.

From this basis, members of the consortium with co-authors from other institutes implemented an efficient method for reasoning about security of higher-order masked implementations in presence of physical defaults. This method follows a language-based approach based on probabilistic information flow for proving security of masked implementations, and so provides further evidence of the benefits of language-based approaches.

As in [BBD+15], this method, as an update of `maskVerif`, follows a divide-and-conquer approach, embodied in two algorithms. The first algorithm checks if leakages are independent of secrets for a fixed admissible set of observations. The algorithm repeatedly applies semantic-preserving simplifications to the symbolic representation of the leakages, until it does not depend on secrets or it fails. In addition to being sound and complete, the new algorithm minimizes false negatives for non-linear cases. The second algorithm explores all admissible observation sets, calling the first algorithm on each of them. This algorithm is carefully designed to minimize the number of sets to explore, using the idea of large sets from [BBD+15]. In addition, both algorithms are specifically tailored to a rich programming model, which is introduced to maximize applicability. The critical feature of this new programming model is that all instructions are annotated with a symbolic representation of leakage. The programming model neatly subsumes several models from the literature, including the probing model, the recently proposed probing model with glitches, and a version of the probing model with transitions. Moreover, the resulting tool applies to three main security notions: probing security, threshold non-interference, and strong non-interference (which is essential for compositional reasoning).

5.2. Register probing model

Although nicely answering a relevant open issue, the developed tool `tightPROVE` to verify the exact probing security of standard circuits at any order still suffers an important limitation. It only applies to Boolean circuits and does not straightforwardly generalize to software implementation processing ℓ-bit registers (for $\ell > 1$).

Therefore, following the work on `tightPROVE` , Sorbonne Université and CryptoExperts proposed an extended verification tool, that we shall call tightPROVE$^+$. This tool can verify the security of any masked bitslice implementation in the register probing model which makes more sense that the bit probing model w.r.t. masked software implementations. The latter assumes that an adversary can

observe a full register at the cost of a single observation and is thus not restricted anymore to the (bit)-probing model. Given a masked bitslice/n-slice implementation composed of standard gadgets for bitwise operations, tightPROVE$^+$ either produces a probing-security proof or exhibits a probing attack.

One step further, the authors provided a more global tool referred to as `tornado`. From the high-level description of an implementation, this tool outputs an efficient Assembly masked implementation of the same functionality with proven security in the register probing model at any chosen order, using tightPROVE$^+$. This contribution will be available in the proceedings of Eurocrypt 2020.

5.3. Random probing model

All previous works provide verification methods in variants of the probing security model. However, the *noisy leakage model* originally introduced by Chari et al. in 1999 [CJRR99] and later extended by Prouff and Rivain in [PR13] as a specialization of the *only computation leaks* model [MR04] better captures the reality of embedded devices. 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 continue to be verified in the probing model relying on the *not tight* reduction that was formally established by Duc, Dziembowki, and Faust [DDF14].

The latter reduction actually comes with an intermediate leakage model, called *random probing model*, to which the security in the noisy leakage model reduces to. Intuitively 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 might not be constant which is not satisfactory from a practical viewpoint (the side-channel noise might not be customizable).

In a work in progress, ANSSI, Université du Luxembourg, and CryptoExperts aim to provide a framework to verify, compose, and build random probing secure circuits from simple gadgets. In particular, they define a verification method that they instantiate in a tool to automatically exhibit the security parameters in the random probing model 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 $f(p)$. From these notations, the new tool, based on top of a set of rules that were previously defined to verify the probing security of implementations in `maskVerif` [BBD$^+$15], takes as input the description of a circuit and outputs an upper bound on the failure probability function f. While it is limited to small circuits by complexity, the state-of-the-art shows that verifying those circuits can be particularly useful in practice (see e.g. the `maskVerif` tool [BBD$^+$15]), for instance to verify gadgets and then deduce global security through composition properties and/or low-order masked implementations.

Références bibliographiques

[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, Heidelberg, 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, Heidelberg, April 2015.

[BBD+16] 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.

[BDK+10] Gilles Barthe, Marion Daubignard, Bruce M. Kapron, Yassine Lakhnech, and Vincent Laporte. On the equality of probabilistic terms. In Edmund M. Clarke and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 16th International Conference, LPAR-16, Dakar, Senegal, April 25-May 1, 2010, Revised Selected Papers*, volume 6355 of *Lecture Notes in Computer Science*, pages 46–63. Springer, 2010.

[BGI+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, Heidelberg, April / May 2018.

[BGR18] 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, Heidelberg, December 2018.

[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*, volume 7275 of *LNCS*, pages 69–81. Springer, Heidelberg, May 2012.

[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, Heidelberg, August 1999.

[Cor18] Jean-Sébastien Coron. Formal verification of side-channel countermeasures via elementary circuit transformations. In Bart Preneel and Frederik Vercauteren, editors, *ACNS 18*, volume 10892 of *LNCS*, pages 65–82. Springer, Heidelberg, July 2018.

[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, Heidelberg, May 2014.

[FGP+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. https://tches.iacr.org/index.php/TCHES/article/view/7270.

[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, Heidelberg, August 2003.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In Alfred Menezes, editor, *CT-RSA 2005*, volume 3376 of *LNCS*, pages 351–365. Springer, Heidelberg, February 2005.

[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, Heidelberg, February 2004.

[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, Heidelberg, May 2013.