IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2021, No. 3, pp. 202–234. DOI:10.46586/tches.v2021.i3.202-234

Breaking Masked Implementations with Many Shares on 32-bit Software Platforms

or When the Security Order Does Not Matter

Olivier Bronchain and François-Xavier Standaert

Crypto Group, ICTEAM Institute, UCLouvain, Louvain-la-Neuve, Belgium.
{olivier.bronchain,fstandae}@uclouvain.be

Abstract. We explore the concrete side-channel security provided by state-of-the-art higher-order masked software implementations of the AES and the (candidate to the NIST Lightweight Cryptography competition) Clyde, in ARM Cortex-M0 and M3 devices. Rather than looking for possibly reduced security orders (as frequently considered in the literature), we directly target these implementations by assuming their maximum security order and aim at reducing their noise level thanks to multivariate, horizontal and analytical attacks. Our investigations point out that the Cortex-M0 device has so limited physical noise that masking is close to ineffective. The Cortex-M3 shows a better trend but still requires a large number of shares to provide strong security guarantees. Practically, we first exhibit a full 128-bit key recovery in less than 10 traces for a 6-share masked AES implementation running on the Cortex-M0 requiring 2^{32} enumeration power. A similar attack performed against the Cortex-M3 with 5 shares require 1,000 measurements with 2^{44} enumeration power. We then show the positive impact of lightweight block ciphers with limited number of AND gates for side-channel security, and compare our attacks against a masked Clyde with the best reported attacks of the CHES 2020 CTF. We complement these experiments with a careful information theoretic analysis, which allows interpreting our results. We also discuss our conclusions under the umbrella of "backwards security evaluations" recently put forwards by Azouaoui et al. We finally extrapolate the evolution of the proposed attack complexities in the presence of additional countermeasures using the local random probing model proposed at CHES 2020.

Keywords: Higher-Order Masking · Bitslice Software · Physical Security Evaluations · Profiled Side-Channel Analysis · Dimensionality Reduction · SASCA

1 Introduction

Motivation and goals. Masking (aka secret sharing) is a popular countermeasure against side-channel attacks. Ideally, it amplifies the noise of an implementation exponentially in a security parameter (known as the security order) that depends on the number of shares [CJRR99, ISW03, PR13, DDF14, DFS15]. Concretely, this exponential security increase only occurs under two assumptions. First, each leakage sample obtained by the adversary should depend on one (linear combination of) share(s) only, which is often referred to as the independence condition. Failing to meet this condition can reduce the security order [MPG05, NRS11, CGP+12, BGG+14]. Second, the leakage samples should be sufficiently noisy. Such strong theoretical guarantees recently motivated the NIST to initiate an effort in order to standardize masking schemes for the protection of single devices against side-channel attacks.[1] This naturally raises the question of what are

[1] https://csrc.nist.gov/publications/detail/nistir/8214a/final.

Received: 2020-10-15 Accepted: 2020-12-15 Published: 2021-07-09

the attack techniques that should be used in order to understand the concrete security guarantees that such schemes provide, first when implemented as stand-alone solutions, next in combination with other countermeasures if needed.

In this paper, we are therefore interested in the practical evaluation of the security that masked software implementations can provide. Somewhat surprisingly, the amount of public research in this direction is quite limited. Examples include multivariate attacks against masked tables' re-computation algorithms [TWO13, BGNT18] and their recent application to an open-source affine masked implementation proposed by the French ANSSI [BKPT, BS20]. These investigations put forward that implementing masking securely in a software device with limited noise is a challenging task, and suggest bitslice implementations as one of the natural directions to reach higher security levels. Yet, and to the best of our knowledge, state-of-the-art bitslice masking schemes (with arbitrary protection order) such as the one of Goudarzi and Rivain [GR17] have never been evaluated against advanced (e.g., horizontal) side-channel attacks.

We extend this limited state-of-the-art by analyzing different implementations of bitslice masking in ARM Cortex-M0 and ARM Cortex-M3 devices. Given the limited noise level that such devices intrinsically provide, we focus in particular on attacks aiming at reducing the noise level in order to make the countermeasure ineffective. Precisely, this goal is calling for attacks mixing multivariate statistics (e.g. using dimensionality reduction to extract as much information as possible on each share [APSQ06, SA08]), and analytical strategies enabling the exploitation of all the shares' manipulations, for multiple intermediate computations [VGS14]. In other words, our goal is to describe tools that allow discussing the balance between the noise, the security order and the number of measurements to perform a successful attack, in a systematic manner.

Contributions. Our main results in this direction are the following ones:

- We show that despite the fact that bitslice masking provides better opportunity for secure implementation than the table-based solutions considered in [BKPT, BS20], it remains hard to reach high security levels in the investigated low-end devices. Attacks against the Cortex-M0 succeed with less than 10 traces for up to 6 shares. Attacks against the Cortex-M3 show a slightly better trend (due to a slightly higher noise) but theoretical predictions suggest that 14 shares would be needed to reach security with up to 10^9 traces. To the best of our knowledge, this is the first report of attacks against masked implementations with such large number of shares.

- We show that contrary to the hardware case, where the noise level can be tightly controlled and the security level primarily depends on the security order [MPL+11, CRB+16, GMK17], the best attacks against low-noise software do not take advantage of reduced security orders and rather exploit the limited noise directly.

- We show that bitslice ciphers with limited number of AND gates better resist the advanced (multivariate, horizontal and analytical) attacks we consider. This observation is of interest for the ongoing NIST Lightweight Cryptography standardization effort since several candidates use bitslice Sboxes like Clyde.[2]

- We apply our attacks against Clyde to the data set made available for the CHES 2020 CTF (https://ctf.spook.dev/) and publish our code under an open-source licence which allows reproducing our results. We additionally compare our results to the best reported attacks of the CHES 2020 CTF that leverage an (expensive) profiling using deep learning. We observe that the analytical strategies we propose reach similar efficiency as (and sometimes outperform) these attacks with limited profiling efforts, and provide a useful possibility to extrapolate conclusions.

[2] https://csrc.nist.gov/projects/lightweight-cryptography.

- We complement our attack results with a careful information theoretic analysis of all the target intermediate variables that they exploit, which allows us to provide a comprehensive explanation and interpretation of these results.

- We finally discuss the evolution of the proposed attack complexities in the presence of additional countermeasures using the local random probing model proposed at CHES 2020 [GGSB20]. Doing so, we highlight that since our target devices have limited physical noise (i.e. the limited security they provide leverages algorithmic noise), some usually considered options to improve security against horizontal attacks (e.g. more refreshing operations [BCPZ16, CS19]) have limited effectiveness.

We then conclude the paper by discussing tracks in order to further improve our attacks and directions to reach higher security levels on (ARM Cortex like) 32-bit devices.

Cautionary note regarding side-channel evaluation methodologies. Evaluating side-channel security is a challenging task and the outcome of an evaluation is in general dependent on the underlying adversarial assumptions. For example, does the adversary know implementation details and can she profile with known randomness?

In this paper, we follow the recent proposal of "backwards evaluations" put forward by Azouaoui et al. [ABB+20]. It suggests to start side-channel security evaluations with the strongest adversarial capabilities in order to approach the (easier to define) worst-case security level of an implementation, and to discuss how relaxed capabilities affect the worst-case attacks' complexities in a second step. We follow this approach for two main reasons. First, we believe targeting worst-case adversaries remains the best and most stable way to define security in general. By contrast, attempting to define a "best practical attack" is difficult because "practical" is a somewhat subjective notion that may evolve with time (and has significantly evolved over the last years, as for example reflected by the permanently updated list of attacks by the JHAS – the JIL Hardware-related Attacks Subgroup). Second, and as a result of this improved stability, worst-case security provides a sound way to compare masked implementations [SMY09]. , which is necessary to guide the discussions towards a standardization effort.[3] Based on these reasons, our investigations start by considering strong profiling capabilities, and we discuss the impact of some relaxations in Section 6, for example the resistance of our attacks against inter-device profiling, which was not yet discussed for analytical attacks in the literature.

We insist that such a backwards evaluation proposal was put forward by authors with various types of affiliations (namely academia, industry, evaluation laboratories and security agencies). So we believe the tools we describe in this paper, which naturally integrate this proposal, have both theoretical an practical interest and are good candidates for analyzing the security of masked software implementations systematically.

Related works. We next consider bitsliced implementations and note that sharesliced implementations have been analyzed against worst-case side-channel attacks in [JS17, GPSS18]. The differences between our results and these previous works are threefold. First, [JS17, GPSS18] provide bounds against worst-case attacks which include quite large "risk factors", for example due to security order reductions or noise reductions. We rather perform concrete key recovery attacks for various security orders, which allows us to make these risks concrete. Second, the parallel nature of the sharesliced implementations makes their evaluation significantly easier in terms of statistical modeling, since removing the need to characterize multivariate distributions as in the bitslice case. Finally, shareslicing has been shown to be a more risky solution (than bitslicing) in terms of security order reductions due to glitches when implemented in an ARM Cortex device [GMPO20]. So while not precluding the possibility that sharesliced implementations can lead to secure

[3] See the Workshop on Threshold Schemes for NIST-Approved Symmetric Block Ciphers in a Single-Device Setting for a discussion: https://www.esat.kuleuven.be/cosic/events/tis-online-workshop/.

and efficient designs in other contexts, bitslicing seems to be a more conservative approach for the devices that we investigate in the current state-of-the-art.

2 Background

In this section, we introduce the tools and notions used in the rest of the paper. We first describe how to extract and quantify information about a given intermediate variable from leakage samples. Second, we describe a heuristic method to recombine information from many variables called Soft Analytical Side-Channel Attack (SASCA). Third, the necessary details about the masking countermeasure we implemented are given.

2.1 Template attacks and information theoretic metric

In order to extract information about a sensitive variable (x), a side-channel adversary can estimate the Probability Density Function (PDF) of the n-dimensional leakage trace $\vec{l}$ given x. To do so, [CRR02] proposed to approximate this PDF with a Gaussian assumption such that

$$\hat{\mathsf{f}}[\vec{l}|x] = \frac{1}{\sqrt{(2\pi)^n |\hat{\Sigma}_x|}} \exp^{-\frac{1}{2}(\vec{l}-\hat{\mu}_x)' \hat{\Sigma}_x (\vec{l}-\hat{\mu}_x)}, \tag{1}$$

where $\hat{\mu}_x$ and $\hat{\Sigma}_x$ are respectively the estimated means and covariances of the leakage given x. The adversary can then use Bayes's theorem to obtain $\hat{\Pr}[x|\vec{l}]$.

In case of a large number of dimensions n, estimating theses distributions might require a large amount of profiling data. In order to limit profiling efforts, the leakage traces can be projected to a linear subspace [APSQ06, SA08]. These will heuristically concentrate the useful information contained in $\vec{l}$ in a reduced number of dimensions n'. Then, the parameters of Equation (1) can be estimated on the projected traces. In this paper, the linear subspace is obtained through a Linear Discriminant Analysis (LDA). We assume that $\vec{\Sigma}_x$ is independent of x so that a single covariance matrix has to be evaluated.

In order to evaluate the amount of information that can be extracted based on that PDF, one can use the Perceived Information (PI) as introduced in [RSV+11] and more formally studied in [BHM+19]. More precisely, given an estimated PDF, we evaluate the PI by sampling and evaluating the equation

$$\hat{\mathrm{PI}}(X, \vec{L}) = \mathrm{H}(X) + \sum_{x \in X} \Pr[x] \sum_{i=1}^{n_t(x)} \frac{1}{n_t(x)} \cdot \log_2 \hat{\Pr}_{\mathrm{model}}[x|\vec{l_i}(x)], \tag{2}$$

where $\mathrm{H}(X)$ is the entropy of X, $n_t(x)$ is the number of fresh (test) leakage samples corresponding to the intermediate value x and $\hat{\Pr}_{\mathrm{model}}[x|\vec{l_i}(x)]$ is the conditional probability of x assigned by the model. Informally, the PI characterizes the amount of information that an adversary can extract from side-channel measurements with a (possibly biased) model. It directly characterizes the complexity of a side-channel attack against the target X using this model (i.e. the number of measurements needed for the attack is inversely proportional to the PI [DFS15]). Concretely, Equation (2) is estimated thanks to cross-validation: a part of the profiling traces is used to build the model, the remaining part if used to test it with fresh samples. In this paper, we use a 10-fold cross-validation.

2.2 Soft analytical side channel attacks

For an adversary to extract all the information within the circuit, she has to estimate the marginal probability on that entire circuit. This strategy rapidly becomes out of reach for

practical adversaries since its complexity grows exponentially with the circuit size [GS18]. A SASCA adversary aims at estimating this marginal probability of the entire circuit from the marginal probability on each of its intermediate values and the knowledge of the circuit. This adversary operates in three steps that we describe next. We refer to [VGS14] for a more formal description of such analytical attacks.

- In the first step, she builds a factor graph of the circuit as represented in Figure 1. A factor graph is a bipartite graph where the nodes either represent variables (i.e. A,B,C,D) or represent functions (i.e. $\oplus$ and $\otimes$). These operations and variables are linked through edges in order to represent the equations of the circuit. In this example, a circuit with equations $C = A \oplus B$ and $D = A \otimes B$ is laid out.

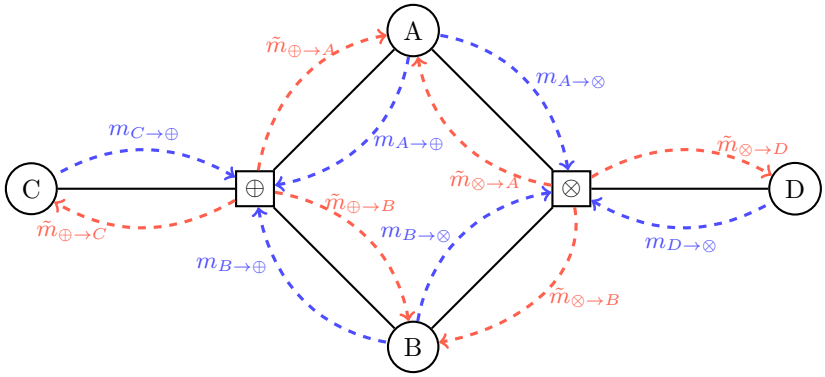


Figure 1: Example of factor graph for SASCA.

- Second, when she receives a leakage sample $\vec{l}$, she computes a marginal probability $\Pr_v[x|\vec{l}]$ for each variable nodes v within the graph. In this work, the marginal probabilities are derived from estimated PDFs similarly to Subsection 2.1. These templates are preliminary built from a profiling phase.

- Third, she runs an iterative algorithm called Belief Propagation (BP). At a high level, it allows reinforcing the knowledge (belief) about one node in the graph by looking at the belief of its neighbors. This is done through passing messages between neighbors along the edges of the factor graph (colored arrows in Figure 1). At a lower level, this is performed by iteratively updating messages according to the following rules. From variables to functions, the update rule is given by

$$m_{v\to f}^{(t+1)}(y) = \Pr_v[y|\vec{l}] \prod_{k\in\partial v\setminus f} \tilde{m}_{k\to v}^{(t)}(y), \tag{3}$$

  and from functions to variables the update rule is given by

$$\tilde{m}_{f\to v}^{(t+1)}(y) = \sum_{\boldsymbol{x}\in\partial f\setminus v} \left(\psi_f(\boldsymbol{x}, y) \prod_{k\in\partial f\setminus i} m_{k\to f}^{(t+1)}(x_k) \right), \tag{4}$$

  where ∂ denotes the neighbors of a node and $\psi_f(\cdot)$ is the compatibility function.[4] Overall, the message passed by a variable node to a function node (i.e. Equation (3)) is the product of the messages it receives from the other edges and the initial $\Pr_v[\cdot]$. The message passed by a function node to a variable node (i.e. Equation (4)) is the sum of the messages received from the other edges.

[4] It is equal to one if the inputs of f are coherent with its outputs and equal to zero otherwise.

This SASCA adversary is proven to converge to the correct marginal probability (and therefore the optimal attack) under two hypothesis. The first assumption is that the PDF estimation on all the variable nodes are independent (which is natural in a side-channel software context). The second one is that the graph does not contain cycles and so has a tree structure (which is rarely the case in the side-channel context). If not fulfilled the method becomes heuristic. Nevertheless, it can offer significantly reduced data complexity compared to other profiled attacks and provides a reasonable estimate of the complete marginal distribution [GS15, BCPZ16, GS18, GRO18, KPP20].

In this paper, we only consider Boolean additions and multiplications operating on 8 bits in parallel ($b = 8$) as function nodes. For a generic function with two inputs, the straightforward evaluation of Equation (4) has complexity in $\mathcal{O}(2^{2b})$, but this complexity can be reduced to $\mathcal{O}(b \cdot 2^b)$ for the Boolean addition thanks to the Walsh-Hadamard Transform [LD16]. Eventually, the messages are normalized and tiled to avoid zeros.

2.3 Boolean masking

In order to provide a protection against side-channel attacks, Boolean masking represents any sensitive variable x as an encoding, denoted $\boldsymbol{x}$. This encoding is a tuple containing d shares such that

$$x = x_0 \oplus x_1 \oplus \ldots \oplus x_{d-1}, \tag{5}$$

where $d-1$ shares are selected independently at random. In order to gain information about x, an adversary has to observe all the shares in $\boldsymbol{x}$ since smaller sets are independent of x. Such a security guarantee can be analyzed for full circuits in the so-called probing model [ISW03]. If the adversary is given $d-1$ values of his choice (i.e. probes) within the entire circuit, her observations have to remain independent of any sensitive (unshared) variable. If this property is ensured, the circuit is called $(d-1)$-probing secure.

Standard building blocks of probing secure circuit are algorithms for additions (Algorithm 1 in Appendix) and multiplications (Algorithm 2 in Appendix) that take as input two encodings and return a third one. Additions are performed share by share and have linear cost in d. Multiplications are more costly. In this paper, we only consider the ISW multiplication gadget that has a cost that is quadratic in d [ISW03]. Informally, it first computes d^2 partial products by multiplying all the shares together and then compresses them securely into the output thanks to additional fresh randomness.

Probing security reduces to the more practical "noisy leakage security" where the adversary has access to noisy observations on all the shares [DDF14]. In that model, it is expected that the mutual information on X is the product of the mutual informations of its shares $\boldsymbol{X}$ [DFS15]. In effect, this can provide noise amplification, meaning that the information about X decreases exponentially with d, which implies that the data complexity (N) of any attack increases at the same rate such as

$$N \geq \frac{c}{\mathrm{MI}(X, \vec{L})^d}, \tag{6}$$

from [DFS15], Equation 20. However, this reduction and the corresponding security guarantees require two hypothesis. The first one is that each leakage sample depends on one (linear combination of) share(s) only. The second one is that the information on a share is sufficiently small so that noise can be amplified by masking. In this paper, we specifically focus on attacks that invalidate the second assumption.

3 Targets' description

We now detail the higher-order ($d > 2$) bitslice masked implementations that we analyse in this work. First, we describe the two micro-controllers (MCUs) that run the software,

as well as the associated measurement setups. Second, we describe the software itself, which relies on the state-of-the-art bitslice implementations of Goudarzi and Rivain [GR17], and their recent optimization by Belaid et al. [BGR18]. Eventually, we detail the similar implementation for the tweakable block-cipher Clyde [BBB+20].

3.1 Micro-controllers and measurement setup

Higher-order software masking is typically aimed at protecting MCUs that do not embed a side-channel resistant hardware implementation. 32-bit MCUs are natural candidates for this purpose and the masking scheme by Goudarzi and Rivain is optimized for these platforms. Similar MCUs are used in benchmarks efforts for the NIST LWC competition [RPM19], confirming that obtaining some protection against side-channel attacks on such platforms is a concretely-relevant goal. Masking has been already implemented and analysed on similar devices [BBC+20b, BS20]. Therefore, we selected two targets with 32-bit architecture from ARM. Namely, we study a Cortex-M0 and a Cortex-M3.[5]

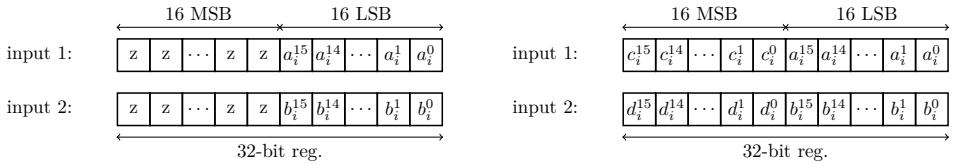
Regarding the measurement setup we reproduce the setup from [BBC+20b] to run our AES implementation. Namely both MCUs were mounted on their off-the-shelf demonstration board. We used the STM32F0DISCOVERY board for the Cortex-M0 and the STM32VLDISCOVERY board for the Cortex-M3. In both cases, a similar effort has been made in order to have clean measurements, which required slight board modifications. All the decoupling capacitors/inductors on the power grid have been removed. After this operation, both MCUs were still functional and no behavioral modifications have been noticed. An external 8 MHz crystal oscillator has been mounted on the boards to derive the system clock. It was set to the maximum of each of the targets thanks to an internal phase locked loop (i.e. 48 MHZ for the Cortex-M0 and 24 MHz for the Cortex-M3). The leakages were measured with a current probe (i.e. the CT1 from Tektronix 1 GHz) placed on a jumper between the on-board power regulator and the MCU under test. This signal was sampled by a PicoScope 5244D at 500 MSamples/s with 12-bit resolution. All the following results are obtained by processing the raw traces fetched from the scope. No signal pre-processing nor averaging is preliminary performed. The experiment for Clyde are performed on the public dataset made available for the CHES2020 CTF.

3.2 Goudarzi and Rivain's bitslice masked implementations

In order to reduce the cost a masked implementation, Goudarzi and Rivain proposed to implement it in a bitslice fashion [GR17]. Bitslice implementations compute functions based on their Boolean representation. To do so, each input bit of the corresponding circuit is stored in different native words of the MCU, that we next call slices.

For example, in order to evaluate an 8-bit function, the Least Significant Bit (LSB) of 8 slices are filled with the input bits. The Boolean circuit is then computed by applying the Boolean operations between the slices. This is natively done in a single cycle on most (if not all) modern processors. Bitslicing allows one to efficiently evaluate multiple times the same Boolean circuit by exploiting parallelism. To do so, the inputs on which the circuit must be evaluated just have to be placed at different indexes in the slices. In the previous example, the 32 bits of a slice can be filled with 32 different input bits on which the circuit must be evaluated. Then, the same bitwise operation on native words can again be applied. For a masked implementation, a similar approach can be taken by additionally ensuring that each share of a given encoding is placed in a different slice (in order to avoid the shareslicing glitch issue observed in [GMPO20]). Additionally, the bitwise operations are replaced by masked gadgets from such as the ones of Appendix A.

[5] For simplicity and comparability, we used the same software instructions in both cases.



(a) Addition. **(b)** Multiplication.

Figure 2: Slices layout for inputs of additions and multiplications, where z denotes an unspecified value and x_i^j is the i-th the share of x in the j-th Sbox.

In the specific case of the AES optimized for 32-bit MCUs, Goudarzi and Rivain proposed to use the Boolean representation of the Sbox given in Appendix B. Because the slices are 32-bit wide, all the 16 Sboxes of the AES can be evaluated in parallel based on that circuit. The representation is composed of three parts: an input linear transformation, a middle non-linear layer and an output linear transformation. It requires 32 multiplications and 83 additions for a total of 123 intermediate variables.

We detail in Figure 2 the layout of the slices for both multiplication and addition gates. In these figures, the slices are filled with the i^{th} shares of the encoding $\boldsymbol{a^j}$, where j is the index of the Sbox. Because multiplication gadgets have a quadratic cost, Goudarzi and Rivain proposed to group independent ISW multiplications by two. The 16 Most Significant Bits (MSBs) are filled with the inputs of the first multiplication (i.e. $\boldsymbol{c} \otimes \boldsymbol{d}$) and the 16 Least Significant Bits (LSBs) with the inputs of the second one (i.e. $\boldsymbol{a} \otimes \boldsymbol{b}$). This allows one to reduce the number of ISW multiplications to execute from 32 to 16. Because the addition gadgets (i.e. $\boldsymbol{a} \oplus \boldsymbol{b}$) have linear cost in d, no particular optimization is performed on them. The slices are filled according to Figure 2a where the encodings are placed on the 16 LSBs of the slices. The 16 MSBs are not specified since unused within the addition gates. In our implementation, the value of the MSBs depends on the gadget producing that encoding. More precisely, if it is the result of a multiplication, the MSBs may contain another sharing. This happens if the encoding of interest was already on the LSBs during the multiplication. By contrast, these MSBs are set to zero if the encoding of interest was placed on the MSBs during the multiplication. In this case, the slices of the multiplication have been right shifted to place the encoding on the LSBs (which sets the MSBs to zero). Besides, if the encoding is the output of another addition gadget, the MSBs are simply set as the addition of the MSBs at its inputs.

Eventually, while [GR17] inserted refresh gadgets at the input of each multiplication to ensure probing security, Belaïd et al. have shown that the circuit remains probing secure without them [BGR18]. We use this optimization to improve the performances of our implementation and reflect the state-of-the-art in software masking.

As for the Clyde tweakable block cipher, its non-linear layer consists in applying 32 independent 4-bit Sboxes. The representation of the Clyde Sbox given in Appendix C, Table 4 allows directly using the full parallelism of the registers, which makes Clyde a well suited candidate for bitslicing on 32-bit MCUs. Compared to the AES, it removes the need to apply two independent AND gates in parallel in order to fully use the 32-bit registers during ISW multiplications. Eventually, one refresh has to be inserted after the first ISW multiplication to preserve register probing security [BDM+20].

We note that we consider the randomness generation as out of scope for this work. Therefore, in our targets the randomness used is taken from a cryptographic PRG. It is precomputed and stored in a memory to be fetched in a few cycles. The AES implementation is written in standard C language as the code generated by the recently proposed Tornado [BDM+20]. Its compilation is optimized for space to avoid aggressive

optimizations that might modify the Boolean representation of the circuit. We ensured that the compiled code respects the previously described slices layout. For Clyde, the code is mostly written in C but the operations manipulating shares are in assembly.

4 Methodology

In this section, we present our attack methodology against masked bitsliced ciphers. We first describe our factor graph construction for masked implementations. We then describe the specific details of the Goudarzi and Rivain masked AES and masked Clyde targets. Finally, we discuss the complexity of each of the steps in a backward evaluation, similarly to [ABB+20]. As mentioned in introduction, we first investigate an adversary with worst-case capabilities having access to the implementation details, knowing all the randomness during profiling, and attacking the same device as the one used for profiling. Next, in Section 6, we evaluate the impact of relaxing these worst-case assumptions.

4.1 Factor graph for masked boolean circuits

When performing a SASCA, the first step is to define the factor graph. A direct and intuitive solution is to include all the intermediate values processed by the target in the factor graph. However for a masked implementation (e.g. using ISW multiplications), this solution rapidly becomes out of reach. Indeed, the number of intermediate variables grows in $\mathcal{O}(d^2)$ and, during the profiling, a PDF has to be estimated for each of them. When running BP, the graph then includes a quadratic number of multiplication nodes for which the message passing rule is more costly than for the additions nodes (Subsection 2.2). Additionally, such a graph includes loops that make the BP heuristic.

We therefore propose an alternative construction that aims at reducing the graph size, the number of loops and the number of multiplication nodes. Precisely, our construction does not include the internal states within the masked gadgets but only their inputs and outputs. Admittedly, this comes at the cost of trading some of the extracted information for an improved time complexity. The main motivation for this choice is to keep the attacks practical for large number of shares (see Subsection 4.3 for a detailed discussion). As observed in [CS19], the internal values within the ISW multiplications are also the ones providing the least information, hence motivating our tradeoff.

Concretely, Figure 3 illustrates an example of our construction for a masked circuit with a single multiplication. The graph contains two types of sub-graphs. The first one is the *encoding graph*. It estimates the marginal probability of a variable a given its encoding $\boldsymbol{a}$. The second one is the *unmasked graph*. It replaces all the gadgets within the masked implementation by their unprotected equivalent. The probabilities on these unmasked inputs/outputs are taken from their respective encoding graphs.

The first advantage is the reduction of the number of loops. While the unmasked graph depends on the circuit under test and may include loops, the encoding graphs have a tree structure. Under the assumption that the initial distributions on the share nodes are independent (which is sound in our case), the marginal distribution of the secret variable is estimated without heuristics. The second advantage is the reduction of the number of multiplication nodes. A single multiplication node per ISW multiplication is placed in the factor graph, making their number constant in d instead of quadratic.

When applying BP on such a graph, additional implementation tricks can be used. We remark that the messages passed from the encoding graphs to the unmasked graph are independent of the unmasked graph (there is only one edge between them), and the adversary is only interested in the probabilities of the unmasked graph. Therefore, the encoding graphs and the unmasked graph can be processed independently.

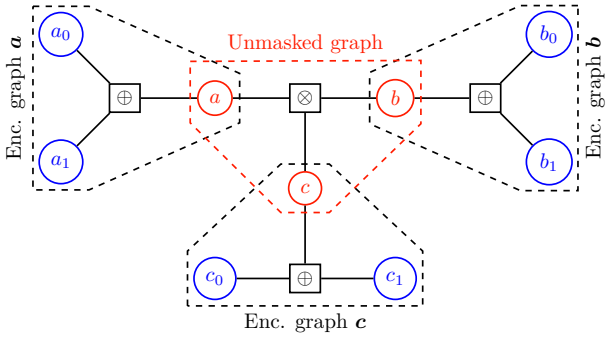


Figure 3: Factor Graph around AND gate for two shares implementation.

First, this observation allows leveraging the fact that the number of iterations required for the encoding graphs and for the unmasked graph is not the same. For encoding graphs, the number of iterations needed to converge is equal to the depth of the tree that is $\log d$. Since the goal is to obtain information on the root node from the leaf nodes, one can only process one stage of the tree at every iteration. Therefore, there is a maximum of d nodes that have to be processed at each iteration. For the unmasked graph, it may contain loops and the number of iterations is not as well defined. We choose twice the graph diameter which was already taken in previous works [VGS14, GS15, GS18, GRO18, KPP20]. Hence, for large enough unmasked graphs (e.g. the ones described next for the AES and Clyde), this trick reduces the number of iterations necessary when running BP.

Second, it allows the adversary to first run BP on all the encoding graphs (possibly leveraging serialization to save memory) and to perform BP on the unmasked graphs afterwards, by keeping the messages passed from the encoding graphs constant.

4.2 Factor graph for (tweakable) block ciphers

The factor graph for the AES masked Sbox is built in a similar manner. To collect information about a single trace, the unmasked graph contains all the function and variable nodes described in Appendix B. In order to attack with multiple traces, several of these graphs have to be built and we denote each of them as a trace graph. The multiple trace graphs can then be connected together. During the BP execution, this will propagate the information from one to the other leading to more efficient attacks [GRO18].

In this work, we link multiple trace graphs through the input linear layer as depicted in Figure 4. There, $\mathcal{K}$ is a set of variable nodes. All of these are either key nodes or an intermediate value of the input linear transformation applied to the key. Each of the nodes in $\mathcal{K}$ is connected to a single node in each trace graph that has the same transformation as the node in $\mathcal{K}$. This connection is done via an addition with a linear transformation of the plaintext. We perform 48 iterations on the unmasked graph which, as previously mentioned, corresponds to twice the graph diameter.

For the AES, we explored multiple options such as connecting only the key nodes, only a few nodes in the linear layer (with different proportions) and we concluded that the best solution was to fully connect that layer. Note that our methodology can be applied to any Boolean masked circuits, but the best trace graph connection remains circuit specific.

By contrast, Clyde does not have a linear layer at the input of its Sbox. Hence, the subgraph $\mathcal{K}$ contains only key nodes, which prevents connecting internal nodes on multiple trace graphs. Not connecting the trace graphs together still leads to successful attacks, but it may impact the data complexity negatively, as observed in [GRO18].

We stress that the construction of a good factor graph is critical for the efficiency of SASCA. Yet, under the worst-case adversarial capabilities considered in this section, we

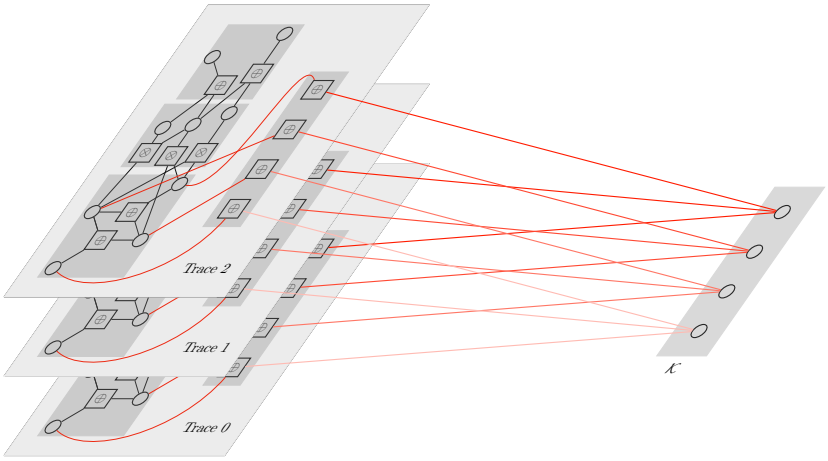


Figure 4: Unprotected graph for a AES SBox like circuit.

have full implementation knowledge. It enables constructing an exact factor graph, which could even be automated given a description of the circuit (e.g. high-level representation, C code or assembly) to avoid the error prone process of writing it by hand. Generalizing the recent Tornado compiler [BDM+20] would be an interesting direction for this purpose.

4.3 Profiling and attack complexities

We now discuss the time and memory complexities of our analysis for each of the steps in [ABB+20]. These are given as $\mathcal{O}$ complexities since practical instantiation allows time vs. memory tradeoffs which depend on the work station used for the analysis and its parameters such as the number of cores available and the RAM or disk access speed.

1. The **measurement** phase records the traces, the input sharings and the seed used by the PRG. Because the execution time of the ISW multiplications is quadratic, the size of the traces grows in $\mathcal{O}(d^2)$. Because the templates estimation methodology is independent of the masking order, the number of traces to collect is constant in d.

2. The **leakage mapping and modelling phase** consist in learning the behaviour of the implementation under test. We divide this profiling phase in three steps.

   (a) The **labelling** consists in deriving the value of all the shares in all the encodings during a recorded trace. This is performed by running a "clone" of the software ran by the target on the same input sharing and PRG seed. Its complexity is also in $\mathcal{O}(d^2)$, but with a smaller constant than step 1. The number of labels outputted at this step equals $123 \cdot d$ for the AES and $13 \cdot d$ for Clyde.

   (b) The **Point of Interests (POIs)** selection is the first processing performed on the traces. It consists in finding the points in time where the leakage contains (first-order) information about a given share. In this work, we compute the Signal-to-Noise Ratio (SNR) defined in [Man04] for all the labels in step 2. Because these are computed on traces of length growing quadratically, the complexity of this step is in $\mathcal{O}(d^3)$. Once the SNR is computed on a share, we keep as POIs at most 3,000 time samples that are (the most) significant ones, in order to limit the complexity of the final profiling step.

   (c) The **LDA + Gaussian Templates** profiling are taken for PDF estimation as given in Subsection 2.1. The input leakages that are sent to the LDA transform

are given by the POIs of step 3. One template is estimated per share, for a total in $\mathcal{O}(d)$ templates. The cost of a template does not increase with d since the number of dimensions is limited to 3,000 in our experiments.

Overall, the bottleneck of the profiling is the SNR computation with its cubic cost. One could reduce this complexity by exploiting the apriori location of the leakages. However, such a method is more heuristic and was not investigated in this work.

3. The **leakage exploitation** consists in recovering the secrets based on leakage samples. In this attack phase, three steps must be performed. We discuss its complexity according to the number of trace graphs t and the number of shares. We note that the time and memory complexities are proportional to the number of nodes in a graph for a single iteration of the BP algorithm.

   (a) The **information extraction** consists in extracting the probabilities from the estimated PDFs. Because there is one estimation for every variable in the encoding graphs, the time complexity of information extraction is in $\mathcal{O}(t \cdot d)$.

   (b) The **information processing** consists in running the BP algorithm. The first step is to process the encoding graphs which requires a total of $\mathcal{O}(d \cdot \log d)$ operations for each encoding graph and therefore the total time complexity grows in $\mathcal{O}(t \cdot d \cdot \log d)$. Additionally, the computation of each encoding graph can be serialized to reduce its memory complexity down to the one of a single encoding graph, which is proportional to $\mathcal{O}(d \cdot \log d)$. In practice, we solve multiple encoding graphs at once, in order to exploit the vectorization of the programming language as well as the multiple cores available on our working station. The second step is to solve unmasked graphs which requires to keep the entire unmasked graph in memory. The latter contains the t trace graphs and the nodes in $\mathcal{K}$ which requires a memory growing in $\mathcal{O}(t)$ as well as its corresponding time complexity. We note that for large t, multiple independent graphs can be built but possibly lead to a loss in attack performances [GRO18].

Concretely, the templates are built on 8-bit words. In the AES case, since the implementation evaluates 16 Sboxes in parallel (which is the maximum parallelism that a single plaintext provides) two independent graphs are constructed, with each of them covering 8 Sboxes. This doubles the cost of all the previously mentioned steps.

We finally note that if the complete graph was exploited (with the internal values of the multiplications), the number of PDFs to estimate would be quadratic and the SNR time complexity would grow in $\mathcal{O}(d^4)$. Furthermore, during the leakage exploitation, this factor graph does not allow serialization and the entire graph must be processed at once for a total memory cost of $\mathcal{O}(t \cdot d^2)$. Because the number of multiplications nodes to process also grows quadratically with d in the complete factor graph, the total time complexity for one BP iteration is therefore in $\mathcal{O}(t \cdot d^2)$, with a large constant (i.e. 2^{16}). Hence, the corresponding attacks rapidly become too expensive compared to the above (simplified) methodology where the number of multiplications is independent of d and the the time complexity is only in $\mathcal{O}(t \cdot d \cdot \log d)$, with a smaller constant (i.e. $8 \cdot 2^8$).

4.4 SCALib: open-source analysis toolbox

Along with this manuscript, we provide a new open-source library, SCALib, that we used to perform our security analysis.[6] Its philosophy is to keep a high-level language to connect the various pieces of the processing, while performing computationally-intensive tasks

[6] https://github.com/simple-crypto/SCALib

with lower-level compiled language. This approach enables high performance and efficient multi-threading while keeping the code easy to interface. Practically, we use Python 3 for generating the factor graph and interfacing with the database. The expensive task which are SNR and templates estimations as well as BP are performed with a Rust custom library. Our code using SCALib to attack Clyde (CHES2020 CTF, [BBC+20b]) is also publicly available.[7]

5 Higher-order attacks in worst-case settings

In the following, we perform attacks against the software implementations of the AES and Clyde presented in Section 3, with a various number of shares. We first select parameters for the PDF estimation and show that the profiling dataset is large enough. Next, we give the attacks' results followed by an extrapolation to larger masking orders. Eventually, we detail which information is exploited by the adversary and compare the attacks.

5.1 Model selection and profiling data complexity

During the profiling, the adversary estimates a PDF in order to extract a maximal amount of information. If she uses the "LDA+Gaussian Template" method, she has to carefully choose the number of dimensions to keep after doing the LDA projection (i.e. n'). This choice depends on her available profiling data. If n' is too small, she may miss information hidden in other dimensions. If n' is too large, the estimation converges more slowly and the collected data may not be sufficient. In order to select this n' parameter, we study the PI of the resulting PDF estimations as defined in Equation (2).

Figure 5 exhibits the PI extracted for an exemplary variable node, according to the number of profiling traces used to build the model, and for various n' values. For both targets, the PI curves increase with n' until a saturation effect appears, meaning that a larger n' will only improve the PI marginally while slowing down its convergence. We observed similar results for all the variable nodes we tested and therefore selected $n' = 6$ for Cortex-M3 and $n' = 11$ for Cortex-M0 based on these results.

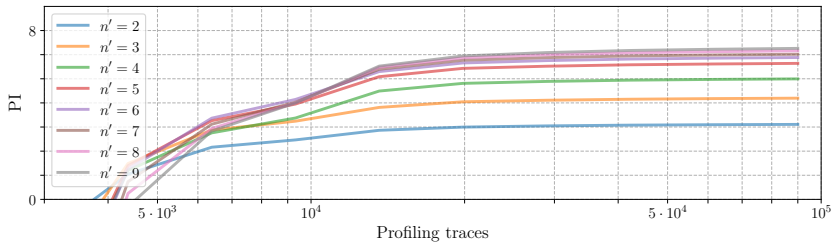
Given these parameters, all our following experiments use 100,000 profiling traces which is sufficient for the required PDF to be well estimated.[8]

5.2 Attack data complexity

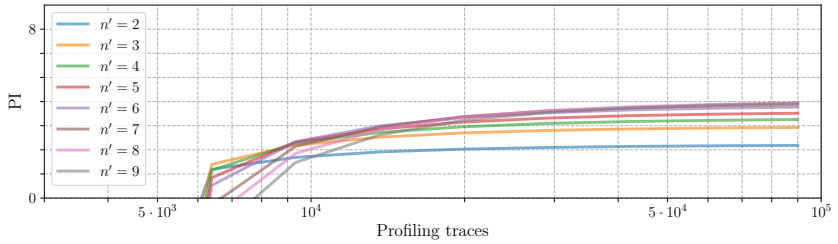
We now report attacks for various masking orders. Precisely, for the AES with 128-bit key we target the Cortex-M0 from three up to six shares, and the Cortex-M3 from three up to five shares. For Clyde (also with 128-bit key), we target implementations up to eight shares. As our extrapolations will show, attacking larger number of shares would be feasible from a data complexity viewpoint. We note that we limited our investigations to 5 and 6 shares for the AES because of technical reasons. Namely, our scope can only capture full traces up to these values and the memory available on our working station should be extended for analyzing larger number of shares. Yet, we insist that those are not fundamental limitations and, for example, a scope/station with more memory or a setup recording traces in multiple chunks combined with further optimizations of the data processing would overcome them, which we leave for future work.

[7] https://github.com/obronchain/BS21_ches2020CTF

[8] The Scikit-Learn implementation of the LDA proposes two solvers. The first one uses singular value decomposition and the second one uses eigenvalue decomposition. We did not notice differences between the PI values that they provide, and therefore selected the second (more efficient) solution which is implemented with SCALib.



(a) Cortex-M0.



(b) Cortex-M3.

Figure 5: PI convergence for various dimensions after projection n'.

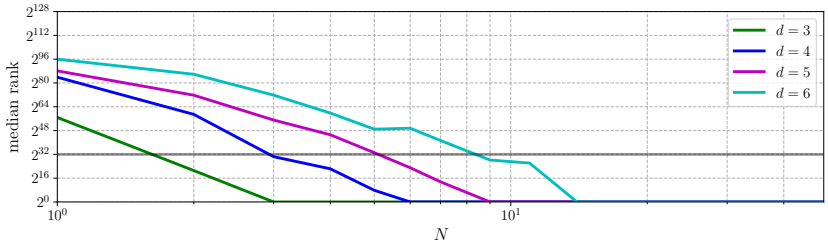
As a result of these considerations, the metric used to evaluate the performances of the attack is the median rank of the full 128-bit key, which is estimated according to [PSG16]. This can be interpreted as the enumeration power that the adversary must spend in order to have one chance out of two to get the correct key. Similarly to [BS20], we conclude that an attack is successful if the median key rank is below 2^{32}. The bottleneck of such a brute force is typically limited by the block cipher execution. Namely, testing 2^{32} keys can be done in less than two minutes on a single core of our working station by relying on the AES implementation of `openssl` while the keys are enumerated in about a second (see Figure 6 in [PSG16]). The median is estimated on 100 independent attacks. For completeness, we report the median rank of the 16 bytes individually as well.

The results of **concrete attacks** against the Cortex-M0 are given in Figure 6. As shown in Figure 6a, all the attacks succeed in less than 10 traces. The three-share implementation is broken with two traces, the four-share one with three traces and the six-share one with nine traces. Similar results are reported for the Cortex-M3 in Figure 7, but with larger complexities. The attack against the three-share target requires 60 traces, against the four-share one about 300 traces and against the five-share one about 2,000 traces.[9] For the attacks against Clyde, 50 traces are needed for the three share version, 120 for 4 shares, 1,100 for 6 shares and 12,000 for 8 shares, as described in Figure 8.

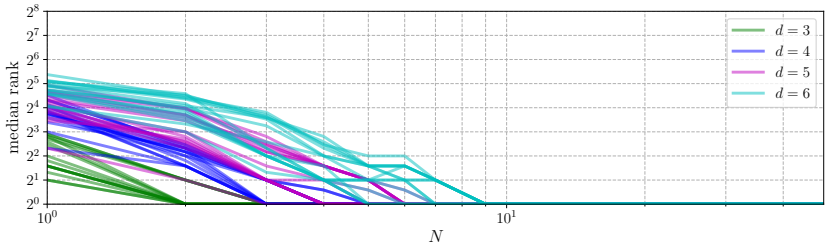
In both cases, we also report the data complexity to recover each byte. We observe that some bytes are more difficult to recover than others, which is in contrast with divide-and-conquer attacks against (e.g. unprotected) AES implementations with table lookups. We assume these differences are due to the asymmetry of the Sbox factor graph.

The previous results allow us to **extrapolate the attack data complexities** to a larger number of shares. Indeed for all targets, we observe that the attacks succeed with a limited data complexity, hinting towards a low physical noise level, especially for the Cortex-M0. Yet, we also observe an exponential increase of the data complexity with the masking order. So our results do not contradict masking security proofs and confirm

[9] For the 5-share Cortex-M3 AES case, we could not fit the factor graph in memory for more than 1060 traces, which is not sufficient to reach the 2^{32} key rank but reduce the key rank down to $2^{43.6}$. The brute force will take 4 days on a single core of our working station and two hours by leveraging the 48 independent cores. The figure is extrapolated for this case.

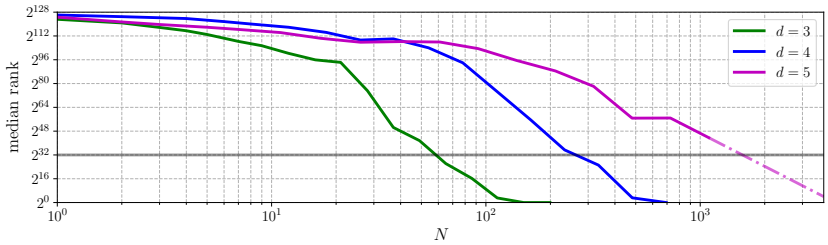


(a) Full 128-bit key.

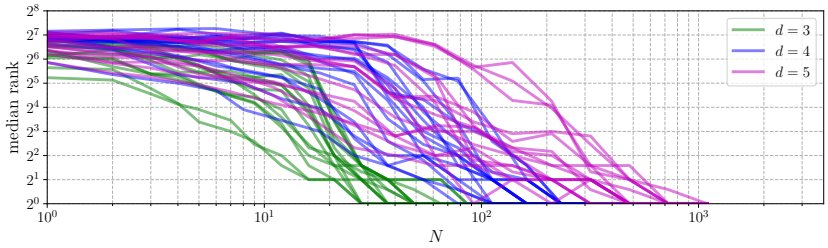


(b) Individual key bytes.

Figure 6: Attacks against the Cortex-M0, AES



(a) Full 128-bit key.



(b) Individual key bytes.

Figure 7: Attacks against the Cortex-M3, AES

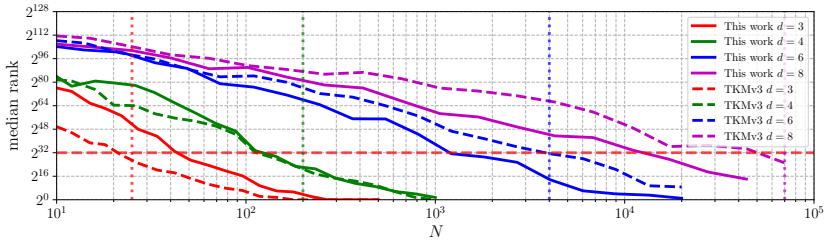


Figure 8: Attacks against the Cortex-M0, Spook. The vertical dashed lines are the numbers reported for the best attack reported in the CHES 2020 CTF [BBC+20b].

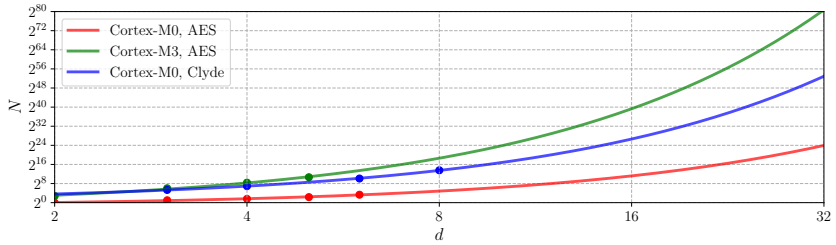


Figure 9: Data complexity of our attacks in function of the number of shares assuming a 2^{32} key enumeration. Markers are for real attacks. Lines are for extrapolations.

that for the considered adversary, the low noise is amplified by the number of shares d as expected from Equation (6). But they also show that the noise level (i.e. the basis of the exponential amplification) is too low for reaching high security with the number of shares we investigated. Based on the observed exponential trend, we propose an extrapolation that consists in evaluating the constant factor and the amplified basis in Equation (6) resulting in the curves reported in Figure 9.[10] For the Cortex-M3, 16 shares provide approximately 2^{40} security and 32 shares lead to 2^{80} security. For the Cortex-M0, the attack complexity does not exceed 2^{24} with 32 shares in the case of AES and 2^{54} for Clyde.

5.3 Information theoretic analysis of the leakages

In order to gain insights about why the previous attacks succeed with a low complexity, we next study the information available for each of the shares. Namely, we first look at the number of dimensions containing information about a share and then analyse this information according to the operations manipulating the shares.

As pointed out in [ABB+20], the results of a worst-case security evaluation depends on the ability of the evaluator to build an efficient measurement setup. As we are interested in a worst-case scenario, we built the best measurement setup we could and our analysis shows that it is possible for an adversary to reduce the physical noise down to a point where it is limited and the main noise source is algorithmic. In Subsection 7.2, we study the case where the adversary is left with noisier measurements.

5.3.1 Impact of the number of leaking dimensions

First, we look at the impact of the number of POIs (n) obtained in Step 3 of Subsection 4.3 on the extracted information. On Figure 10, a bullet corresponds to a single share within the circuit. Its position depends on n (x-axis) and the PI obtained on that share by exploiting these n time samples (y-axis). Overall, we observe that a larger n, meaning that the variable is more manipulated, leads to a larger PI. This shows a general trend that the more a share is involved in various computations, the more information can be extracted about it, thanks to multivariate characterization. We additionally observe (again) that the Cortex-M0 leaks more information than the Cortex-M3. This is expected from the previous attack results where the Cortex-M0 is an easier target.

5.3.2 Impact of the circuit structure

Second, since the previous experiments show that the noise level of the Cortex targets is limited, we searched for a relation between the operations performed on a share and its actual PI. An important motivation for this study is to try deducing whether the (limited)

[10] These are estimated by minimizing the mean squared error between the extrapolation and the experimentally observed data complexities in Figures 6a and 7a and 8.

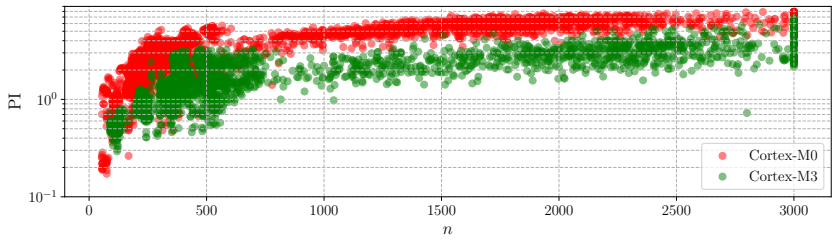


Figure 10: Relation between the number of profiled dimensions and the extracted information per share. Each point is for a single share in the AES implementation.

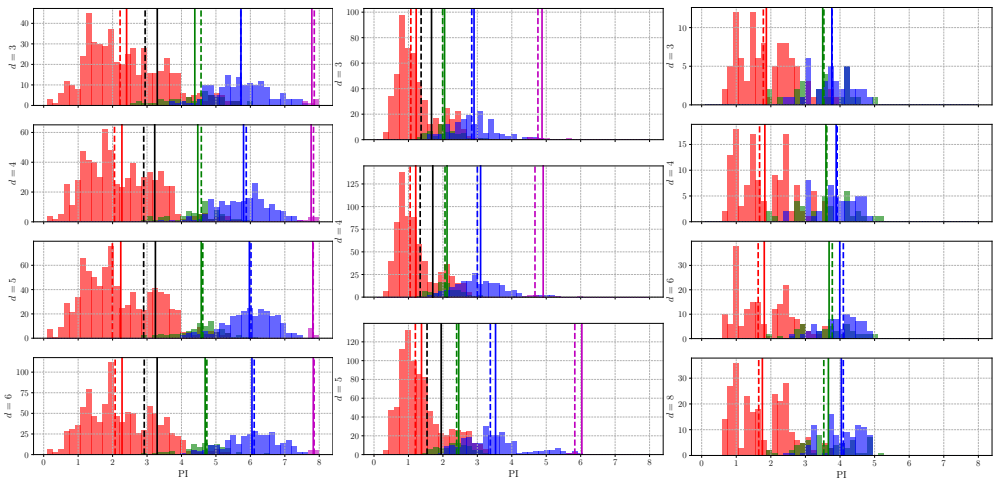
security we observe is due to algorithmic noise (i.e. noise coming from data that is not estimated by the adversary's model) or physical noise (which is inherent to the targets and measurement setups). For this purpose, we analyze two parameters. First, the number of times the encoding is used by a multiplication gadget, which we denote as m. Second, the number of times the encoding is used by an addition gadgets, which we denote as a. We also discuss the impact of the masking order d on the information extracted.

We start by studying the impact of the **number of multiplications** m in Figure 11, which contains histograms of the PI measured for all the profiled shares within the encoding graphs. The color codes correspond to the value m of the share. For example, red histograms contain shares that are not used in an ISW multiplication so that $m = 0$ (i.e. these shares are only used in the linear layers), the blue ones are used in two different ISW multiplications (i.e. $m = 2$), ... The different plots for a given target correspond to various security orders, with low number of shares at the top of the figure and larger number of shares at its bottom. We see that increasing m leads to a significantly higher PI (which is in line with the circuit size parameter of masking security proofs) and this is effect is stronger for the AES than for Clyde. This can be explained by the optimization considered by Goudarzi and Rivain for 32-bit architectures. It aims at reducing the number of ISW multiplications by running two of them in parallel and therefore using full slices: one is done on the LSBs and the other one on the MSBs (see Figure 2a). As a result, because an encoding is not necessarily placed in parallel to the same other one before applying the ISW multiplication, the implementations we target are affected by algorithmic noise that can be averaged when m increases. Precisely, 16 bits of algorithm noise change when manipulating the same shares in different multiplications. This optimization is not present in the case of Clyde because it natively uses the 32-bit registers. The bits placed in parallel to a share remain the same across the multiplications. Therefore, the algorithmic noise is constant and so cannot be averaged. Only the limited physical noise can be averaged in this case. This observation suggests that the algorithmic noise has more impact compared to the physical one in our case studies.

Next, we study the effect of the **number of additions** a when the number of multiplications is set to zero. Regarding the AES, for all the additions applied to an encoding, the shares are always laid out according to Figure 2a. The same (8 or 24) bits are therefore always placed in parallel to the 8 bits profiled by our adversary. In this context, there is no algorithmic noise that can be averaged. The only noise that can be averaged by increasing a is physical since each share is loaded a times. On Figure 12, we observe that by increasing a, the PI only slightly increases compared to increasing m. Hence, we conclude again that there is limited physical noise to average.[11]

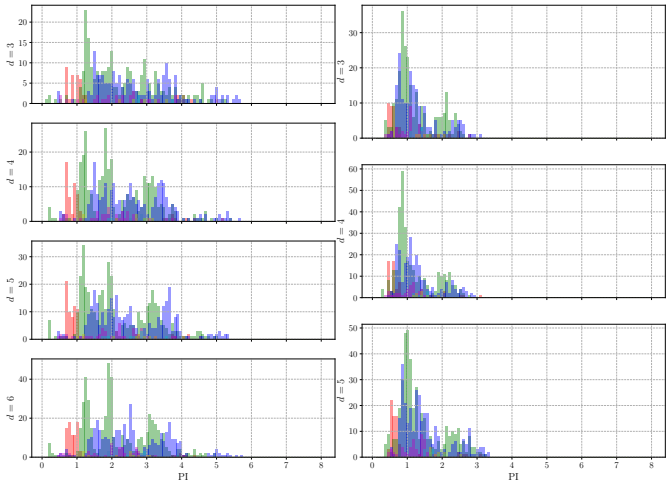
The impact of the **number of shares** d is also hinting towards the intuition that the physical noise of our implementations is very limited. Similarly to the number of additions,

[11] For Clyde, a is always equal one. Therefore we do not report its influence here.



(a) Cortex-M0, AES **(b)** Cortex-M3, AES **(c)** Cortex-M0, Clyde

Figure 11: Histograms of the PI on variable nodes for various masking orders and targets. The color code is the number of multiplications using a share m . for $m = 0$, for $m = 1$, for $m = 2$ and for $m = 3$. The continuous lines are the means and the dashed one the medians of the different distributions.



(a) Cortex-M0, AES. **(b)** Cortex-M3, AES.

Figure 12: Histograms of the PI on variable nodes for various masking orders and targets. The color code is the number of additions using a share a while not being used in a multiplication ($m = 0$). for $a = 0$, for $a = 1$, for $a = 2$ and for $a = 3$.

increasing the number of shares may allow the adversary to average the physical noise (due d manipulations of each share in a multiplication) but not the algorithmic noise (since the shares used within a multiplication are all affected by the same algorithmic noise). The fact that the PI values are marginally increasing with d suggests once again that there is limited physical noise to average. For the investigated platforms and masking orders, this falsifies the assumption that an increasing noise is required for the ISW multiplications when their number of shares increases [BCPZ16].

So overall for the investigated d values, we observe that the impact of algorithmic noise is much larger than the one of physical noise. This conclusion suggests that the use of multiplication gadgets with improved resistance against horizontal attacks such as [BCPZ16, CS19], which increase the amount of refreshing internally to the multiplications, is unlikely to improve security in our context. Indeed, it only allows to limit the physical noise averaging which is not dominant in our context. By contrast for the AES, adding refresh gadgets between the multiplications (in order to reduce m) has more potential as hinted in [GGSB20]. We investigate these directions in Section 7.

6 Adversaries with relaxed capabilities

In the above attacks, we assume an adversary in a worst-case setting, meaning that she has the full control and knowledge of the target implementation. In the following, we relax the adversary's capabilities in a backwards fashion [ABB+20]. Namely, we first study the increase in attack data complexity induced by profiling on one device and attacking another one. Then, we compare our attack to the existing attacks against Clyde, which does not exploit the full knowledge of the implementation and can possibly succeed with a less informed profiling phase (e.g. without randomness knowledge).

6.1 Inter-device profiling portability

In the following, we estimate the attack complexity of an adversary that builds the profiles on one device and attacks another device. More precisely, we are interested in the factor by which the attack complexity will be increased compared to the previous adversary, for which the profiling and attack devices are the same. To do so, we use the following equation:

$$\gamma_{i,j} = \mathbb{E}\left[\frac{\min(0, \mathrm{PI}_{j,i})}{\mathrm{PI}_{j,j}}\right], \tag{7}$$

where $\mathrm{PI}_{j,i}$ is the PI available when profiling a device j and attacking a device i.[12] The denominator corresponds to the worst-case where the profiling and attack are performed on the same device. The numerator corresponds to the relaxed adversary which has to train and attack different devices. The mathematical expectation is over multiple intermediate variables which in our case are shares in the encoding graphs. Based on the exponential trend with the available information from Equation (6), we extrapolate that the data complexity is also growing exponentially such that

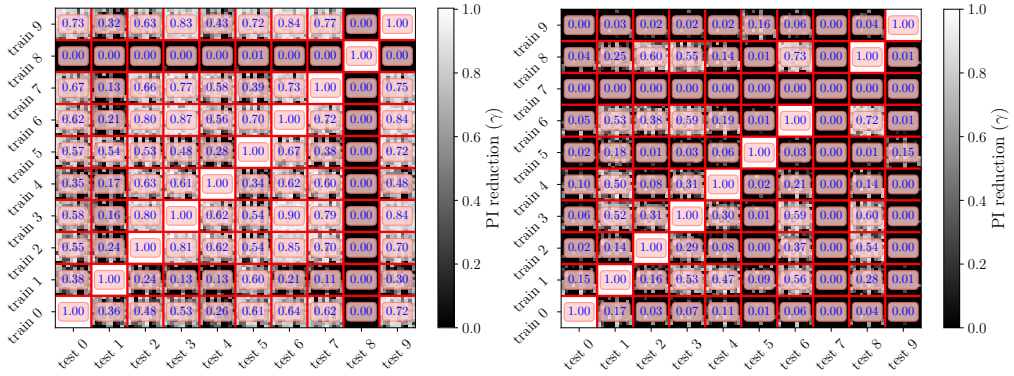
$$N_{j,i} \approx N_{j,j} \cdot \left(\frac{1}{\gamma_{j,i}}\right)^d, \tag{8}$$

where $N_{j,j}$ is the data complexity in the worst-case setting estimated in Subsection 5.2. The term $N_{j,i}$ is the data complexity of the relaxed adversary.

[12] When the adversary's model is too incorrect to extract information, the estimation of the PI can be negative and therefore we set it to zero with the min operator.

In order to quantitatively estimate the impact of cross-device profiling on the attacks in the worst-case setting, we reproduced the setup presented in Subsection 3.1 for 10 different Cortex-M0's and 10 different Cortex-M3's. For each of the MCU's, we ran the AES software with three shares. One dataset of 100, 000 traces is collected to estimate a model and another 10, 000 traces is used to estimate the PI values. A particular care has been taken to reduce the setup variations when collecting traces between the devices. Namely, we have built a daughter board that contains all the trigger and communication connections. In order to measure a new target, it only has to be plugged into the daughter board and the current probe has to be plugged into the current measurement jumper. Both the daughter board and the probe were maintained at the same relative position. Data collection was done in a lab with temperature control.

The obtained $\gamma_{j,i}$ values are reported in Figure 13.[13] For the Cortex-M0 on Figure 13a, we notice that most of the pairs (j, i) lead to significant information preservation, with most of them such that $\gamma_{j,i} \geq 0.5$. The pair minimizing the information loss is the pair $\gamma_{3,6} = 0.9$. Therefore, for an adversary profiling on the third device and attacking the sixth device, the increase on the data complexity – which is worth (0.9^d) – remains small even for large masking order. For example, in the case of an implementation with 8 shares, the loss factor is around 2. For the Cortex-M3 on Figure 13b, the portability of the templates is slightly worse. Yet, the best profiling pair $\gamma_{8,6} = 0.73$ only leads to a limited loss factor of 8 in data complexity, for an implementation with 8 shares.



(a) Cortex-M0 **(b)** Cortex-M3

Figure 13: Estimated PI loss factor ($\gamma_{j,i}$) for various training and testing devices. Estimation is done on 100 randomly selected intermediate variables.

Overall, the results from Figure 13 show that the portability of templates between two devices induce some information loss, confirming results such as [RSV+11, EG12], but that this loss is too limited to provide strong security guarantees. In an evaluation scenario, performing the profiling and the attack on the same device therefore allows to lower bound the complexity of the best attack, independent of this loss. This approach removes the risk of a false sense of security induced by an evaluation performed on a poorly portable profile / attack pair (which is easy to circumvent by profiling on a few devices, as performed in this section). Yet, finding the best way to attack a fresh device based on a pool of profiling devices is an interesting question that we leave for further research.

[13] The device 7 for Cortex-M3 was no more functional after the board modifications.

6.2 Reduced implementation knowledge

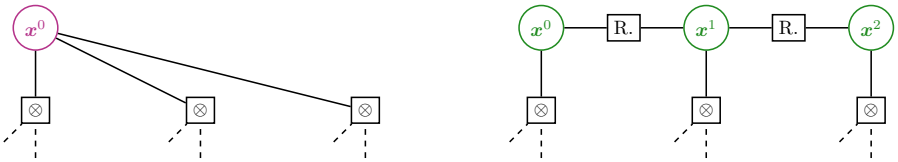
In order to discuss the impact of reduced implementation knowledge, we compare our attacks with the package TKMv3 which reports successful key recoveries against Clyde and is the winner of the CHES2020 CTF [BBC+20b]. While the CTF provides challengers with implementation details (and all the randomness for profiling), TKMv3 does not exploit these details and only targets the leakage of the shares at the output of the Sbox. Concretely, the TKMv3 adversary first evaluates the probabilities for each single shared individual bit. Then she recombines these probabilities in order to obtain a probability for the unshared bits at the output of the Sbox. Based on them, the adversary finally updates the 4-bit key guess probabilities by inverting the Sbox.

By comparing the two attacks' performances in Figure 8, we observe that SASCA gains interest as the number of shares increases. More precisely, for 3 shares TKMv3 performs better, for 4 shares the attacks are equivalent and for more shares, our attacks performs better. For example, they reduces the TKMv3 data complexity for the 8 shares target by a factor larger than 4. This improvement of our adversary over the TKMv3 one can be explained by her ability to efficiently profile the internal computations of the Sbox thanks the full implementation knowledge, which is enabled by SASCA. The improvement goes in two directions. First, the TKMv3 attack can only profile a limited number of bits within a register (concretely, only one bit is profiled), in order to be able to transfer the probabilities to an enumerable (four times larger) key guess. In the context of a software running on MCU's with limited physical noise, this does not allow reducing the dominant algorithmic noise by profiling more bits manipulated in parallel in the same register. By using SASCA, our adversary is able to do so and keeps the same guess size as the profiled variables (i.e. 8 bits). Second, our adversary is able to exploit the leakage on the internal variables which may also leak information (such as the input linear layer in the AES). As a counterpart of these gains, we note that the SASCA adversary transfers the information obtained from the Sbox computations in a heuristic manner (since it relies on BP) while TKMv3 does that part optimally. So there is a balance between the amount of information collected and the optimality of its treatment. In the investigated (CHES 2020 CTF) case study, the balance is in favor of our SASCA adversary.

Overall, this discussion highlights that an adversary that is relaxed in the sense that she does not have access to precise implementation details can reach similar complexities as a worst-case adversary. This provides a case for using SASCA as an evaluation tool leading to good estimates of the worst-case security level at limited profiling cost (and enabling extrapolations thanks to models such as [GGSB20], as will be illustrated next). It also suggests the combination of the powerful information extraction that deep learning enables with analytical strategies as an interesting open problem.

6.3 Randomness-free profiling

Finally, we note that another solution to evaluate/attack an implementation in a black box manner is to use moment-based detection/distinguishers. In this case, not only the implementation details are not needed but the model profiling can be performed without randomness knowledge. It is already known that applying leakage detection to software masked implementation becomes rapidly unpractical with large number of shares in that setting [BSS19]. Yet, we briefly study this option in Appendix D Figure 17, where we report the output of a Test Vector Leakage Assessment (TVLA) searching to second order leakages in a third-oder implementation [GJJR11, CMG+]. The latter shows that the independence assumption needed for masking is not completely fulfilled. But exploiting these lower-order leakages requires more than 1,000 traces while our attack succeeds in 2 traces on the same target. This confirms that in the context of low noise software that



(a) Original Construction. **(b)** Reduced Fan Out.

Figure 14: refreshing strategy to reduce the m parameter.

we consider, analytical attacks directly targeting the low noise of the implementation are more powerful than divide-and-conquer attacks targeting a reduction of the security order.

Whether the efficiency of our analytical attacks can be approached with a less informed profiling is an interesting direction for further investigation, and the same holds for attacks exploiting information leakages at different security orders.

7 Extrapolated countermeasures impact

In the previous sections, we have highlighted that our target implementations require a large number of shares to resist worst-case analytical adversaries. We stressed that this weakness is partially imputable to the low physical noise and possibility to average algorithmic noise that these targets offer. In this section, we analyze some options to improve the side-channel security of such devices. Namely, we evaluate a countermeasure that avoids the algorithmic noise averaging for the AES implementation and then quantify the impact of physical noise addition for both the Clyde and the AES implementations.

7.1 Reduced multiplicative manipulations through refresh gadgets

In order to avoid algorithmic noise to be averaged for encodings used in multiple ISW multiplications of the AES implementation, one has to avoid using the same encoding with uncorrelated algorithmic noise multiple times, as shown in Figure 11. To do so, a solution is to add simple refresh gadgets (i.e. additions with encodings of zero) so that any encoding is placed at the input of a single multiplication.[14] Informally, this still allows the adversary to observe multiple encodings of the same sensitive variable, but not anymore to average the noise on the shares. Hence, she can observe multiple times the noise amplified thanks to masking instead of a single time a reduced amplification of the averaged noise. Informally, such a solution should shift the blue ($m = 2$) and purple ($m = 3$) distributions in Figure 11 to the green one ($m = 1$). The refreshing strategy we used in our analyzes is described in Figure 14 and was applied to all the nodes with $m > 1$.

In order to rapidly assess the impact of this proposal, we use the Local Random Probing Model (LRPM) which is an information theoretic model that can efficiently bound the efficiency of SASCA by analyzing the information propagation within a factor graph [GGSB20]. For an encoding graph, the information that is available on a secret variable is proportional to the product of the normalized information (λ) on each of the shares, leading to λ^d. For example, the five-share implementation on the Cortex-M3 with $m = 3$, leads to a normalized information of approximately $(\frac{6}{8})^5 = 0.23$ (the PI value of 6 is taken from Figure 11). When inserting the refresh gadgets, the adversary rather gets three times an encoding with $m = 1$. Hence, she can accumulate the leakage on each of these three observations and recover a total of $3 \cdot (\frac{2.5}{8})^5 = 0.008$ bits of normalized

[14] Since the circuit is already probing secure without the refresh gadgets, these additional refresh gadgets do not require strong composability properties like strong non-interference.

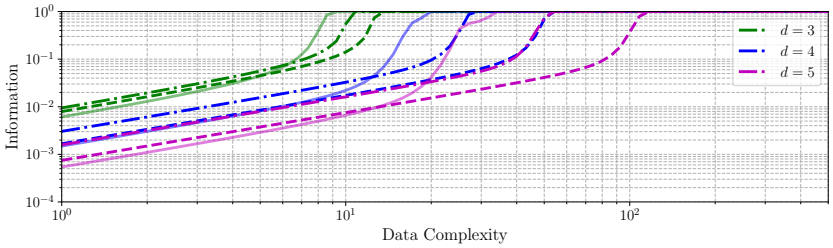


Figure 15: LRPM analysis of the Cortex-M3. − for actual PI at each node, −· for mean PI according to m and −− for mean PI with additional refresh gadgets inserted.

information. So inserting the refresh gadgets should reduce the amount of information leaked for some operations (even more when the number of shares increases). The main question in this respect is whether this solution is sufficient to compensate the impact of all the (e.g. linear) operations that are not affected by such a refreshing strategy.

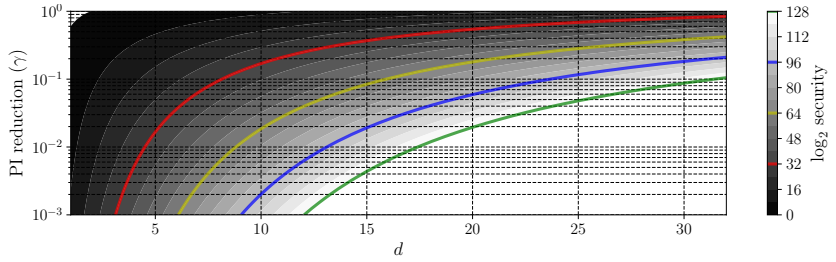
In order to answer this question, we applied the LRPM to the entire factor graph with the PI values extracted from the Cortex-M3 device. The results are reported in Figure 15. The continuous lines are the representations of the previously performed attacks in the LRPM model. The dashed dotted lines are performed on the same factor graph but the actual PI values are replaced by the mean of their distribution, depending on the m of the encoding. Dashed lines represent the modified factor graph with additional refresh gadgets. As expected, by comparing the two factor graphs, the modified one requires more data to reach a full key leakage. Furthermore, the gain that the additional refresh gadgets bring grows with the masking order. However, in practice, even for $d = 5$ this gain remains small with a factor around 2. We posit that this limited impact is due to the fact that secret variables with $m > 1$ are a minority of the variables in the entire graph (in the bitslice Sbox studied, many nodes have a $m = 0$ because of linear layers). Besides, we note that for other circuits with fewer $m = 0$ operations the gain of this circuit modification should be to be more significant (e.g. lightweight ciphers like Clyde).

7.2 Information reduction countermeasures

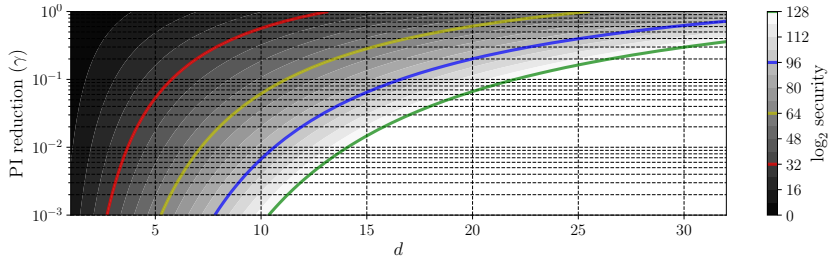
The main conclusion of our previous experiments is that the lack of physical noise in low-end MCU's can make the exponential security increase that masking provides pretty slow in d. Admittedly, this conclusion is in part related to the fact that we analyze masking as a stand-along countermeasure, while it could potentially be combined with other countermeasures. We recall that our goal in this paper is to provide tools for the analysis of the noise level needed to implement masking, for example to support the NIST standardization of masking schemes effort. So the concrete evaluation of such combinations of countermeasures is left as an open problem.

Yet, and as a first step in the directions of designing software implementations with higher security at lower cost, we next extrapolate the impact of combining masking with a (generic) countermeasure reducing the information available on each share. Namely, for a given information reduction factor γ, we report the estimated attack data complexity based on Equation (8). We note that this extrapolation is independent of the method used to decrease the information since the effect of any countermeasure can be quantified with information theoretic metrics. Among others, it covers the possibility for an implementer to reduce the exploitable signal at the software level by using custom encodings [CESY14, MSB16, BJP17] or at the gate level by using custom logic [TV03, TV04]. It also covers the case where noise is increased at the technological level [LBBS20] or emulated at the algorithmic level by using random delays [CK09, CK10] or shuffling [HOM06, VMKS12].

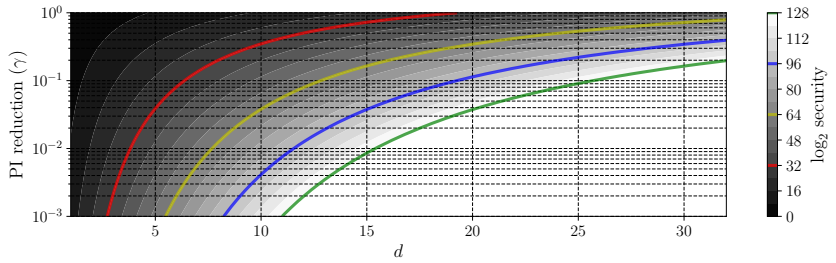
It finally covers the case of a less efficient measurement setup used by the adversary.



(a) Cortex-M0, AES



(b) Cortex-M3, AES



(c) Cortex-M0, Clyde

Figure 16: Extrapolated attack data complexity with 2^{32} enumeration power according to the information reduction on a single share compared to our adversary.

For example, Figure 16 shows that by reducing the information per share by a factor 10 (which corresponds to $\gamma = 0.1$), the AES and the Clyde implementations on the Cortex-M0 still require 16 shares to resist key recovery with up to 2^{64} data complexity. For the AES on the Cortex-M3, 11 shares are needed to have the same security guarantees. If the countermeasure is able to reduce the information per share by a factor 100, 8 shares are still needed on the Cortex-M0 and 7 on Cortex-M3. As in the previous section, Clyde is showing a slightly better trend and for all the implementations with a $\gamma = 10^{-3}$, 6 shares are assumed to provide high enough security. While admittedly abstract, these extrapolations can be used in order to set a clear target for the design of software hiding countermeasures to be used in combination with bitslice masking.

8 Conclusions & open problems

The results in this work show that securing the two investigated 32-bit embedded devices against side-channel analysis thanks to masking is highly challenging. In case clean measurement setups can be built, the amount of physical noise that off-the-shelf ARM Cortex-M0 or M3 devices provide is small and hardly leads to secure implementations unless very high number of shares are used, leading to large (possibly prohibitive) performance

overheads. Extrapolations of the security guarantees offered by the addition of information reduction countermeasures show how much this number of shares could be reduced with less informative leakage. It is therefore an important problem to study how to ensure these lower information leakages in practice and to determine what is the best combination of countermeasures to be used on low-cost MCUs. We believe the tools described in this work are good candidates to analyze such combinations of countermeasures.

We note that the cost of protected primitives on such platforms is pushing for a limited number of calls to them. Leakage-resistant modes of operation enabling leveled implementations, where the number of calls to the protected primitive is independent of the message length and the bulk of the computation does not require strong protection [BBC+20a], therefore appear as interesting solutions to amortize the cost of masking in software.

Eventually, and from a security analysis viewpoint, it also remains an open question to analyse qualitatively the impact of removing completely the randomness knowledge during the profiling phase. Such a knowledge is critical in our and the TKMv3 attacks and it would be interesting to study whether better options exist than the expensive (moment-based) higher-order side-channel attacks that we briefly discussed in Subsection 6.3.

Acknowledgments

The authors thank Gaëtan Cassiers, co-author of SCALib, for its time spent in improving the leakage analysis tools. François-Xavier Standaert is senior research associate of the Belgian Fund for Scientific Research (F.R.S.-FNRS). This work has been funded by the EU through the ERC project SWORD (724725).

References

[ABB+20] Melissa Azouaoui, Davide Bellizia, Ileana Buhan, Nicolas Debande, Sébastien Duval, Christophe Giraud, Éliane Jaulmes, François Koeune, Elisabeth Oswald, François-Xavier Standaert, and Carolyn Whitnall. A systematic appraisal of side channel evaluation strategies. In *SSR*, volume 12529 of *Lecture Notes in Computer Science*, pages 46–66. Springer, 2020.

[APSQ06] Cédric Archambeau, Eric Peeters, François-Xavier Standaert, and Jean-Jacques Quisquater. Template attacks in principal subspaces. In *CHES*, volume 4249 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2006.

[BBB+20] Davide Bellizia, Francesco Berti, Olivier Bronchain, Gaëtan Cassiers, Sébastien Duval, Chun Guo, Gregor Leander, Gaëtan Leurent, Itamar Levi, Charles Momin, Olivier Pereira, Thomas Peters, François-Xavier Standaert, Balazs Udvarhelyi, and Friedrich Wiemer. Spook: Sponge-based leakage-resistant authenticated encryption with a masked tweakable block cipher. *IACR Trans. Symmetric Cryptol.*, 2020(S1):295–349, 2020.

[BBC+20a] Davide Bellizia, Olivier Bronchain, Gaëtan Cassiers, Vincent Grosso, Chun Guo, Charles Momin, Olivier Pereira, Thomas Peters, and François-Xavier Standaert. Mode-level vs. implementation-level physical security in symmetric cryptography - A practical guide through the leakage-resistance jungle. In *CRYPTO (1)*, volume 12170 of *Lecture Notes in Computer Science*, pages 369–400. Springer, 2020.

[BBC+20b] Davide Bellizia, Olivier Bronchain, Gaetan Cassiers, Charles Momin, François-Xavier Standaert, and Balazs Udvarhelyi. *Spook CTF (CHES2020)*, 2020.

[BCPZ16] Alberto Battistello, Jean-Sébastien Coron, Emmanuel Prouff, and Rina Zeitoun. Horizontal side-channel attacks and countermeasures on the ISW masking scheme. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 23–39. Springer, 2016.

[BDM+20] Sonia Belaïd, Pierre-Évariste Dagand, Darius Mercadier, Matthieu Rivain, and Raphaël Wintersdorff. Tornado: Automatic generation of probing-secure masked bitsliced implementations. In *EUROCRYPT (3)*, volume 12107 of *Lecture Notes in Computer Science*, pages 311–341. Springer, 2020.

[BGG+14] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 64–81. Springer, 2014.

[BGNT18] Nicolas Bruneau, Sylvain Guilley, Zakaria Najm, and Yannick Teglia. Multivariate high-order attacks of shuffled tables recomputation. *J. Cryptology*, 31(2):351–393, 2018.

[BGR18] Sonia Belaïd, Dahmun Goudarzi, and Matthieu Rivain. Tight private circuits: Achieving probing security with the least refreshing. In *ASIACRYPT (2)*, volume 11273 of *Lecture Notes in Computer Science*, pages 343–372. Springer, 2018.

[BHM+19] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 713–737. Springer, 2019.

[BJP17] Shivam Bhasin, Dirmanto Jap, and Thomas Peyrin. Practical evaluation of FSE 2016 customized encoding countermeasure. *IACR Trans. Symmetric Cryptol.*, 2017(3):108–129, 2017.

[BKPT] Ryad Benadjila, Louiza Khati, Emmanuel Prouff, and Adrian Thillard. https://github.com/ANSSI-FR/SecAESSTM32.

[BS20] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):1–25, 2020.

[BSS19] Olivier Bronchain, Tobias Schneider, and François-Xavier Standaert. Multi-tuple leakage detection and the dependent signal issue. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):318–345, 2019.

[CESY14] Cong Chen, Thomas Eisenbarth, Aria Shahverdi, and Xin Ye. Balanced encoding to mitigate power analysis: A case study. In *CARDIS*, volume 8968 of *Lecture Notes in Computer Science*, pages 49–63. Springer, 2014.

[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 *COSADE*, volume 7275 of *Lecture Notes in Computer Science*, pages 69–81. Springer, 2012.

[CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 156–170. Springer, 2009.

[CK10] Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of CHES 2009. In *CHES*, volume 6225 of *Lecture Notes in Computer Science*, pages 95–109. Springer, 2010.

[CMG+] Jeremy Cooper, Elke De Mulder, Gilbert Goodwill, Josh Jaffe, Gary Kenworthy, and Pankaj Rohatgi. Test vector leakage assessment (TVLA) methodology in practice (extended abstract). ICMC 2013. http://icmc-2013.org/wp/wp-content/uploads/2013/09/goodwillkenworthtestvector.pdf.

[CRB+16] Thomas De Cnudde, Oscar Reparaz, Begül Bilgin, Svetla Nikova, Ventzislav Nikov, and Vincent Rijmen. Masking AES with d+1 shares in hardware. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 194–212. Springer, 2016.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *CHES*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[CS19] Gaetan Cassiers and François-Xavier Standaert. Towards globally optimized masking: From low randomness to low noise rate or probe isolating multiplications with reduced randomness and security against horizontal attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):162–198, 2019.

[DDF14] Alexandre Duc, Stefan Dziembowski, and Sebastian Faust. Unifying leakage models: From probing attacks to noisy leakage. In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, 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 (1)*, volume 9056 of *Lecture Notes in Computer Science*, pages 401–429. Springer, 2015.

[EG12] M. Abdelaziz Elaabid and Sylvain Guilley. Portability of templates. *J. Cryptogr. Eng.*, 2(1):63–74, 2012.

[GGSB20] Qian Guo, Vincent Grosso, François-Xavier Standaert, and Olivier Bronchain. Modeling soft analytical side-channel attacks from a coding theory viewpoint. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):209–238, 2020.

[GJJR11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side channel resistance validation. NIST non-invasive attack testing workshop, 2011. http://csrc.nist.gov/news_events/non-invasive-attack-testing-workshop/papers/08_Goodwill.pdf.

[GMK17] Hannes Groß, Stefan Mangard, and Thomas Korak. An efficient side-channel protected AES implementation with arbitrary protection order. In *CT-RSA*, volume 10159 of *Lecture Notes in Computer Science*, pages 95–112. Springer, 2017.

[GMPO20] Si Gao, Ben Marshall, Dan Page, and Elisabeth Oswald. Share-slicing: Friend or foe? *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):152–174, 2020.

[GPSS18] Benjamin Grégoire, Kostas Papagiannopoulos, Peter Schwabe, and Ko Stoffelen. Vectorizing higher-order masking. In *COSADE*, volume 10815 of *Lecture Notes in Computer Science*, pages 23–43. Springer, 2018.

[GR17] Dahmun Goudarzi and Matthieu Rivain. How fast can higher-order masking be in software? In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 567–597, 2017.

[GRO18] Joey Green, Arnab Roy, and Elisabeth Oswald. A systematic study of the impact of graphical models on inference-based attacks on AES. In *CARDIS*, volume 11389 of *Lecture Notes in Computer Science*, pages 18–34. Springer, 2018.

[GS15] Vincent Grosso and François-Xavier Standaert. Asca, SASCA and DPA with enumeration: Which one beats the other and when? In *ASIACRYPT (2)*, volume 9453 of *Lecture Notes in Computer Science*, pages 291–312. Springer, 2015.

[GS18] Vincent Grosso and François-Xavier Standaert. Masking proofs are tight and how to exploit it in security evaluations. In *EUROCRYPT (2)*, volume 10821 of *Lecture Notes in Computer Science*, pages 385–412. Springer, 2018.

[HOM06] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An AES smart card implementation resistant to power analysis attacks. In *ACNS*, volume 3989 of *Lecture Notes in Computer Science*, pages 239–252, 2006.

[ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private circuits: Securing hardware against probing attacks. In *CRYPTO*, volume 2729 of *Lecture Notes in Computer Science*, pages 463–481. Springer, 2003.

[JS17] Anthony Journault and François-Xavier Standaert. Very high order masking: Efficient implementation and security evaluation. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 623–643. Springer, 2017.

[KPP20] Matthias J. Kannwischer, Peter Pessl, and Robert Primas. Single-trace attacks on keccak. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):243–268, 2020.

[LBBS20] Itamar Levi, Davide Bellizia, David Bol, and François-Xavier Standaert. Ask less, get more: Side-channel signal hiding, revisited. *IEEE Trans. Circuits Syst.*, 67-I(12):4904–4917, 2020.

[LD16] Yi Lu and Yvo Desmedt. Walsh-hadamard transform and cryptographic applications in bias computing. *IACR Cryptol. ePrint Arch.*, 2016:419, 2016.

[Man04] Stefan Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In *CT-RSA*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

[MPG05] Stefan Mangard, Thomas Popp, and Berndt M. Gammel. Side-channel leakage of masked CMOS gates. In *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 351–365. Springer, 2005.

[MPL+11] Amir Moradi, Axel Poschmann, San Ling, Christof Paar, and Huaxiong Wang. Pushing the limits: A very compact and a threshold implementation of AES. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 69–88. Springer, 2011.

[MSB16] Houssem Maghrebi, Victor Servant, and Julien Bringer. There is wisdom in harnessing the strengths of your enemy: Customized encoding to thwart side-channel attacks. In *FSE*, volume 9783 of *Lecture Notes in Computer Science*, pages 223–243. Springer, 2016.

[NRS11] Svetla Nikova, Vincent Rijmen, and Martin Schläffer. Secure hardware implementation of nonlinear functions in the presence of glitches. *J. Cryptology*, 24(2):292–321, 2011.

[PR13] Emmanuel Prouff and Matthieu Rivain. Masking against side-channel attacks: A formal security proof. In *EUROCRYPT*, volume 7881 of *Lecture Notes in Computer Science*, pages 142–159. Springer, 2013.

[PSG16] Romain Poussier, François-Xavier Standaert, and Vincent Grosso. Simple key enumeration (and rank estimation) using histograms: An integrated approach. In *CHES*, volume 9813 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2016.

[RPM19] Sebastian Renner, Enrico Pozzobon, and Jurgen Mottok. Benchmarking software implementations of 1st round candidates of the NIST LWC project on microcontrollers. *Lightweight Cryptography Workshop*, 2019.

[RSV+11] Mathieu Renauld, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Dina Kamel, and Denis Flandre. A formal study of power variability issues and side-channel attacks for nanoscale devices. In *EUROCRYPT*, volume 6632 of *Lecture Notes in Computer Science*, pages 109–128. Springer, 2011.

[SA08] François-Xavier Standaert and Cédric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In *CHES*, volume 5154 of *Lecture Notes in Computer Science*, pages 411–425. Springer, 2008.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *EUROCRYPT*, volume 5479 of *Lecture Notes in Computer Science*, pages 443–461. Springer, 2009.

[Sta18] François-Xavier Standaert. How (not) to use welch's t-test in side-channel security evaluations. In *CARDIS*, volume 11389 of *Lecture Notes in Computer Science*, pages 65–79. Springer, 2018.

[TV03] Kris Tiri and Ingrid Verbauwhede. Securing encryption algorithms against DPA at the logic level: Next generation smart card technology. In *CHES*, volume 2779 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2003.

[TV04] Kris Tiri and Ingrid Verbauwhede. A logic level design methodology for a secure DPA resistant ASIC or FPGA implementation. In *DATE*, pages 246–251. IEEE Computer Society, 2004.

[TWO13] Michael Tunstall, Carolyn Whitnall, and Elisabeth Oswald. Masking tables - an underestimated security risk. In *FSE*, volume 8424 of *Lecture Notes in Computer Science*, pages 425–444. Springer, 2013.

[VGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *ASIACRYPT (1)*, volume 8873 of *Lecture Notes in Computer Science*, pages 282–296. Springer, 2014.

[VMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *ASIACRYPT*, volume 7658 of *Lecture Notes in Computer Science*, pages 740–757. Springer, 2012.

A Masked gadgets

Algorithm 1 Masked XOR gate.

Input: $\boldsymbol{a}$ with $a = \sum_{i=0}^{d-1} a_i$ and $\boldsymbol{b}$ with $b = \sum_{i=0}^{d-1} b_i$.
Output: $\boldsymbol{c}$ with $c = \sum_{i=0}^{d-1} c_i$ and $c = a \oplus b$.

for $i \in [0, \ldots, d-1]$ **do**
  $c_i \leftarrow a_i \oplus b_i$

Algorithm 2 ISW multiplication.

Input: $\boldsymbol{a}$ with $a = \sum_{i=0}^{d-1} a_i$ and $\boldsymbol{b}$ with $b = \sum_{i=0}^{d-1} b_i$.
Output: $\boldsymbol{c}$ with $c = \sum_{i=0}^{d-1} c_i$ and $c = a \otimes b$.

for $i \in [0, \ldots, d-1]$ **do**
  $c_i \leftarrow a_i \otimes b_i$
for $i \in [0, \ldots, d-1]$ **do**
  for $j \in [i+1, \ldots, d-1]$ **do**
    $s \leftarrow \{0, 1\}$
    $c_i \leftarrow c_i \oplus s$
    $s' \leftarrow (s \oplus (a_i \otimes b_j)) \oplus (a_j \otimes b_i)$
    $c_j \leftarrow c_j \oplus s'$

B AES Sbox circuit

Table 1: Top linear layer

$y14 = x3 \oplus x5$	$y1 = t0 \oplus x7$	$y15 = t1 \oplus x5$	$y17 = y10 \oplus y11$
$y13 = x0 \oplus x6$	$y4 = y1 \oplus x3$	$y20 = t1 \oplus x1$	$y19 = y10 \oplus y8$
$y12 = y13 \oplus y14$	$y2 = y1 \oplus x0$	$y6 = y15 \oplus x7$	$y16 = t0 \oplus y11$
$y9 = x0 \oplus x3$	$y5 = y1 \oplus x6$	$y10 = y15 \oplus t0$	$y21 = y13 \oplus y16$
$y8 = x0 \oplus x5$	$t1 = x4 \oplus y12$	$y11 = y20 \oplus y9$	$y18 = x0 \oplus y16$
$t0 = x1 \oplus x2$	$y3 = y5 \oplus y8$	$y7 = x7 \oplus y11$	

Table 2: Middle non-linear layer

$t2 = y12 \otimes y15$	$t23 = t19 \oplus y21$	$t34 = t23 \oplus t33$	$z2 = t33 \otimes x7$
$t3 = y3 \otimes y6$	$t15 = y8 \otimes y10$	$t35 = t27 \oplus t33$	$z3 = t43 \otimes y16$
$t5 = y4 \otimes x7$	$t26 = t21 \otimes t23$	$t42 = t29 \oplus t33$	$z4 = t40 \otimes y1$
$t7 = y13 \otimes y16$	$t16 = t15 \oplus t12$	$z14 = t29 \otimes y2$	$z6 = t42 \otimes y11$
$t8 = y5 \otimes y1$	$t18 = t6 \oplus t16$	$t36 = t24 \otimes t35$	$z7 = t45 \otimes y17$
$t10 = y2 \otimes y7$	$t20 = t11 \oplus t16$	$t37 = t36 \oplus t34$	$z8 = t41 \otimes y10$
$t12 = y9 \otimes y11$	$t24 = t20 \oplus y18$	$t38 = t27 \oplus t36$	$z9 = t44 \otimes y12$
$t13 = y14 \otimes y17$	$t30 = t23 \oplus t24$	$t39 = t29 \otimes t38$	$z10 = t37 \otimes y3$
$t4 = t3 \oplus t2$	$t22 = t18 \oplus y19$	$z5 = t29 \otimes y7$	$z11 = t33 \otimes y4$
$t6 = t5 \oplus t2$	$t25 = t21 \oplus t22$	$t44 = t33 \oplus t37$	$z12 = t43 \otimes y13$
$t9 = t8 \oplus t7$	$t27 = t24 \oplus t26$	$t40 = t25 \oplus t39$	$z13 = t40 \otimes y5$
$t11 = t10 \oplus t7$	$t31 = t22 \oplus t26$	$t41 = t40 \oplus t37$	$z15 = t42 \otimes y9$
$t14 = t13 \oplus t12$	$t28 = t25 \otimes t27$	$t43 = t29 \oplus t40$	$z16 = t45 \otimes y14$
$t17 = t4 \oplus t14$	$t32 = t31 \otimes t30$	$t45 = t42 \oplus tt41$	$z17 = t41 \otimes y8$
$t19 = t9 \oplus t14$	$t29 = t28 \oplus t22$	$z0 = t44 \otimes y15$	
$t21 = t17 \oplus y20$	$t33 = t32 \oplus t24$	$z1 = t37 \otimes y6$	

Table 3: Bottom linear layer

$t46 = z15 \oplus z16$	$t49 = z9 \oplus z10$	$t61 = z14 \oplus t57$	$t48 = z5 \oplus z13$
$t55 = z16 \oplus z17$	$t63 = t49 \oplus t58$	$t65 = t61 \oplus t62$	$t56 = z12 \oplus t48$
$t52 = z7 \oplus z8$	$t66 = z1 \oplus t63$	$s0 = t59 \oplus t63$	$s3 = t53 \oplus t66$
$t54 = z6 \oplus z7$	$t62 = t52 \oplus t58$	$t51 = z2 \oplus z5$	$s1 = \overline{t64 \oplus s3}$
$t58 = z4 \oplus t46$	$t53 = z0 \oplus z3$	$s4 = t51 \oplus t66$	$s6 = \overline{t56 \oplus t62}$
$t59 = z3 \oplus t54$	$t50 = z2 \oplus z12$	$s5 = t47 \oplus t65$	$s7 = \overline{t48 \oplus t60}$
$t64 = z4 \oplus t59$	$t57 = t50 \oplus t53$	$t67 = t64 \oplus t65$	
$t47 = z10 \oplus z11$	$t60 = t46 \oplus t57$	$s2 = \overline{t55 \oplus t67}$	

C Clyde Sbox circuit

Table 4: Clyde Sbox

$tmp0 = x0 \otimes x1$	$tmp1 = x3 \otimes x0$	$tmp2 = y1 \otimes x3$	$tmp3 = y0 \otimes y1$
$y1 = tmp0 \oplus x2$	$y0 = tmp1 \oplus x1$	$y3 = tmp2 \oplus x0$	$y2 = tmp3 \oplus x3$

D Moment-based leakage detection

We report in Figure 17 the results of a TVLA searching for second-order flaws in the ISW multiplications of our AES three-share target. We observe that such a flaw is detectable with about 1,000 traces. However, since the noise is low, directly performing a third-order attack leads to attacks that succeed in much lower complexity. This result is a concrete counterpart of the discussion in [Sta18], which showed that as the number of shares in an implementation increases with too limited noise, estimating the higher-order statistical moments of the leakage distribution becomes an increasingly suboptimal strategy.

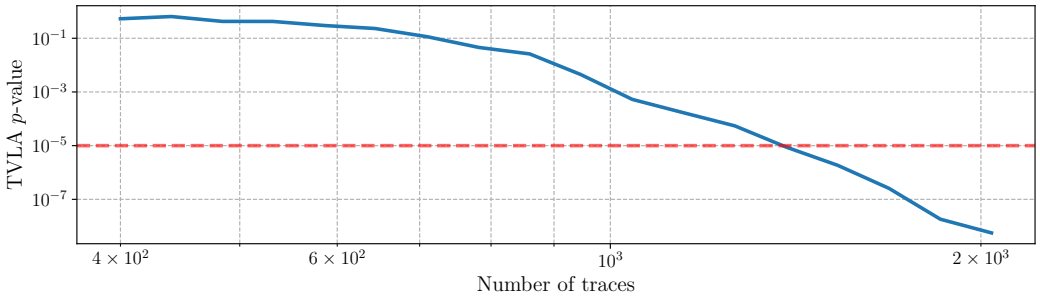


Figure 17: Multivariate second-order TVLA evaluation for the Cortex-M0 with $d = 3$ shares. The p-value is adapted in function of the number of sample within the traces.