ADD-based Spectral Analysis of Probing Security

Maria Chiara Molteni
Department of Computer Science, Università degli Studi di Milano, Italy
maria.molteni@unimi.it

Vittorio Zaccaria
Department of Electronics, Information and Bioengineering Politecnico di Milano, Italy
vittorio.zaccaria@polimi.it

Valentina Ciriani
Department of Computer Science, Università degli Studi di Milano, Italy
valentina.ciriani@unimi.it

***Abstract*—In this paper, we introduce a novel exact verification methodology for non-interference properties of cryptographic circuits. The methodology exploits the Algebraic Decision Diagram representation of the Walsh spectrum to overcome the potential slow down associated with its exact verification against non-interference constraints. Benchmarked against a standard set of use cases, the methodology speeds-up 1.88x the median verification time over the existing state-of-the art tools for exact verification.**

***Index Terms*—Probing security, non-interference, Walsh spectrum**

I. Introduction

Probing a circuit, i.e., measuring the power consumption or EM emissions from a subset of nodes of a circuit, is a useful technique through which an attacker can derive information correlated with the secret manipulated by a cryptographic circuit. A circuit is d-probing secure if, given d probes, it is impossible to derive information about the secret values encoded in the masks/shares. *Probing security* is the branch of research that tries to devise models, tools and countermeasures against this type of attacks [1]. Proving the security for small circuits (*gadgets*) requires typically a small effort, but reasoning about their composition is still not trivial. In fact, one of the main problems addressed is the *composability* of security properties, i.e., determining, given two probing secure gadgets f, g, if their functional composition $g \circ f$ is probing secure itself.

Preserving security across composition might be ensured through *refreshing*, i.e., keeping the secret's shares into a uniformly random state [2] and on the *strong-non-interference* properties [3] of the circuit itself. The latter ensures that the probabilistic distribution of the probed values does not depend on all of the secret's shares, but it varies only with the number of internal probes [3]. Putting non-interference to work in proof mechanization has been the goal of several works in the recent past [3]–[5] while later developments concerned the realistic application to circuits that might leak even through transient glitches on the circuit's internal nodes [6], [7].

In this paper, we address the problem of tooling needed for the verification of non-interference properties. To contextualize our work, note that existing *heuristic* tools such as `maskVerif` [8] can be helpful in verifying if a fixed configuration instance of a gadget is d-probing secure or d strong-non-interferent (d-SNI). Notwithstanding the efficiency of `maskVerif`, its developers argue that *more precise approaches remain important, when verification with more efficient methods fail* [8]. Therefore, the importance of studying exact techniques is quite evident. A few other approaches have been proposed in the past to address this verification problem through some kind of approximation [9], [10], while existing exact approaches either suffer from size and exponential time complexity [11] or have not been tested on $d > 3$ [12].

In this paper, we introduce an Algebraic Decision Diagram (ADD)-based [13] methodology for the exact validation of circuits against required strong non-interference properties. To our knowledge, the methodology is faster than the state of the art exact methods proposed in the literature and builds on decades of work on BDD/ADD libraries. Benchmarked against a standard set of use cases (taken from the `maskVerif` [8] repository), we show that the proposed exact tool is able to compete with heuristic methods as well.

The organization of this paper has the following structure. Section II presents the theoretical background and the state of the art of the problem at hand while Section III presents the proposed methodology. Finally, Section IV compares our approach with existing methods (exact and heuristic) while Section V concludes by highlighting the future work.

II. Background and state of the art

Let us consider a generic vector Boolean function:

$$\phi(x_1, \ldots, x_n) : \mathbb{F}_2^n \rightarrow \mathbb{F}_2^m$$

where some of the values x_i are *sensitive* (i.e., they have been computed using a secret). A side-channel attack consists of measuring the power consumption of internal nodes of the circuit (e.g., through *probes*) and recovering, through some kind of correlation analysis, some or all the sensitive values that could have produced such measure.

Mitigation against a side-channel attack are designed by splitting each sensitive value x_i into d values $a_i = \{a_{i,j}\}_{j \in 1 \ldots d}$ such that $\sum_j a_{i,j} = x_i$; these d values are called *shares*. In principle, this is done by using $d - 1$ auxiliary random values (aka *masks*) and, all d shares $a_{i,j}$ are obtained, the correlation of each share with the sensitive value x_i is null [1]. The implementation of ϕ must convert in order to produce the output with a set of shares much like the original sensitive values. The computation of each output ϕ_i is thus split into a set of d vector functions $f_i = \{f_{i,j}\}_{j \in 1 \ldots d}$ such that

$$\phi(x_1, \ldots, x_n) = \sum_j f_j(A_1, \ldots, A_n), A_i \subseteq \{a_{i,1}, \ldots a_{i,d}\}$$

where each f_j is called an *output share* of ϕ and it must be impossible to derive the value of ϕ unless one obtains all d output shares.

In the *probing-security* attack model, aside from regular output shares f_i, attackers can observe a group of the internal probed values of the circuit; we model these as additional outputs $P = \{p_1, \ldots, p_{|P|}\}$ where each p_i is a function of the input shares. A mitigation against a probing attack ensures that none of the p_i are correlated with the original sensitive values. To design such countermeasures designers introduce randomness, i.e., an additional group of inputs $R = \{r_1 \ldots r_{|R|}\}$ which are uniformly random summed to the internally computed values of the function so as to make each p and f not correlated with the sensitive values. The resulting correlation between each f and p with any a and r is critical to determine whether the circuit is probing secure. It has been shown that this data is derivable exactly from the Walsh matrix of the combined vectorial function of each f and p [14], [15].

Given a vectorial Boolean function $f : \mathbb{F}_2^n \to \mathbb{F}_2^m$, we define its Walsh transform as a $2^m \times 2^n$ matrix $\widehat{f}$ whose elements are:

$$\widehat{f}_{\omega,\alpha} = \sum_{x \in \mathbb{F}_2^n} (-1)^{\omega^\intercal f(x) \oplus \alpha^\intercal x} \tag{1}$$

where $\omega \in \mathbb{F}_2^m, \alpha \in \mathbb{F}_2^n$ being the binary encoding of the row and column indices, called *spectral coordinates* (or sometimes *masks*).

These matrices encode the correlation information between input variables' XOR-combinations and the corresponding output ones, which can be readily computed [16]: $W_f = 2^{-n}\widehat{f}$. Most importantly, a function is d-probing secure if its correlation matrix presents null values in correspondence of any sensitive value and the combination of up to d probes and regular outputs.

A. Composition

When functions f, g at hand fall into certain classes, it is also possible to reason about the d-probing security of their functional composition $g \circ f$.

A function f is d-non interferent (abbreviated as d-NI) if, when given a total of s outputs and internal probes, $s \leq d$ implies a dependency with maximum s input shares. A function f is strongly d-non interferent (d-SNI) if $s \leq d$ implies a dependency with maximum i input shares, where i is the number of internal probes, among those placed [4]. In general, a d-NI or d-SNI is a d-probing secure function but not vice versa. Besides d-NI and d-SNI do not always compose into a d-NI function but it is possible to show that, if f is d-SNI and g is d-NI (d-SNI) then the composition is d-NI (d-SNI).

1) Example of composition: Figure 5 shows the structure of the composition of two functions f and g where f is d-NI and g is d-SNI; f refreshes its input a with two random bits r_f:

$$o_f(a_0, a_1, a_2, r_0, r_1) = [a_0 \oplus r_0 \oplus r_1, a_1 \oplus r_0, a_2 \oplus r_1]$$

and has a probe at location $p_f = a_0 \oplus r_0$. On the other hand, $g(a, b, r_g)$ is the ISW multiplication [1] consuming 3 random bits r_g to refresh its outputs and has a single probe $p_g = a_2 \wedge b_1$.

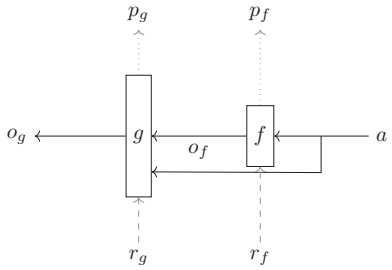

Fig. 1: The composition pattern of f (d-NI) and g (d-SNI) derived from [2].

			0	0	0	0	0	0	0	0	0	0	$\ldots \rho_g$
			0	0	0	0	1	1	1	1	2	2	$\ldots \rho_f$
			0	1	2	3	0	1	2	3	0	1	$\ldots \alpha$
π_f	π_g	ω_g											
0	0	0	1										
0	0	1											
0	0	2											
0	0	3	1			1							
0	1	0	1	1			1	1	1		1	1	
0	1	1											
0	1	2											
0	1	3	1	1	1	1	1	1	1	1	1	1	
1	0	0						1					
1	0	1											
1	0	2											
1	0	3						1	1				
1	1	0		1	1	①	1	1	1			1	
1	1	1											
1	1	2											
1	1	3	1	1	1	1	1	1	1	1	1	1	

Fig. 2: Compact representation of the correlation matrix of the example (greek letters indicate the spectral coordinate associated with each variable, i.e., α is the spectral coordinate associated with variable a and so on). White areas indicate where the composition $g \bullet f$ must have zero values to meet d-NI hypotheses. Circled black 1 indicates the witness of this construction being not d-NI.

Figure 2 shows the compact representation of a correlation matrix of the composition ($g \circ f$) with the specified two probes where a 1 indicates a non-null correlation between a specific combination of outputs and inputs, zero otherwise (for a definition of compact representation of correlation matrix see [11]). We note that there is a non-null correlation between row $[1, 1, 0]$ and column $[0, 0, 3]$ which indicates that one needs only two probed values to get three shares; h is thus not 2-NI.

B. Tooling

There are a number of automated approaches for verifying security of masked implementations. `maskVerif` [8] is an approach based on probabilistic information flow for proving security of shared (or *masked*) implementations. It applies semantic-preserving simplifications to the symbolic representation of the leakages, until it does not depend on secrets or it fails. The algorithm is sound and complete for linear systems

but does not completely eliminate false negatives for non-linear circuits. In [9] Bloem et Al. propose an approximated method based on Fourier coefficients. Their tool primarily applies to the first-order circuits and does not consider strong non-interference. An attempt to exactly model the joint probability distributions through ROBDDs has been proposed in [12] with the tool named SILVER, but it is not clear how much this approach scales for $d > 3$. Authors of [11] try to overcome this limit by arguing for an exact automated approach based on Walsh transforms; in particular they propose an approach to represent Walsh matrices as list of lists which, however, gives exponentially higher analysis times. In this paper, we show that Binary Decision Diagrams (BDDs) [17] and Algebraic Decision Diagrams [13] can be used to improve the performance issues of exact, correlation-based approaches and provide a natural way to express the queries related to non-interference.

C. Binary Decision Diagrams

In this section we briefly review two data structures based on decision diagrams. We first give some definitions and properties of standard Binary Decision Diagrams (BDDs) and then we describe their generalization to Algebraic Decision Diagrams (ADDs).

A *Binary Decision Diagram* (BDD) [17] on a set of Boolean variables $\{x_1, \ldots, x_n\}$ is a rooted, connected direct acyclic graph, where each internal node N is labeled by a Boolean variable x_i and it has two outgoing edges, the 0-edge and the 1-edge, pointing to two nodes, i.e., the 0-child and the 1-child of node N, respectively. Terminal nodes (or leaves) are labeled with a constant value 0 or 1. Usually, binary decision diagrams are exploited to represent Boolean functions.

A BDD is *ordered* (OBDD) if there exists a total order $<$ over the set of variables such that if an internal node is labeled by x_i, and its 0-child and 1-child have labels x_{i_0} and x_{i_1}, respectively, then $x_i < x_{i_0}$ and $x_i < x_{i_1}$. The choice of the variable order can have a dramatic impact on the size of the BDD. A OBDD is *reduced* if there exist no nodes whose 1-child is equal to the 0-child and there do not exist two distinct nodes that are roots of isomorphic subgraphs. A reduced and ordered BDD is called *ROBDD*. Note that, usually, the term BDD is used instead of the correct term ROBDD.

Many operations on Boolean functions can be efficiently implemented by ROBDD's manipulations. For example Boolean operations (AND, OR, EXOR, etc.) between two ROBDDs g_1 and g_2 have complexity $O(|g_1| \cdot |g_2|)$, and the negation of a function f has constant complexity $O(1)$.

Note that the representation of Boolean functions with ROBDDs allows to perform operations that do not depend on the number of inputs that are equal to 1 or 0; for this reason, algorithms based on ROBDDs are usually defined implicit algorithms. For functions with logical structure, a BDD representation can be exponentially smaller than the explicit one.

An *Algebraic Decision Diagram* (ADD) [13] can be described as a BDD with a generalized set of constant values. Therefore, an ADDs is the representation of a function $f : \{0,1\}^n \to S$, where S is an arbitrary set. When S is $\{0,1\}$ the ADD is a classical BDD. Due to the implicit nature of BDDs and ADDs, in this paper we exploit these data structures for representing sparse matrices.

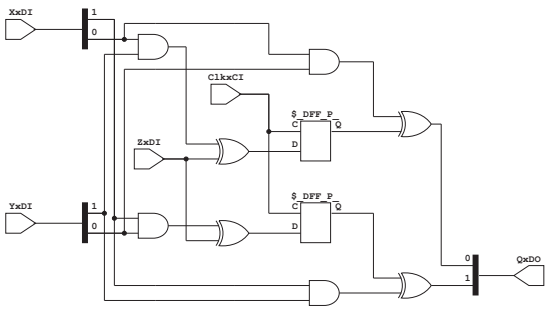

Fig. 3: The DOM-1 multiplication circuit.

Several packages exist for efficiently manipulating decision diagrams. In this paper we use the package CUDD [18], which supports both BDD and ADD representation.

III. Methodology

The overarching goal of this paper is to present a new methodology that facilitates the analysis of a circuit description, in order to provide a proof that it is strongly d-non interferent (d-SNI), i.e., given s outputs as long as $s \leq d$ implies a dependency with maximum i input shares, where i is the number of internal probes [4].

From a high level point of view (see Figure 5), the first step of the proposed methodology is a reading phase of a gate-level circuit description, which is annotated with sensitive variables, sensitive outputs, and their corresponding shares. The description is then "unfolded" to produce all the intermediate probes that can be derived. Moreover, an overall Walsh transform is computed for any combination of either outputs/probes (1). Then, the derived Walsh transform is compressed into a compact representation exploiting Algebraic Decision Diagrams (2). To perform the interference check, the latter is then multiplied by a *relation vector*, which has non-null values only in the regions where the Walsh transform must be zero (3, see also Figure 2). If the resulting value is not zero then it means that the function is not d-SNI; otherwise we pass to the next output/probe combination. The following paragraphs show a detailed description of the above steps.

A. Reading and "unfolding" the circuit description

The tool reads-in a standard intermediate language (ILANG) format as produced by YOSYS tool [19]; Figure 4 shows part an example annotation for the Domain Oriented Masking AND [20] protected at the first order (whose circuit is shown in Figure 3).

The description is extended with a `Maskverif` compliant set of annotations for identifying sensitive inputs (e.g., XxDI), outputs (e.g., QxD0) and additional random bit (ZxDI). Being an implementation protected at the second order, each sensitive value (e.g., XxDI) is encoded in two shares.

"Unfolding" the circuit means deriving the expression of all the possible intermediate nodes in the circuit. This is of course

```
# Generated by Yosys 0.7 ...
...
module \dom_and
  ## public \ClkxCI \RstxBI
  ## input \XxDI
  ## input \YxDI
  ## output \QxDO
  ## random \ZxDI
  ...
  wire width 2 input 3 \XxDI
  wire width 2 input 4 \YxDI
  wire width 1 input 5 \ZxDI
  wire width 2 output 6 \QxDO
  # (other wires and cells)
  ...
```

Fig. 4: Annotated ILANG file

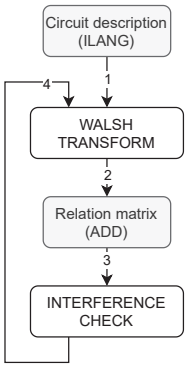

Fig. 5: The methodology proposed in this paper.

a potentially exponential operation whose time increases with the levels of the circuit. Practically, at least for the considered benchmarks, see Section IV, the complexity is still manageable. This part produces a C++ file which builds the BDDs for all the outputs/probes that have been found (by exploiting the C++ bindings of the CUDD library). The idea is that the corresponding manager will be able to build an internal representation exploiting common subexpressions, especially when these correspond to actual factors and co-factors of another function already parsed.

B. Computing the Walsh Spectrum and the corresponding relation matrix

The Walsh transform of each base output/probe is computed through the Fujita Walsh transform [21]; the algorithm works on the BDD representation of the function and returns an ADD whose variables are the bits associated with the spectral coordinates. In principle, one could use this transform to work on any combination of output/probes. However, we have found the performance of the algorithm suboptimal with respect to a simpler computation which exploits the known fact that the row of the correlation matrix associated with multiple base output/probes is proportional to the convolution of the source rows when these are represented in suitable associative container data types. While state of the art solutions are based on list of lists [11], in this paper we propose to adopt hash-based containers (in C++ parlance these are called *unordered maps*). Operations on such containers are $O(1)$ on average and allow fast insertion/update times of the result of the convolution. The data is then converted back in an ADD for further processing.

C. Interference check

The machinery associated with BDD/ADDs allows to quickly prove predicates over the data itself; in particular, one can express and solve existentially quantified predicates, over the convolution W computed above, and have the ADD manager work out the result. The interference check can be defined as a suitable predicate of this form:

$$\exists\alpha.T(\alpha,\rho) \wedge W(\alpha,\rho) \wedge (\rho = 0)$$

where α and ρ are the spectral coordinates of sensitive values and refresh values, $T(\alpha,\rho)$ is a predicate matrix which is equal to 1 only where the convolution W is expected to be 0 (essentially the white areas in Figure 2). If the predicate evaluates to true, then it means that the function is not d-SNI. If the predicate is false then it means that, for this particular combination of output/probes, no vulnerability has been found. However, the search must continue for combinations of up to d among outputs and probes for determining whether the function is d-SNI. To speed up the search it has already been noted [8] that it is useful to start from combinations of the maximum size and evaluate simpler combinations if those are not found vulnerable; this is because there is a low probability that multiple output/probes mask out single output/probe vulnerabilities.

IV. Experimental results

This experimental result section has a threefold goal; *i)* to compare the performance of the proposed methodology with the state of the art exact method in [11], *ii)* to compare alternative implementations of the proposed methodology with varying degree adoption of BDD/ADD, and *iii)* to show a comparison with other current state of the art approaches.

The experiments are based on the benchmarks from the `maskVerif` repository [8]; these benchmarks are a set of primitive cryptographic gadgets implemented to prevent probing attacks. In particular, for the first level of security, we test the Threshold Implementation algorithm (ti-1 in Tables I and II) [22], Trichina (trichina-1) [23] and ISW multiplication (isw-1) [1]; DOM (dom-*) [20] is tested from the first to the fourth level, while the implementation of probing-protected Keccak algorithm (keccak-*) [24] from the first to the third. We run our experiments on a single core Intel Celeron N3150 at 1.601GHz. First we compare the performance of our methodology with the implementation proposed in [11], where the authors exploited a lists of lists (**LIL**) data structure to store the Walsh spectrum and compute both the convolution and the verification over such lists. Table I, shows a comparison between **LIL** and our method (*maps improved* or **MAPI**). The first column refers to the tested security level, while the names of gadgets are listed in the second column; third and fourth columns report the time taken for the implementation with **LIL** and with **MAPI** respectively; the last column shows the speed-up of **MAPI**, computed as the ratio between the two previous columns. The overall execution time and the breakout of the convolution and

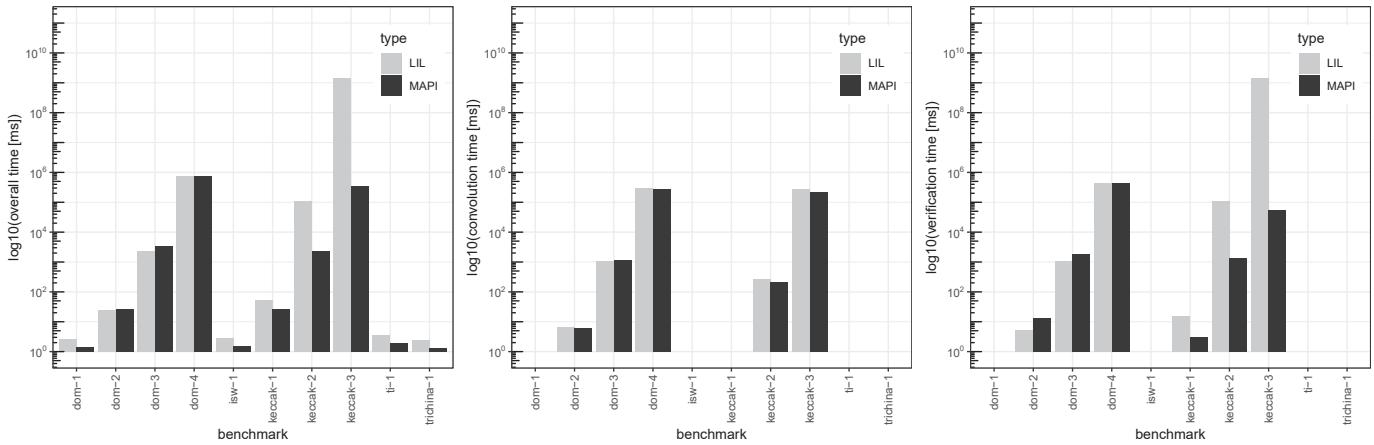

Fig. 6: Comparison of overall (left), convolution (middle) and verification (right) times between the method proposed in [11] (**LIL**) and the proposed method (**MAPI**)

TABLE I: Results of the comparison between our methodology and lists of lists implementation. Values in third and fourth columns are in seconds.

sec. lev.	gadget	LIL	MAPI	speed-up
1	ti-1	0.00367	0.00194	1.89
	trichina-1	0.00248	0.00129	1.93
	isw-1	0.00276	0.00157	1.76
	dom-1	0.00272	0.00145	1.87
	Keccak-1	0.05506	0.02633	2.09
2	dom-2	0.02478	0.02731	0.91
	Keccak-2	106.60330	2.39039	44.6
3	dom-3	2.38042	3.29725	0.72
	Keccak-3	1482378.91197	351.71293	4214.74
4	dom-4	756.00070	740.17401	1.02
median				1.88

TABLE II: Evaluation of different implementation choices. Values from third to sixth columns are in seconds.

sec. lev.	gadget	LIL	FUJITA	MAP	best method
1	ti-1	**1.89**	6.70	1.94	1.89
	trichina-1	**1.93**	10.83	1.96	1.93
	isw-1	**1.76**	9.08	1.79	1.76
	dom-1	1.87	9.74	**1.84**	1.84
	Keccak-1	2.09	**1.37**	2.10	1.37
2	dom-2	0.91	2.44	**0.84**	0.84
	Keccak-2	44.6	**5.19**	30.89	5.19
3	dom-3	0.72	1.75	**0.57**	0.57
	Keccak-3	4214.74	**34.76**	1629.05	34.76
4	dom-4	1.02	1.43	**0.56**	0.56
median		1.88	5.94	1.89	1.80

verification operations is presented in Figure 6. Note that the y axis is logarithmic so the breakout is not meant to be additive as one can intuitively think. Numerically, we can note:

- On convolution, the methods are comparable with a slight advantage for **MAPI**, given perhaps the faster average access time.
- On verification, the use of ADDs by **MAPI** allows a significant speedup which benefits the overall execution time.
- On the overall median, **MAPI** can provide a speedup of 1.88x with respect.
- Whenever the speedup is lesser than one (in only two over ten cases) the difference is less than 30%.
- For Keccak, which is a benchmark of greater complexity with respect the other ones, **MAPI** shows a speedup of at least 3 orders of magnitude.

One could think that applying ADDs also for convolution would imply a better performance. To answer this question we evaluate our methodology with two variants, one in which both computation and verification is done only with hash maps[1] (**MAP**) and the case in which both convolution and verification is done with ADDs (**FUJITA**, using the Fujita method [21]).

[1]https://en.cppreference.com/w/cpp/container/unordered_map

Table II reports the speed-ups of our method (**MAPI**) with respect to all the others (**LIL**, **MAP**, **FUJITA**) while the absolute execution times are shown in Figure 7. Overall **MAPI**'s mixing of hash maps and ADDs improves with respect to all other methods (median 1.8x), except for the DOM benchmark. We suppose that this behaviour is due to Walsh matrices being very sparse and thus not requiring a significant effort in verification.

We conclude by giving in Table III a comparison of **MAPI** with other state-of-art tools, and in particular `maskVerif` [8], the approximate technique proposed by Bloem et Al. in [9] (called Bloem's in the Table) and SILVER [12]. Being exact, **MAPI** implies obviously more computation time respect the first two heuristic methods, but not so much more, especially for Keccak-3. Instead, the comparison with SILVER is difficult, due to the different choice of benchmarks; in this case, for DOM algorithm, SILVER and **MAPI** seem to need close processing time. Note that some results in Bloem's column are marked by a *, because the benchmarks provided for their tool in [9] only concern the verification of one secret instead of all 5 secrets of elaborated by the gadget; also, their technique verifies probing security and not the strong non-interference.

V. Conclusions

In this work, we propose a new methodology that allows to exactly verify strong-non-interference properties of a gadget;

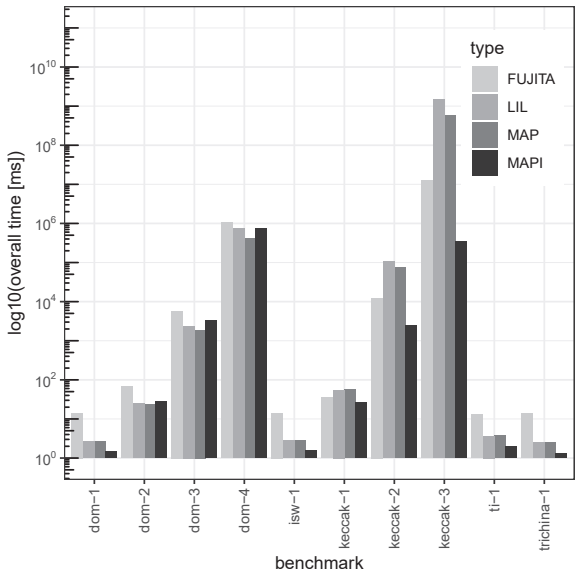

Fig. 7: Comparison of overall computation times of the proposed method (**MAPI**) and other implementations analysed in the experimental results.

TABLE III: Comparison between **MAPI** and the state-of-art tools: `maskVerif` [8], Bloem's [9] and SILVER [12]. Values from third to sixth columns are in seconds.

		heuristic		exact	
sec. lev.	gadget	`maskVerif`	Bloem's	SILVER	**MAPI**
1	ti-1	0.01	$\leq$1	–	0.0019
	trichina-1	0.01	$\leq$1	–	0.0013
	isw-1	0.01	$\leq$1	–	0.0016
	dom-1	0.01	$\leq$1	0.0	0.0015
	Keccak-1	0.01	$\leq$1	–	0.0263
2	dom-2	0.01	$\leq$1	0.0	0.0273
	Keccak-2	0.2	$\leq$10*	–	2.3904
3	dom-3	0.04	$\leq$4	3.7	3.2972
	Keccak-3	41	$\leq$240*	–	351.7129
4	dom-4	0.34	$\leq$120	–	740.1740

our approach combines both hash maps and ADDs and provides, on a standard set of use cases, a median speed-up of 1.88x against other exact methods. Timing-wise, the results are also not dramatically far from *non-exact* approaches appeared in literature. We reserve for the future the more detailed inspection about the improvement's gap between Keccak and DOM algorithms with **MAPI** method. Another possible future work is the application of our tool to more complex gadgets, with higher security levels and by exploiting parallelization. Moreover, also we expect to include the verification of other probing security properties (e.g., PINI [25]).

REFERENCES

[1] Y. Ishai, A. Sahai, and D. Wagner, "Private Circuits: Securing Hardware against Probing Attacks," in *Advances in Cryptology — CRYPTO 2003*, ser. L.N. in C.S. Springer, 2003, pp. 463–481.

[2] J.-S. Coron, "Higher Order Masking of Look-Up Tables," in *Advances in Cryptology — EUROCRYPT 2014*, ser. L.N. in C.S. Springer, 2014, pp. 441–458.

[3] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, B. Grégoire, P.-Y. Strub, and R. Zucchini, "Strong Non-Interference and Type-Directed Higher-Order Masking," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 116–129.

[4] G. Barthe, S. Belaïd, F. Dupressoir, P.-A. Fouque, and B. Grégoire, "Compositional Verification of Higher-Order Masking: Application to a Verifying Masking Compiler," *IACR Crypt. ePrint Arc.*, vol. 2015, p. 506, 2015.

[5] S. Belaïd, D. Goudarzi, and M. Rivain, "Tight Private Circuits: Achieving Probing Security with the Least Refreshing," *IACR Crypt. ePrint Arc.*, no. rn 439, 2018.

[6] G. Barthe, F. Dupressoir, S. Faust, B. Grégoire, F.-X. Standaert, and P.-Y. Strub, "Parallel implementations of masking schemes and the bounded moment leakage model," *Lecture Notes in Computer Science*, vol. 10210 LNCS, pp. 535–566, 2017.

[7] S. Faust, V. Grosso, S. M. D. Pozo, C. Paglialonga, and F.-X. Standaert, "Composable Masking Schemes in the Presence of Physical Defaults and the Robust Probing Model," *IACR Crypt. ePrint Arc.*, no. rn 711, 2017.

[8] G. Barthe, S. Belaïd, G. Cassiers, P.-A. Fouque, B. Grégoire, and F.-X. Standaert, "maskVerif: Automated analysis of software and hardware higher-order masked implementations," *IACR Crypt. ePrint Arc.*, no. rn 562, 2018.

[9] R. Bloem, H. Gross, R. Iusupov, B. Könighofer, S. Mangard, and J. Winter, "Formal Verification of Masked Hardware Implementations in the Presence of Glitches," in *Advances in Cryptology — EUROCRYPT 2018*, ser. L.N. in C.S. Springer, 2018, pp. 321–353.

[10] L. D. Meyer, B. Bilgin, and O. Reparaz, "Consolidating Security Notions in Hardware Masking," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 119–147, May 2019.

[11] M. C. Molteni and V. Zaccaria, "On the spectral features of robust probing security," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 24–48, Aug. 2020.

[12] D. Knichel, P. Sasdrich, and A. Moradi, "Silver – statistical independence and leakage verification," in *Advances in Cryptology – ASIACRYPT 2020*. Cham: Springer International Publishing, 2020, pp. 787–816.

[13] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi, "Algebraic Decision Diagrams and Their Applications," *Formal Methods Syst. Des.*, vol. 10, no. 2/3, pp. 171–206, 1997.

[14] G. Z. Xiao and J. L. Massey, "A spectral characterization of correlation-immune combining functions," *IEEE Transactions on Information Theory*, vol. 34, no. 3, pp. 569–571, May 1988.

[15] C. Carlet, "Vectorial Boolean Functions for Cryptography," in *Boolean Models and Methods in Mathematics, C.S., and Engineering*. Cambridge: Cambridge University Press, 2010, pp. 398–470.

[16] J. Daemen, R. Govaerts, and J. Vandewalle, "Correlation matrices," in *Fast Software Encryption*, ser. L.N. in C.S. Springer, 1995, pp. 275–285.

[17] R. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, no. 8, pp. 677–691, 1986.

[18] F. Somenzi, "Cudd: Cu decision diagram package-release 2.4. 0," *University of Colorado at Boulder*, 2012.

[19] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/.

[20] H. Gross, S. Mangard, and T. Korak, "Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order," in *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, ser. TIS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 3.

[21] M. Fujita, J. Chih-Yuan Yang, E. Clarke, Zudong Zhao, and P. McGeer, "Fast spectrum computation for logic functions using binary decision diagrams," in *Proceedings of IEEE International Symposium on Circuits and Systems - ISCAS '94*, vol. 1. London, UK: IEEE, 1994, pp. 275–278.

[22] S. Nikova, V. Rijmen, and M. Schläffer, "Secure Hardware Implementation of Nonlinear Functions in the Presence of Glitches," *Journal of Cryptology*, vol. 24, no. 2, pp. 292–321, Apr. 2011.

[23] E. Trichina, T. Korkishko, and K. H. Lee, "Small size, low power, side channel-immune aes coprocessor: Design and synthesis results," in *Advanced Encryption Standard – AES*. Berlin, Heidelberg: Springer, 2005, pp. 113–127.

[24] H. Gross, D. Schaffenrath, and S. Mangard, "Higher-order side-channel protected implementations of keccak," in *2017 Euromicro Conference on Digital System Design (DSD)*, 2017, pp. 205–212.

[25] D. Goudarzi, T. Prest, M. Rivain, and D. Vergnaud, "Probing security through input-output separation and revisited quasilinear masking," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2021, no. 3, p. 599–640, Jul. 2021. [Online]. Available: https://tches.iacr.org/index.php/TCHES/article/view/8987