SYSTEMATIC
PARIS REGION SYSTEMS & ICT CLUSTER

VERISICC
Deliverable 1.1
State of the Art of formal Verification Techniques

APPEL A PROJET: FUI25

ACRONYME DU PROJET: VERISICC

NOM DU PROJET: VERISICC

Leader of this deliverable

- Partner: University of Luxembourg
- Nom du contact: Jean-Sébastien Coron
- Contact information: jean-sebastien.coron@uni.lu

Leader of the project

- Company: CryptoExperts
- Contact name: Sonia Belaïd
- Contact information: sonia.belaid@cryptoexperts.com, 06 68 75 30 66

Partners

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

Version : July 2019

Sommaire

1. Introduction

1.1. The masking countermeasure

Masking is the most widely used countermeasure against side-channel attacks for block-ciphers and symmetric-key algorithms. In a first-order countermeasure, all intermediate variables x are masked into a pair (x', r) where r is a randomly generated value and $x' = x \oplus r$. For such countermeasure, it is usually straightforward to verify its security against first-order attacks; namely it suffices to check that all intermediate variables have the uniform distribution, or at least that their distribution is independent from the key; therefore an attacker processing the side-channel leakage of intermediate variables separately (as in a first-order attack) does not get useful information.

However second-order attacks combining the leakage on x' and r can be mounted in practice, so it makes sense to design masking algorithms resisting higher-order attacks. This is done by extending Boolean masking to n shares with $x = x_1 \oplus \cdots \oplus x_n$; in that case an implementation should be resistant against t-th order attacks, in which the adversary combines leakage information from at most $t < n$ intermediate variables.

1.2. Security proofs

In principle any countermeasure against high-order attacks should have a security proof, but such proof can be either missing, incomplete, or incorrect.

The first step is to specify what it means for a masking countermeasure to be secure, *i.e.* what is the security model. Such formalization was initiated by Ishai, Sahai and Wagner in [ISW03]. In this model, the adversary can probe at most t wires in the circuit, but he should not learn anything about the secret key. The approach for proving security is based on simulation: one must show that any set of t wires probed by the adversary can be perfectly simulated without the knowledge of the secret-key. This shows that the t probes do not bring any useful information to the attacker, since he could run this simulation by himself.

More precisely, the simulation technique consists in showing that any set of t probes can be perfectly simulated by the knowledge of only a proper subset of the input shares x_i. At the beginning of the algorithm an original variable x is shared into n shares x_i. When x is part of the secret-key, this pre-sharing cannot be probed by the adversary. Since any subset of at most $n-1$ input shares x_i are uniformly and independently distributed, the simulation of the probed variables can be performed without knowing the secret-key.

The main result in [ISW03] is to show that any circuit C can be transformed into a new circuit C' of size $\mathcal{O}(t^2 \cdot |C|)$ that is resistant against an adversary probing at most t wires in the circuit. The construction is based on secret-sharing every variable x into n shares with $x = x_1 \oplus \cdots \oplus x_n$, and processing the shares in a way that prevents a t-limited adversary from learning any information about the initial variable x, using $n \geq 2t + 1$ shares.

One advantage of the ISW model is that its conceptual simplicity makes it amenable to formal verification. This has been demonstrated in a series of works, including [MOPT12, BRNI13, EWS14, BBD$^+$15, BBD$^+$16, Cor18, ZGSW18, BGI$^+$18, BIKM18].

The most immediate benefit of formal verification is its automation, allowing to deal with the combinatorial complexity of proving masked implementations secure. This complexity is specially significant for implementations where secrets are split into a large number of shares; we call such

implementations higher-order. Perhaps more importantly, formal verification has also been instrumental for advancing the state-of-the-art in masking. First, formal verification tools have been used to reduce the randomness cost of existing schemes. Second, strong non-interference, which solves a long-standing problem of compositional reasoning for masking, has first emerged in the context of formal verification, before being adopted in the literature on masking.

Another model is the noisy leakage model introduced by Chari et al. [CJRR99] then extended by Prouff and Rivain [PR13]. It describes many realistic side-channel attacks where an adversary obtains each intermediate value perturbed with a "δ-noisy" leakage function. A leakage function L is called δ-noisy if for a uniformly random variable Y we have $\mathrm{SD}(Y; Y|\boldsymbol{L}_Y) \leq \delta$, with SD the statistical distance. It was shown in [DFS15] that an equivalent condition is that the leakage is not too informative, where informativity is measured with the standard notion of mutual information $\mathrm{MI}(Y; \boldsymbol{L}_Y)$. In contrast with the ϵ-probing model, the adversary obtains noisy leakage for each intermediate variable. For example, in the context of masking, he obtains $\mathsf{L}(Y_i, \boldsymbol{R}_i)$ for all the shares Y_i, which is reflective of actual implementations where the adversary can potentially observe the leakage of all these shares, since they are all present in leakage traces.

A third and intermediate model is getting more and more important in the literature recently, the random probing model. Basically, it assumes that every wire or intermediate variable in a circuit leaks with some probability p. The circuit is then secure if the probability to leak intermediate variables which jointly depend on the secrets is negligible. This model benefits from being more realistic than the probing model since it captures a larger set of attacks. For instance, horizontal attacks that may exploit the multiple use of a variable to get information on its value is not captured in the probing model while it is handled in the random probing model where the probability to get the value is increased by the repetition.

Duc et al. showed that security against probing attacks implies random probing security for some probability which itself implies security against noisy leakages [DDF14]. This result leads to the natural strategy of proving security in the (simpler) probing model while stating security levels based on the concrete information leakage evaluations (as discussed in [DFS15]).

2. Formal verification tools of masking

A first sequence of work manages to provide verification of masked implementations, but is mainly restricted to first-order masking schemes. We give an overview below and then focus on recent tools that achieves verification of higher-order verification schemes.

2.1. Verification of first-order masking schemes

In 2012, Moss et al. [MOPT12] implement the first automated method to verify and build masked implementations. Basically, they design a type-based masking compiler that tracks variables that are masked by random values and iteratively modifies an unprotected program until all secrets are masked. This strategy achieves first-order probing security. Although efficient and scalable, this method is overly conservative and rejects secure programs.

Bayrak et al. [BRNI13] investigate in 2013 the logic-based verification which offers interesting trade-offs between efficiency and expressiveness. Their SMT-based method is able to analyze the probing security of first-order masked implementations by proving statistical independence between secrets and leakage. It was later extended by Eldib, Wang and Schaumont [EWS14] to achieve

higher-order masking verification with a method based on model counting. The authors use incremental verification in order to circumvent the resulting exponential blow-up. Despite these improvements, the scope of the method remains limited on concrete examples. Finally, Zhang et al. [ZGSW18] provide additional improvements in terms of precision and scalability with their tool SCInfer (partly inspired from `maskVerif` below).

2.2. Verification of higher-order masking schemes

In the following, we focus only on recent tools able to provide verifications of concrete masking schemes at higher orders.

2.2.1. `maskVerif`

In 2015, Barthe et al. exhibited an automated method to prove the security of masked implementation against t-th order attacks, for small values of t [BBD+15]. The method only works for small values of t because the number of possible t-tuples of intermediate variables grows exponentially with t. To formally prove the security of a masking algorithm, the authors describe an algorithm to construct a bijection between the observations of the adversary (corresponding to a t-tuple of intermediate variables) and a distribution that is syntactically independent from the secret inputs; this implies that the adversary learns nothing from this particular t-tuple of intermediate variables. All possible t-tuples of intermediates variables are then examined by exhaustive search.

The authors obtain a formal verification of various masked implementations, up to second order masked implementation of AES, and up to 5-th order for the masked Rivain-Prouff multiplication [RP10]. In particular, the authors were able to rediscover some known attacks and discover new ways of attacking already broken schemes.

The main drawback of this approach is that it can only work for small orders t and small programs, since the running time is exponential in t and the size of the program.

The tool allows to check various security properties, like probing security, non-interference (NI) and strong non-interference (SNI). It is also able to perform security analysis in presence or in absence of glitches from its extension in 2019, or with transition or without transition. It provides a small intermediate language for describing implementation but it can also take Verilog implementation as input.

2.2.2. `CheckMasks`

A simplification and extension of the formal verification results from [BBD+15] and [BBD+16] was described in [Cor18]. Two complementary approaches were described: a generic approach for the formal verification of any circuit, but for small attack orders only (as in [BBD+15]), and a specialized approach for the verification of specific circuits, but at any order (as in [BBD+16]).

For the generic verification of countermeasures at small orders, the author uses a different formal language from [BBD+15]. In particular the underlying circuit is represented as nested lists, which leads to a simple and concise implementation in Common Lisp, a programming language well suited to formal manipulations. The author is then able to formally verify the security of the Rivain-Prouff countermeasure with very few lines of code. The running times for formal verification are similar to those in [BBD+15]. Thanks to this simpler approach, the author also extends [BBD+15] to handle a combination of arithmetic and Boolean operations, and formally verifies the security of the recent

	Number of files	Lines of code
`maskVerif` [BBD+15]	13	4 678
CheckMasks	1	459

Table 1: Number of files and number of lines of code in the EasyCrypt-based tool from [BBD+15] and in our CheckMasks tool.

Boolean to arithmetic conversion algorithm from [Cor17b]. This approach is implemented in a tool, referred to as `CheckMasks`.

For the verification of specific gadgets at any order (instead of small orders only with the generic approach), our technique is quite different from [BBD+16] and consists in applying elementary transforms to the circuit, until the t-NI or t-SNI properties become straightforward to verify. For a set of well-chosen elementary transforms, the formal verification time becomes polynomial in t (instead of exponential with the generic approach); this implies that the formal verification can be performed at any order. The `CheckMasks` tool provides a formally verified proof of the t-SNI property of the multiplication algorithm in the Rivain-Prouff countermeasure, and of the mask refreshing based on the same multiplication algorithm; in both cases the running time of the formal verification is polynomial in the number of shares n.

Finally, the authors show how to get the best of both worlds, at least for simple circuits: they show how to automatically apply the circuit transforms that lead to a polynomial time verification, based on a limited set of generic rules. Namely we identify a set of three simple rules that enable to automatically prove the t-SNI property of the multiplication based mask refreshing, and also two security properties of mask refreshing considered in [Cor17b]. The source code of our `CheckMasks` verification tool is publicly available at [Cor17a], under the GPL v2.0 license.

Comparison of code size. In Table 1 we give the number of lines of code of `CheckMasks` tool and `maskVerif`. The source code from [BBD+15] is publicly available at [sou].

2.2.3. Bloem et al. hardware verification tool

Bloem et al. went one step further with an automated method to verify the probing security of a hardware circuit in the presence of *glitches* [BGI+18]. The latter are a kind of physical default occurring when the information does not propagate simultaneously throughout execution. Basically, the output of a computation may be unstable before all the information get to it. Such physical defaults may introduce dependencies between the leakage of an instruction and of its predecessors (in the sense of dataflow analysis) that are not yet captured in the probing model. Hardware implementations may thus be proved secure in the probing model and be practically vulnerable against side-channel attacks.

Independently, Bloem et al. and Faust et al. [FGP+18] thus extended the original probing model to capture glitches by assuming that any computation between two storage in a register would leak all its inputs. In the following, we will denote this extended model by *probing model with glitches*.

Based on this new model, Bloem et al. developed an automated tool taking a circuit in Verilog as input to verify its security in the probing model with glitches. Their method is based on an estimation of Fourier coefficients and benefits from being adapted to concrete hardware implementations. The

authors proved its application on a concrete set of examples. Nevertheless, their approach is mostly limited to the first-order setting where variables are split into two shares due to its computational cost.

3. Formal tools for masking generation

3.1. maskComp

Barthe *et al.* studied the composition property of masked algorithms. In particular, the authors introduce the notion of *strong simulatability*, a stronger property which requires that the number of input shares necessary to simulate the observations of the adversary in a given gadget is independent from the number of observations made on output wires. This ensures some separation between the input and the output wires: no matter how many output wires must be simulated (to ensure the composition of gadgets), the number of input wires that must be known to perform the simulation only depends on the number of internal probes within the gadget.

The paper [BBD+16] has a number of important contributions that we summarize below. Firstly, the authors introduce the t-NI and t-SNI definitions. The t-NI security notion corresponds to the original security definition in the ISW probing model [ISW03]; it requires that any $t_c \leq t$ probes of the gadget circuit can be simulated from at most t_c of its input shares. The stronger t-SNI notion corresponds to the strong simulatability property mentioned above, in which the number of input shares required for the simulation is upper bounded by the number of probes t_c in the circuit, and is independent from the number of output variables $|\mathcal{O}|$ that must be simulated (as long as the condition $t_c + |\mathcal{O}| < t$ is satisfied).

The authors show that the t-SNI definition allows for securely composing masked algorithms; *i.e.* for a construction involving many gadgets, one can prove that the full construction is t-SNI secure, based on the t-SNI security of its components. The advantages are twofold: firstly the proof becomes modular and much easier to describe. Secondly as opposed to [ISW03] the masking order does not need to be doubled throughout the circuit, as one can work with $n \geq t + 1$ shares, instead of $n \geq 2t + 1$ shares. Since most gadgets have complexity $\mathcal{O}(n^2)$, this usually gives a factor 4 improvement in efficiency. In [BBD+16], the authors prove the t-SNI property of several useful gadgets: the multiplication of Rivain-Prouff [RP10], the mask refreshing based on the same multiplication algorithm, and the multiplication between linearly dependent inputs from [CPRR13].

Moreover, in [BBD+16] the authors also machine-checked the multiplication of Rivain-Prouff and the multiplication-based mask refreshing in the EasyCrypt framework [BDG+14]. The main point is that their machine verification works for any order, whereas in [BBD+15] the formal verification could only be performed at small orders t. However, the approach seems difficult to understand (at least for a non-expert in formal methods), and when reading [BBD+16] it is far from obvious how the automated verification of the countermeasure can be implemented concretely; this seems to require a deep knowledge of the EasyCrypt framework.

Finally, the authors built an automated approach for verifying that an algorithm constructed by composing provably secure gadgets is itself secure. They also implemented an algorithm for transforming an input program P into a program P' secure at order t; their algorithm automatically inserts mask refreshing gadgets whenever required.

3.2. tightPROVE

In the same line of work, Belaïd, Goudarzi, and Rivain recently proposed tightPROVE [BGR18] which exactly and directly verifies the software probing security of a circuit based on standard gadgets at any order.

The tightPROVE verification tool aims to verify the probing security of a shared Boolean circuit. More specifically, it takes as input a file containing 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: Game 0 to Game 3. In each of them, an adversary $\mathcal{A}$ chooses a set of probes $\mathcal{P}$ (indices pointing to wires in the shared circuit) in the target circuit C, and a simulator $\mathcal{S}$ 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 inputs sharings. In Game 2, the circuit C is replaced by another circuit C' that is functionally equivalent and has a multiplicative depth of one, through a transformation called Flatten. In a nutshell, each output of a multiplication or refresh gadget is considered as a new input with a fresh sharing. 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* [BBD+16]. As detailed in the previous section, 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 denotes the number of probes that are not outputs.

The last game can be interpreted as a linear algebra problem. In the flattened circuit, the inputs of multiplication gadgets are linear combinations of the circuit inputs, which can be seen as boolean vectors that we call *operand vectors*, with ones at indexes of involved inputs. From the definition of the last game, the $2t$ probes made by the adversary all target these operand vectors for chosen shares. These probes can be distributed into $t+1$ matrices $M_0, M_1, \ldots, M_t$, where $t+1$ correspond to the (tight) number of shares, such that for each probe targeting the share i of an operand vector $\mathbf{v}$, with i in $\{0, 1, \ldots, t\}$, $\mathbf{v}$ is added as a row to matrix M_i. The problem of whether a circuit is t-probing secure can then be reduced to verifying whether $Im(M_0) \cap \cdots \cap \langle() M_t) \neq \emptyset$. 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 operand vectors of the multiplications in $\mathcal{G}_1$ but $\mathbf{w}$.
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 $\mathbf{w}$, meaning that from a certain t, the attacker can recover the actual value w corresponding to the sharing $\mathbf{w}$, with only t probes on the corresponding $(t+1)$-shared circuit. In the other two scenarios, the circuit is proven to be t-probing secure for any value of t.

Note that while `maskComp` and `tightPROVE` are recalled in the generation tools category, they can also be used to verify the security of masked implementations. In this scenario, `tightPROVE` takes as inputs standard circuits made of sharewise additions, multiplications gadgets from [ISW03] and SNI refreshing gadgets.

Références bibliographiques

[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.

[BDG+14] Gilles Barthe, François Dupressoir, Benjamin Grégoire, César Kunz, Benedikt Schmidt, and Pierre-Yves Strub. *EasyCrypt: A Tutorial*, pages 146–166. Springer International Publishing, Cham, 2014.

[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.

[BIKM18] Roderick Bloem, Rinat Iusupov, Martin Krenn, and Stefan Mangard. Sharing independence & relabeling: Efficient formal verification of higher-order masking. Cryptology ePrint Archive, Report 2018/1031, 2018. https://eprint.iacr.org/2018/1031.

[BRNI13] 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, Heidelberg, August 2013.

[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.

[Cor17a] Jean-Sébastien Coron. CheckMasks: formal verification of side-channel countermeasures, 2017. Publicly available at https://github.com/coron/checkmasks.

[Cor17b] Jean-Sébastien Coron. High-order conversion from boolean to arithmetic masking. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 93–114, 2017.

[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.

[CPRR13] Jean-Sébastien Coron, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Higher-order side channel security and mask refreshing. In *Fast Software Encryption - 20th International Workshop, FSE 2013, Singapore, March 11-13, 2013. Revised Selected Papers*, pages 410–424, 2013.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *EUROCRYPT 2014*, volume 8441 of *LNCS*, pages 423–440. Springer, 2014.

[DFS15] Alexandre Duc, Sebastian Faust, and François-Xavier Standaert. Making masking security proofs concrete - or how to evaluate the security of any leaking device. In *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 401–429. Springer, 2015.

[EWS14] 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.

[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.

[MOPT12] 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, Heidelberg, September 2012.

[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.

[RP10] 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.

[sou] maskVerif. Publicly available at https://gitlab.com/benjgregoire/maskverif.

[ZGSW18] Jun Zhang, Pengfei Gao, Fu Song, and Chao Wang. Scinfer: Refinement-based verification of software countermeasures against side-channel attacks. In *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*, pages 157–177, 2018.