Worst-case side-channel security: from evaluation of countermeasures to new designs

Olivier Bronchain

January 2022

ICTEAM
Louvain School of Engineering
Université catholique de Louvain
Louvain-la-Neuve
Belgium

Thesis Committee:

Pr. Standaert F.-X.	UCLouvain/ICTEAM, Belgium
Pr. Peters T.	UCLouvain/ICTEAM, Belgium
Pr. Bol D.	UCLouvain/ICTEAM, Belgium
Dr. Schneider T.	NXP Semiconductors, Austria
Dr. Grosso V.	CNRS, France
Dr. Prouff E.	Sorbonne Université, France
Pr. Oswald E.	Alpen-Adria-Universität Klagenfurt, Austria

Worst-case side-channel security: from evaluation of countermeasures to new designs

by Olivier Bronchain

ICTEAM
UCLouvain
Place Sainte-Barbe, 2
1348 Louvain-la-Neuve
Belgium

This work has been funded in parts by the European Union (EU) through the ERC project 724725 (acronym SWORD).

Acknowledgments

This thesis is the result of 4 wonderful years of research. I would like to thank all the ones who where next to me during these years.

Thanks to François-Xavier Standaert. First by introducing me to the field, pushing me to do meaningful and rigorous research. Then by being confident in my ability to conduct research projects from an initial idea to a research paper.

I would like to thank the jury members for reading the thesis and the discussions that followed during the private defense. Thanks to Tobias Schneider for shepherding me during my first year as a PhD student. Thanks to Vincent Grosso for collaborations and discussions. Thanks to Emmanuel Prouff for the constructive interactions when we discovered attacks against ANSSI's implementation. Thanks to Thomas Peters for being both a co-author of one of my favorite papers and being the secretary of the jury. Thanks to Elisabeth Oswald for feedbacks when I presented my projects to the REASSURE consortium. Thanks to David Bol for being the president of the jury.

I thank all of my colleagues. Among others, I would like to thank Gaëtan Cassiers for always having interesting inputs into my works. Thanks for jumping into the SCALib project and making a success of it. Thanks to Charles Momin for our numerous collaborations and taking care of Alice. Thanks to Balazs Udvarhelyi for its support with the measurement setups and our collaborations.

Thanks to my friends, family and wife. You endorsed listening to my stories about cryptography, the related community, publication process and new attack vectors. I guess you deserve some beers and / or coffees charged on me.

Abstract

Modern cryptography relies on the Kerckhoff's principle which states that everything about a crypto-system should be public, except for the secret keys. In this thesis, I apply these principles to cryptography implementations by providing the evaluator full knowledge of the implementation, in a worst-case manner. As a result, it allows to estimate their long-term security against side-channel attacks.

The first part is dedicated to the verification of the two underlying assumptions in masking proofs. More precisely, I contribute to proof-based evaluations by putting forward that *i)* the verification of independence thanks to leakage detection benefits from multivariate statistics *ii)* the noise can be estimated in a quantitative manner thanks to formal bounds on the mutual information.

The second part of the thesis is dedicated to efficient methodologies to perform worst-case attacks which are a useful shortcut for evaluators since they allow to estimate, with a reduce profiling data complexity, the online complexity other attacks strategies. Concretely, I propose worst-case attacks against a recent open-source AES implementation by the French ANSSI and masked higher-order bitslice implementations.

From the lessons learned from the side-channel evaluation of current proposals, I propose new designs to obtain strong side-channel protection. More precisely, the noise versus security trade-off can be improved thanks to a sound combination of masking and shuffling. Concretely, I apply masking and shuffling to ISW multiplications in order to exponentially amplify with masking the effect of shuffling.

Contents

Introduction 1

Building blocks of symmetric cryptography are abstract objects, next called *primitives*, such as block-ciphers or permutations. These objects have specific properties in the *black-box model* where an adversary can only observe and control the input and/or the output of the primitive. Over the years, the research community has gained confidence that some constructions are indeed good instantiations of these abstract primitives. This success has been reached thanks to strong adversarial model as well as public specifications which encourage external cryptanalysis. This methodology reflects the Kerckhoff's principle that is summarized with:

Everything about a crypto-system should be public except for the secret keys.

Even tough this state of affair is satisfying in the black-box model, cryptography must be implemented carefully. Indeed, implementations of cryptographic primitives are vulnerable to *side-channel attacks* (SCA) who exploit physical signals generated by physical components such as electronic circuits or computers, to efficiently recover cryptographic secrets. More precisely, the adversary can infer from these signals (noisy) intermediate states for which the cryptographic guarantees are not fully met. Such vulnerabilities are at least known since the 90's and exploit leakages such as timing [Koc96], power consumption [KJJ99] or electromagnetic emanations [QS01]. As a result, a cryptographic implementation should also be secure in the *gray-box model*. While a worst-case approach has been widely adopted in the black-box model, it is not yet in the gray-box model. In this thesis, I will focus on the evaluation and the protection of implementations by considering worst-case side-channel adversaries.

1.1 Current design and evaluation of secure implementations

In order to secure implementations against SCA, designers leverage countermeasures. These are defense mechanism that makes key recovery by side-channel adversaries harder. For a designer, the goal is to design

and parametrise these countermeasures to avoid successful attacks who exploit a fixed number of measurements N (data complexity) while minimizing the performance overheads. For an evaluator, the goal is to assess that no attack can succeed with less than N measurements, confirming the expectation of the designer.

Design of secure implementations. One prominent countermeasure, that I will discuss the most in this thesis, is *masking*. The principle of masking is to *compile* or *transform* an unprotected (Boolean) circuit into a masked circuit. To do so, all the wires within the unprotected circuit carrying the data x are replaced by d wires containing random shares x_i with the constraint that $x = \sum_{i=0}^{d-1} x_i$ [CJRR99]. Each of the logic gates are replaced by the corresponding masked gadget that performs the same operation but taking shares as input [ISW03].

In order to build a secure implementation, designers can prove the security of masked circuit in various models. These models consider adversaries with different capabilities being either realistic or abstract. In the (abstract) *probing model*, the adversary is allowed to place $d-1$ *probes* on any wire within the masked circuit to obtain their value during the circuit execution [ISW03]. In the (closer to standard SCA settings) *noisy leakage model*, the adversary obtains noisy version of all the shares [PR13].

One interest of these various models is their connection to each other through reductions [DDF14, DFS15, PGMP19]. Based on these, a designer can reason first in an abstract model for which proofs of security can be efficiently generalized from single gadgets to entire circuits thanks to composition [BGR18, BDM+20, CFOS21]. By doing so, he has the guarantee that the resulting implementations will be secure in the noisy leakage model if some hypotheses are fulfilled. Indeed, these reductions model physical implementations. Hence, they require two hypotheses on their physical properties:

1. **The independence** assumption requires that the leakage observed by the adversary is a linear combination of a noisy function of the shares. If not fulfilled, the masking order d can be reduced leading to a reduced security guarantee [MPL+11, CRB+16, GMK17].

2. **The noise** assumption requires that the noise on each of the shares is large enough. Otherwise, masking is not effective [BCPZ16, GS18, BS21, DBL20].

As a result, masking ensures that the data complexity of worst-case attacks increases exponentially with the masking order d, which is so a cryptographic security parameter that can be tuned by the designer depending both on physical parameters and expected security level.

Evaluation of secure implementation. In a second step, an evaluator assesses that the side-channel security is indeed the one expected by the designer. Next we describe two widely used solutions that are *attack-based evaluation* and *leakage detection*.

A first approach, that is the historical approach of the side-channel community, is for the evaluator to take the place of an adversary [KJJ99, CRR02, BCO04]. This solution, next called attack-based evaluation consists for the evaluator to demonstrate that he can instantiate an adversary that breaks the implementations with a complexity lower than the expected security level.

The second common approach is called leakage detection. It is a "pass-or-fail" methodology that leverages statistical tests such as t-test, χ^2-test or T^2-test [GJJ$^+$11, CMG$^+$, MRSS18, BSS19, MWM21]. Its most used instantiation, called *Test Vector Leakage Assessment* (TVLA), consists in verifying empirically that the independence assumption is met [GJJ$^+$11, CMG$^+$]. By running leakage detection, the evaluator hopes that if he is not able to detect significant data dependent statistical differences, the adversary will not be able to exploit the leakage to recover secret keys.

1.2 Limitations of current state of the art

Limitations of attack-based evaluations. While attack-based evaluations are appealing because they directly map to a defined adversarial strategy, it is in opposition with the principles of modern cryptography. Indeed, modern cryptography is built on schemes and primitives for which the security depends exponentially on a security parameter (e.g., number of shares for masking) allowing to defeat any polynomially bounded adversaries [KL14]. That is, it ensures (up to negligible probability) that no attack performed by any bounded adversary will succeed, independently of her strategy.

Because attack-based evaluations rely on practical instantiations of (polynomial) adversaries bounded by N measurements and a fixed computational power, it does not allow to claim security against more powerful (yet realistic) adversaries.

Limitations of leakage detection. Leakage detection is usually considered as a preliminary evaluation due to its simplicity. Yet, since it is based on hypothesis testing, it suffers from a risk of false positive and false negative that makes their interpretation tedious in the context of side-channel evaluations (see [WO19a, WO19b, BSS19] for recent discussions). Because it is of a "pass-or-fail" nature, leakage detection can only be a qualitative indication for security and not a quantitative one. Additionally while it is not necessary the case (e.g., with [MRSS18, MWM21]) its most common instantiation, TVLA, only restricts to the evaluation of the independence assumption. Therefore, in the case the noise assumption is not met, the implementation can be interpreted as secure based on leakage detection while higher-order attacks can succeed with low complexity [BS21, BS20].

1.3 What are worst-case security evaluations?

Interestingly, the current situation exposed above is to consider adversaries with full knowledge of the implementation during the design stage. Yet, it may not be the case for the final security evaluation. In this thesis, I attempt to close this gap by contributing to the field of *worst-case security evaluation*.

Goal of worst-case security evaluations. The goal pursued in this thesis can be summarized with Figure 1.1 which represents the improvements of side-channel adversaries with time. There, the green vertical line represents the moment at which the implementation is deployed. At this time, the evaluator should define the security level N that is claimed for the implementation. By considering worst-case evaluations, the evaluator attempts to define the blue horizontal line which is a lower bound to the data complexity independent of any adversarial strategy.

It allows to circumvent the major drawbacks of attack-based evaluations. In the short term, at the time of deployment, it exists generally a gap between the best currently known attack and the worst-case bound [ABB+20]. Yet since it anticipates future improvements of the state of the art, this gap is expected to vanish in the long term. This is for example important when considering the recent improvements of adversaries based on deep-learning [ZBHV20, WAGP20, LZC+21].

General principles. Worst-case evaluation requires that the evaluator can fully characterise the physical implementations. Hence,

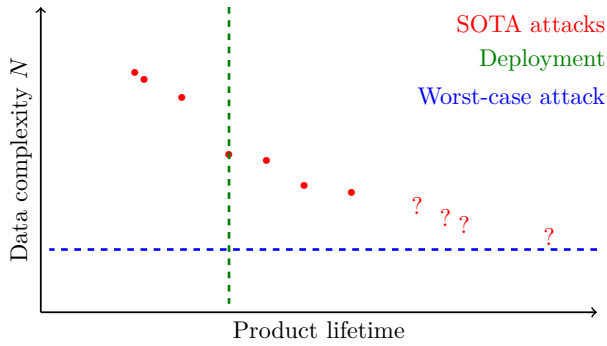


Figure 1.1: Goal of worst-case security evaluations.

worst-case evaluation favors open-source which allows the evaluator to modify, control and observe the entire protected design. Following the Kerckhoff's principle, it considers that the leakage behavior/distribution are public functions. Thanks to these principles, the previous limitations of attack-based and leakage detection can (partially) be escaped.

As mentioned in [SMY09], the leakage function is unknown to the evaluator. Indeed, while possible engineering intuition can provide prior "relatively good" approximation of the leakage behavior, it depends on the physical and technological behavior of the circuit under test [WOS14]. Therefore, the first step to perform a worst-case evaluation is always to characterize this leakage function. To do so, the evaluator has access to the source code, randomness ($\mathcal{R}$) and secrets (k) as illustrated in Figure 1.2. While the target is processing these inputs, the evaluator collects the side-channel leakages $\boldsymbol{l}$. He then reproduces all the deterministic computation (thanks to source code and randomness) and determines the manipulated data such as the shares x_i. Based on these, he characterizes the distribution of $\boldsymbol{l}$ conditioned to manipulated data that we denote as $\mathsf{m}(x|\boldsymbol{l})$.

Inherent limitations. Since worst-case evaluations start with the characterization of the leakage, it inherently suffers from limitations. First, the output of the evaluation depends on the quality of the measurements obtained by the evaluator. While he has, thanks to the open-source settings, the opportunity to build the cleanest measurement setup as possible, there is no guarantee that a better engineering effort would not result in better acquisition hence reduced security guarantees [ABB+20]. This also covers the case of another measurement technique (EM or power).

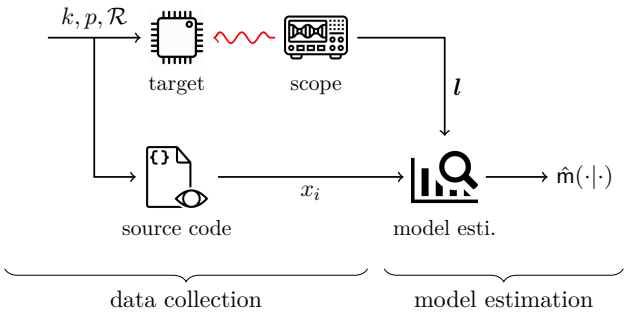


Figure 1.2: Profiling and leakage modeling for worst-case evaluations.

Second, the modeling of l denoted $\mathsf{m}(x|\boldsymbol{l})$ also depends on the estimation methodology. While we developed a few tools (Chapter 4) to bound the error made by this modeling, they do not scale to all models used in this thesis. In order to reflect these two limitations and as clear from the context, I often refer to close to worst-case evaluations.

1.4 Contributions and outline

This thesis is built on some of the research papers I had the opportunity to contribute to. Each of the following chapters are based on one or two of them. Next, I briefly describe the contribution of each of the chapters.

Proof-based evaluation. Part I of this thesis is dedicated to *proof-based* evaluations which exploit masking proofs to define security level [SMY09, DFS15, GS18]. It consists of verifying the two assumptions of masking proofs. First, the independence assumption can be heuristically verified by leveraging leakage detection [GJJ+11, CMG+, MRSS18, BSS19]. Second in order to access the noise assumption, the evaluator can quantify the noise with metrics such as the *Signal-to-Noise Ratio* (SNR) or *Mutual Information* (MI) [Man04, RSV+11, BHM+19]. Based on these evidence, the evaluator then estimates – up to constant factors [dCGRP19] – the complexity of the worst-case attack by raising the noise to the power d [GS18] or by using more complex models such as *Local Random Probing Model* (LRPM) [GGSB20].

In this thesis, I contribute to proof-based evaluations with tools to evaluate the two hypotheses. In Chapter 3, I illustrate that leakage detection benefits from open-source implementations and allows

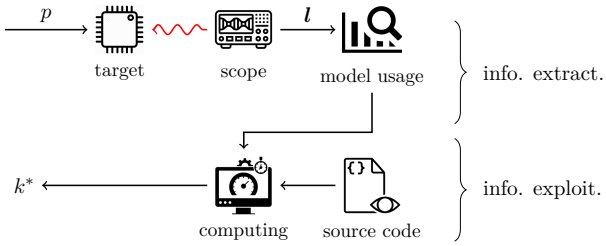


Figure 1.3: Worst-case attack evaluations.

to prevent/reduce the inherent limitations of the TVLA. To do so, I replace the univariate t-test by the multivariate Hotelling's T^2-test. In Chapter 4, I contribute to the quantification of the noise assumption. Namely, masking proofs require to know the MI which cannot be computed on real measurements since it requires the exact (not approximated) distribution of the leakage. To overcome this issue, I propose to use the *Hypothetical Information* (HI) which overestimates (is an upper-bound) the MI. As a result, using this upper-bound in the masking proofs allows to lower-bound the complexity of a side-channel attacks [BHM+19].

(Close to) worst-case attack evaluation. The second possibility to estimate the worst-case security of implementation is to perform *(close to) worst-case attacks*. There, the evaluator attacks an implementation thanks to its modeling of the leakage, knowledge of the implementation and its own computational resources, as illustrated in Figure 1.3. Namely, based on its modeling, he is able to extract information from the leakage. Then he can exploit this information to recover the cryptographic secret. By performing attacks on weak instantiations (low number of shares), he can finally extrapolate the attack data complexity for larger security parameters [BS21].

In Part II, I contribute to worst-case attack evaluations. In Chapter 5, I propose a classification of various profiled distinguishers. Namely we compare them based on the hypothesis made about the leakage distribution, the required (or not) knowledge of the randomness and the ability to exploit lower order flaws. There, I put forward that for an asymptotic number of profiling data, both attacks exploiting or not $\mathcal{R}$ are equivalent. However, knowledge of $\mathcal{R}$ significantly reduces the need for profiling traces which illustrates the interest of worst-case attacks. It essentially eases the profiling.

In Chapter 6, I propose a methodology to attack a combination

of countermeasures which recombines the results of *"partial"* attack on each of its components. I apply the methodology to an open-source masked AES provided by the *Agence Nationale de la Sécurité des Systèmes d'Information* (ANSSI) and revise the security analysis of the implementation. Namely, it demonstrates a successful key recovery with less than 1,000 traces while no weaknesses were previously found with 100,000 measurements by considering other adversarial approaches (i.e., not worst-case).

In Chapter 7, I propose a new methodology to evaluate masked bitslice implementations which is based on Gaussian template for information extraction and *Soft Analytical Side-Channel Attacks* (SASCA) for information exploitation. I first use it to perform attacks on implementations with low security parameters (from 3 to 8 shares). Then based the obtained attack complexities and the trends expected from masking proofs, I extrapolate the security to larger security parameters and additional countermeasures. It concludes that a large number of shares is needed in order to ensure large N. Namely, at least 16 shares are needed to ensure that $N > 2^{40}$.

Implication for new designs. From Part II, it appears that the intrinsic noise provided by low-cost microcontrollers is limited. Hence, it raises the questions of the protection of these platforms.

In Part III, Chapter 8, I study the belief that countermeasures need to be combined to be effective. There, it appears that it is feasible to combine shuffling and masking in such a way that shuffling is amplified by masking. However, this combination is not always the most efficient in terms of cost vs. security. As always when considering physical implementations, the best choice depends on the physical parameters such as noise and randomness cost. Eventually, we put forward that in the case of low-cost microcontrollers investigated in Part II, the noise is too low for shuffling to be effective.

Background 2

This chapter is based on the background of some of the research papers included in this thesis, with minor modifications.

Next, I introduce necessary notions that are used in multiple chapters of this thesis. If necessary, the beginning of each chapter is dedicated to introducing the additional tools needed to understand it. I start by introducing the notations and the information theoretic metrics for side-channel analysis. I continue with two side-channel countermeasures that are Boolean masking and shuffling. Finally, I conclude this chapter by describing evaluation tools going from standard template attacks to SASCA.

2.1 Notations

Next, we describe the notations that are common to all the chapters in this thesis. Yet, we note for certain topics, specific notations are needed and therefore the subsequent notations are not applicable.

Random variables are denoted with capital letters Y and their realizations with lower cases y. We use the notation $\boldsymbol{y}$ for the vector and $\boldsymbol{Y}$ for random vectors. Concretely, $\boldsymbol{y}$ is a vector and each element of the vector is denoted as y_i.

When considering masking, we use the notation y^j for the j-th share of y such that $\sum_{i=0}^{d-1} y^j = y$. Sharing can also apply to vectors such that $\boldsymbol{y}^j$ is the j-th share of $\boldsymbol{y}$ such that with the element-wise addition $\sum_{j=0}^{d-1} \boldsymbol{y}^j = \boldsymbol{y}$. Operations on vectors are element-wise. We further denote y_i^j as the j-th share of element i such that $\sum_{j=0}^{d-1} y_i^j = y_i$.

When we refer to shuffling, $\boldsymbol{\theta}$ denotes the permutation that is drawn from the set of all possible permutations $\boldsymbol{\Theta}$.

We assume that the leakage of a vector is a vector and the leakage of a variable can be either a variable or a vector. The leakage is denoted with l with the target random variable in superscript. For example:

- $\boldsymbol{l^y}$ is the leakage (vector) generated by the data vector $\boldsymbol{y}$.

- $\boldsymbol{l}^{\boldsymbol{y}^i}$ is the leakage (vector) generated by the i-th share vector of $\boldsymbol{y}$.
- $l^{y_i^j}$ is the leakage on the j-th share of the i-th element of $\boldsymbol{y}$.
- l^{θ_c} is the leakage on the permutation at index c.
- $l^{y^j_{\boldsymbol{\theta}_c}}$ is the leakage obtained when accessing the j-th share of the element indexed at cycle c when shuffling with the permutation $\boldsymbol{\theta}$.

2.2 Information theoretic metrics

Next, we introduce *Information Theoretic* (IT) metrics that can be used to evaluate the effectiveness of side-channel countermeasures (and, as we will see in Chapter 8, of adversarial strategies as well). The interest of these metrics is that the number of traces N required to perform a (worst-case) statistical attack against a leaking implementation is inversely proportional to the *Mutual Information* $\mathsf{MI}(\boldsymbol{Y}; \boldsymbol{L})$ between the secret random vector $\boldsymbol{Y}$ (e.g., encryption key) and the leakage $\boldsymbol{L}$ [DFS15]:

$$N \geq \frac{cst}{\mathsf{MI}(\boldsymbol{Y}; \boldsymbol{L})} \cdot \tag{2.1}$$

Since the cardinality of $\boldsymbol{Y}$ is generally too large to be exhausted, an adversary generally targets each element Y_i (e.g. a key byte) of the vector independently in a divide-and-conquer manner. Similarly, $\mathsf{MI}(Y_i; \boldsymbol{L})$ is usually computed in a similar way where the complexity to recover one such element which is given by:

$$N_{dc} \geq \frac{cst}{\mathsf{MI}(Y_i; \boldsymbol{L})} \cdot \tag{2.2}$$

The relation is given for a small constant cst (that depends on Shannon's entropy of Y_i $\mathsf{H}(Y_i)$ and the target success rate of the attack [dCGRP19]). In the context of attacks against implementations where the Y_i variables and their leakages are independent, the divide-and-conquer approach does not imply information losses [GS18]. As will be discussed Chapter 8, the situation is different when shuffling, since it creates dependencies between the leakages of the different Y_i variables.

2.2.1 Estimating MI – from simulations

In general, computing $\mathsf{MI}(\boldsymbol{Y}; \boldsymbol{L})$ requires the knowledge of the (true) Probability Density Function (PDF) of the leakage conditioned on $\boldsymbol{y}$, that we denote as $\mathsf{f}(\boldsymbol{L} = \boldsymbol{l} | \boldsymbol{Y} = \boldsymbol{y})$ or $\mathsf{f}(\boldsymbol{l}|\boldsymbol{y})$ for short. Such a knowledge

is only available in simulated evaluation contexts, where the leakage function is defined by the evaluator. In this thesis, and unless mentioned otherwise, we simulate leakages that follow a (possibly multivariate) Gaussian distribution with noise covariance $\mathbf{\Sigma}$ so that:

$$\mathsf{f}(\boldsymbol{l}|\boldsymbol{y}) = \mathcal{N}(\boldsymbol{l}|\mu_{\boldsymbol{y}}, \mathbf{\Sigma}). \tag{2.3}$$

Thanks to this PDF, the conditional probability of a sensitive variable $\boldsymbol{Y}$ given the leakage, denoted as $\Pr[\boldsymbol{Y} = \boldsymbol{y}|\boldsymbol{L} = \boldsymbol{l}] := \mathsf{p}(\boldsymbol{y}|\boldsymbol{l})$ and $\Pr[\boldsymbol{y}|\boldsymbol{l}]$ for short notation, can be computed via Bayes. Assuming that $\boldsymbol{Y}$ is uniformly distributed (which is the case for the cryptographic secrets we aim to recover), it is expressed as:

$$\mathsf{p}(\boldsymbol{y}|\boldsymbol{l}) = \frac{\mathsf{f}(\boldsymbol{l}|\boldsymbol{y})}{\sum_{\boldsymbol{y}^* \in \boldsymbol{\mathcal{Y}}} \mathsf{f}(\boldsymbol{l}|\boldsymbol{y}^*)}. \tag{2.4}$$

The MI can then be estimated via sampling, as described in [BHM+19], Equation 6:

$$\hat{\mathsf{MI}}(\boldsymbol{Y}; \boldsymbol{L}) = \mathsf{H}(\boldsymbol{Y}) + \sum_{\boldsymbol{y} \in \boldsymbol{\mathcal{Y}}} \mathsf{p}(\boldsymbol{y}) \cdot \sum_{i=1}^{n_t(\boldsymbol{y})} \frac{1}{n_t(\boldsymbol{y})} \cdot \log_2 \mathsf{p}(\boldsymbol{y}|\boldsymbol{l}^{\boldsymbol{y}}(i)), \tag{2.5}$$

where $\mathsf{H}(\cdot)$ is the Shannon entropy and $n_t(\boldsymbol{y})$ is the number of leakage samples $\boldsymbol{l}^{\boldsymbol{y}}(i)$ against which the conditional probability distribution is "tested" in the estimation (i.e., the larger $n_t(\boldsymbol{y})$, the better the evaluation of $\hat{\mathsf{MI}}(\boldsymbol{Y}; \boldsymbol{L})$).

Eventually, we note that MI can also be exactly computed (without sampling) by using its integral form [BHM+19]. We do not make use of such an expression in this work. Indeed, we make use of MI to analyse the security guarantees of protected designs. In this context, the leakages are generally highly multivariate making integration not affordable with our computing infrastructure.

2.2.2 Estimating PI – from any leakage

When moving from simulated evaluations to the concrete evaluation of actual leakages measured from a chip, and as initially observed in [SMY09, RSV+11], the true leakage distribution is generally unknown. The best option of an adversary is then to approximate the leakage distribution with a model, that we next denote as $\hat{\mathsf{f}}(\cdot|\cdot)$. This model is typically obtained by profiling the target device [CRR02]. The amount of information that can be extracted thanks to this model is captured by the so-called *Perceived Information* (PI). As the MI, the PI can be

estimated by sampling by replacing $\mathsf{p}(\cdot|\cdot)$ with $\hat{\mathsf{f}}(\cdot|\cdot)$ in Equation 2.5 such that:

$$\hat{\mathsf{PI}}(\boldsymbol{Y};\boldsymbol{L}) = \mathsf{H}(\boldsymbol{Y}) + \sum_{\boldsymbol{y}\in\mathcal{Y}} \mathsf{p}(\boldsymbol{y}) \cdot \sum_{i=1}^{n_t(\boldsymbol{y})} \frac{1}{n_t(\boldsymbol{y})} \cdot \log_2 \hat{\mathsf{f}}(\boldsymbol{y}|\boldsymbol{l}^{\boldsymbol{y}}(i)). \tag{2.6}$$

The PI is smaller than the MI unless a perfect model is used by the adversary (see [BHM+19] discussed in Chapter 4). In previous use cases of the PI, this metric was mostly used to capture estimation and assumption errors caused by an imperfect leakage profiling. In Chapter 8, we additionally use it to capture suboptimal adversarial strategies designed to be computationally efficient against shuffled implementations.

These notions will be used in most of the chapters of this thesis. Eventually in Chapter 4, we will formally prove that PI is a lower bound to MI. We will also introduce the metric HI, which is the case where an exhaustive model is built, is an upper bound to MI. As a result, MI can be bounded by this HI and PI.

2.3 Countermeasures

In the next section, we describe the two countermeasures we focus on in this thesis. We start with masking and then continue with shuffling. In both cases, we start by describing the countermeasure followed by its expected impact on the MI, which reflects its effectiveness.

2.3.1 Masking

General principle. Boolean masking is a popular countermeasure against side-channel attacks [CJRR99]. It consists in representing a variable a as an encoding (tuple) which is a vector $\boldsymbol{a}$ of d uniformly distributed elements (or shares) fulfilling $a = \sum_i a^i$. By doing so, all the sets of $d-1$ shares remain independent of the secret, which was formalised as *d-probing security* [ISW03]. Masked implementations then aim to maintain this property through the computations.

For this purpose, the generic solution is to perform operations with *masked gadgets* that ensure probing security. The linear operations are done share-by-share as illustrated with Algorithm 1 where $\oplus$ denotes a bitwise addition. Multiplication gadgets are used for the non-linear operations (e.g., Sbox): they allow multiplying two encodings. A prominent example is the ISW multiplication introduced by Ishai, Sahai and Wagner [ISW03] which is recalled in Algorithm 2 where $\otimes$ denotes

the multiplication in $\mathsf{GF}(2)$. It has a randomness requirement that is quadratic in the number of shares.

Algorithm 1 Masked Boolean addition.

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**
        $r \leftarrow \{0,1\}$
        $c^i \leftarrow c^i \oplus (r \oplus (a^i \otimes b^j))$
        $c^j \leftarrow c^j \oplus (r \oplus (a^j \otimes b^i))$

When combining multiple gadgets to build a circuit, designers must ensure that the composition of gadgets remains secure in the appropriate model. This can be done with automated tools. Regarding software implementations, [BGR18] can prove correct composition in probing model thanks to `tightPROVE`. Its subsequent work, [BDM+20] can generate implementations and prove that the entire circuit is secure in the *register probing model* thanks to the tool `Tornado`. Similar tools exist for masked hardware implementations in the *robust probing model* [CGLS20, FGP+18].

Concrete security. Duc *et al.* showed in [DDF14] that probing security reduces to security in the more realistic *noisy leakage model* where the adversary has access to the noisy leakages of all the intermediate variables. The (noise and independence) conditions needed for this result to hold in practice and its connection with the previous information theoretic metrics have then been made explicit in [DFS15], leading to the following bound for the data complexity of a side-channel

attack against a masked implementation:

$$N^{msk} \geq \frac{cst}{\prod_{j=0}^{d-1} \mathsf{MI}(\boldsymbol{Y}^j; \boldsymbol{L})}. \tag{2.7}$$

Precisely, the exponential complexity increase is only relevant if $\mathsf{MI}(\boldsymbol{Y}^j; \boldsymbol{L})$ is small enough and the security order d is maintained as long as the leakage function is a noisy combination of the shares.

Leakage distribution. Eventually, we describe the PDF of leakage conditioned to a masked variable x. Namely, its expression is given by:

$$\mathsf{f}(\boldsymbol{l}|x) = \sum_{\boldsymbol{x} \in \boldsymbol{X}} \mathsf{p}(\boldsymbol{x}) \cdot \mathsf{f}'(\boldsymbol{l}|\boldsymbol{x}), \tag{2.8}$$

where $\boldsymbol{X}$ is the set of all the possible combination of d shares to encode values of x and $\boldsymbol{x}$ is one of them. The PDF is so a mixture where each of the components $\mathsf{f}'(\cdot|\cdot)$ is the leakage PDF condition to an encoding $\boldsymbol{x}$. We note that the number of components in the mixture grows exponentially with the number of shares d. Indeed, the number of possible encodings of x is equal to $|X|^{d-1}$ where $|X|$ is the field size.

Bitslice implementations. Bitslicing is a software programming technique that represents an algorithm (e.g., a block cipher) with Boolean operations [Bih97]. It takes advantage of the parallelism enabled by bitwise instructions which are available in most the modern micro-controllers (MCU). For example, it is possible to place 32 bits in a register $\boldsymbol{a}$, 32 bits in a register $\boldsymbol{b}$, and to obtain the 32 bits corresponding to $\boldsymbol{a} \oplus \boldsymbol{b}$ in a single instruction. This strategy is particularly appealing in the context of masking where 32 bitwise masked gadgets such as Algorithm 1 and Algorithm 2 can be applied in parallel. Several works have shown how it can be efficiently used to protect implementations with arbitrary masking order [GR17, BGR18, BDM+20].

2.3.2 Shuffling

General principle. Shuffling is another side-channel countermeasure that leverages independent operations within a circuit (e.g., the 16 AES Sbox). It consists in performing these operations in a random order with the goal to confuse the side-channel adversary. Algorithm 3 is an example of shuffling that applies an arbitrary function $\mathsf{op}(\cdot)$ independently to all the elements of the input vector $\boldsymbol{y}$. The first step is to generate a permutation $\boldsymbol{\theta}$ which is uniformly selected among all the permutations of the set $\{0, \ldots, |\boldsymbol{y}|-1\}$, that we denote $\boldsymbol{\Theta}$. The size of the

permutation is next given by π. In this small example, it corresponds to η which is the number of independent operations on which we shuffle and corresponds to the size of $\boldsymbol{y}$. As will be seen in Chapter 8, there are also cases where $\pi > \eta$. For now, the algorithm iterates deterministically over the elements of this permutation. On the c-th iteration, the index θ_c of the element of the input vector to process during that iteration (denoted s) is loaded. The output is finally updated such that $z_s \leftarrow \mathsf{op}(y_s)$.

Algorithm 3 Shuffled execution.

Input: $\boldsymbol{y}$ and $\mathsf{op}(\cdot)$ with $|\boldsymbol{y}| = \eta$ and $\pi = \eta$.
Output: $\boldsymbol{z}$ such that $\forall i, 0 \leq i < \pi, z_i = op(y_i)$.

$\boldsymbol{\theta} \leftarrow \boldsymbol{\Theta}$ ▷ Perm. generation
for c in $[0, \ldots, \pi - 1]$ **do**
  $s \leftarrow \theta_c$ ▷ Leaks $\rightsquigarrow l^{\theta_c}$
  $z_s \leftarrow \mathsf{op}(y_s)$ ▷ Leaks $\rightsquigarrow l^{y_{\theta_c}}$ and $l^{z_{\theta_c}}$

In the context of a side-channel attack, all these operations generate leakage on the processed data. Namely, at iteration c, the c-th element in the permutation generates some leakage denoted as l^{θ_c}. We shall refer to it as the *permutation leakage*. The manipulation of the input vector also generates some leakage that we next call *data leakage*. We use the notation $l^{y_{\theta_c}}$ to represent the leakage generated when processing the θ_c-th index in the vector $\boldsymbol{y}$ during cycle c.

Concrete security. Under the assumption that the noise is sufficient to hide the permutation indexes, shuffling offers an increase of a side-channel attack's data complexity that is linear in the number of independent operations η on which the permutation is applied [HOM06, VMKS12]. Using the same information theoretic notations as for the masking countermeasure, it gives:

$$N^{shu} \geq \frac{cst \cdot \eta}{\mathsf{MI}_u(\boldsymbol{Y}; \boldsymbol{L})}, \tag{2.9}$$

where the denominator is the MI_u on a similar un-shuffled implementation.

Computing the PDF. The optimal way to compute the leakage PDF of a shuffled implementation is given by:

$$\mathsf{f}(\boldsymbol{l}|\boldsymbol{y}) = \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \mathsf{p}(\boldsymbol{\theta}) \cdot \mathsf{f}(\boldsymbol{l}|\boldsymbol{\theta}, \boldsymbol{y}), \tag{2.10}$$

where $\boldsymbol{l}$ is the concatenation of the permutation and data leakage vectors. However, summing over all the permutations rapidly turns out to be too computationally intensive (e.g., for $\pi = 16$, the number of modes of this mixture is already $\approx 2^{44.2}$). As a result, more computationally efficient approaches have been proposed, at the cost of a possible information loss. A first proposition has been made in [VMKS12] that we call AC12 and is described with Equation 8.3. In Subsection 8.3.1, Equation 8.4, we propose a new strategy that improves AC12 while keeping the same computational complexity.

2.4 Leakage modeling

In this work, we propose to approximate the true leakage PDF $\mathsf{f}(\boldsymbol{l}|x)$ of a variable x with Gaussian templates in linear subspaces that we denote $\hat{\mathsf{f}}(\boldsymbol{l}|x)$. To do so, the profiling phase consists in two steps that we describe below.

1. **Signal-to-Noise Ratio (SNR).** The leakage vector $\boldsymbol{L}$ can be too large to be modeled at once. In order to estimate its distribution, we first find the Points-of-Interest (POIs) in the traces. That is the indexes (i) for which the means of the leakages L_i depend on the variable to model x, and so allow to distinguish between the possible x's. To do so, we make use of the univariate metric SNR as defined by Mangard [Man04]. It corresponds to:

$$\mathsf{SNR}_i^X = \frac{\mathsf{Var}_{x \leftarrow X}\left[\mathsf{E}\left[L_i^x\right]\right]}{\mathsf{E}_{x \leftarrow X}\left[\mathsf{Var}\left[L_i^x\right]\right]}, \tag{2.11}$$

   where E and Var are respectively the expectation and variance operators. The n_s indexes i showing the largest SNR are then kept to build the templates. Since SNR is connected to IT metrics [Man04, MOS11], this methodology allows to only keep the indexes i containing the most univariate information. Even though this is not a guarantee that is maximizes the multivariate information, this thesis shows that it is a simple yet efficient heuristic.

2. **Gaussian Templates (gT).** The leakage vector $\tilde{\boldsymbol{L}}$ composed of these n_s POIs is then model with Gaussian templates [CRR02]. Additionally, we built these templates in a linear subspace with n'_s dimensions that is defined with the projection matrix $\mathbf{W}$ with dimensions $n'_s \times n_s$ [APSQ06, SA08]. Overall, the leakage PDF is

estimated with:

$$\hat{\mathsf{f}}(\tilde{\boldsymbol{l}}|x) = \frac{1}{\sqrt{(2\pi)^{n'_s} \cdot |\boldsymbol{\Sigma}|}} \cdot \exp^{\frac{1}{2}(\mathbf{W}\cdot\tilde{\boldsymbol{l}}-\mu_x)\boldsymbol{\Sigma}(\mathbf{W}\cdot\tilde{\boldsymbol{l}}-\mu_x)'} \tag{2.12}$$

where $\boldsymbol{\Sigma}$ is a pooled covariance matrix and μ_x is the mean of $\tilde{\boldsymbol{L}}$ conditioned to x. Eventually with the available samples for profiling, the evaluator (or adversary) first builds $\mathbf{W}$ either with Principal Component Analysis (PCA) or Linear Discriminant Analysis (LDA). Then, he estimates the means and the covariance.

While using a projection can possibly reduce the information extracted by the adversary if all the template parameters are perfectly estimated, it generally allows to reduce the data complexity required to estimate these parameters. It is particularly handy when dealing with a large number of POIs n_s. We refer to the Appendix of [BMPS21] for a quantitative comparison.

2.5 Attack strategies

In this thesis, we make only use of templates exploiting first order information (mean dependent) which are *"easy"* to estimate. To attack protected implementations, we so need to recombine the outcome of many of these templates as we detail next.

2.5.1 Divide-and-conquer

We start by describing a template attack [CRR02]. The goal of this attack is to recover a full key $\boldsymbol{k}$ by recovering each of its elements k_i independently hence it is a Divide-and-Conquer (D&C) strategy.

The first step in a template attack is to build a gT denoted $\hat{\mathsf{f}}(\boldsymbol{l}|x_i)$ where x_i is the output of the i-th Sbox (e.g., thanks to Section 2.4). During the attack phase, the adversary has N measurements $\boldsymbol{l}_n$ and the associated plaintexts m_i^n. She then applies Bayes' rule:

$$\hat{\mathsf{p}}(x_i|\boldsymbol{l}_n) = \frac{\hat{\mathsf{f}}(\boldsymbol{l}|x_i)}{\sum_{x'\in X}\hat{\mathsf{f}}(\boldsymbol{l}|x')}\,, \tag{2.13}$$

as well as a maximum likelihood to derive a key guess such that:

$$\hat{k}_i = \text{argmax}_{k_i^*} \prod_{n=0}^{N-1} \hat{\mathsf{p}}(x_i = w(k_i^*, m_i^n)|\boldsymbol{l}_n), \tag{2.14}$$

where $w(a, b) = \text{Sbox}(a \oplus b)$. The data complexity of this attack depends on the $\mathsf{MI}(\boldsymbol{L}; X_i)$ with the relation presented in Equation 2.2. It also depends on the ability of the gT to capture correctly the available information which is quantified with $\mathsf{PI}(\boldsymbol{L}; X_i)$.

Finally, we illustrated a template attack in the use case of a blockcipher where $w(\cdot, \cdot)$ is the Sbox output. However, the same methodology applies in other contexts. As an example, $w(\cdot, \cdot)$ can be taken as a tag verification function which compares a tag candidate controlled by the adversary and the correct tag that the adversary tries to recover [BBC+20a, BMPS21].

2.5.2 Soft analytical side-channel attack (SASCA)

The previous attack is limited in the sense that the adversary can only exploit leakages that depend on a single (or a few) k_i of the secret vector $\boldsymbol{k}$. 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 2.1. 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.

- Second, when she receives a leakage sample $\boldsymbol{l}$, she computes a marginal probability $\hat{\mathsf{p}}(v|\boldsymbol{l})$ for each variable nodes V within the graph thanks to previously estimated gTs for each of them.

- 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 2.1). At a lower level, this is performed by iteratively

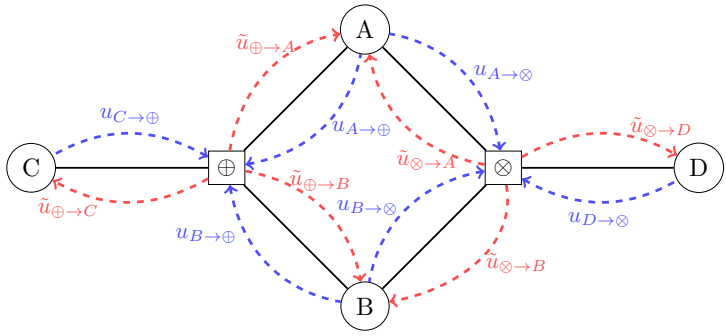


Figure 2.1: Example of factor graph for SASCA**.**

updating messages according to the following rules. From variables to functions, the update rule is given by:

$$u_{V\to f}^{(t+1)}(y) = \hat{\mathsf{p}}(V = y|\boldsymbol{l}) \prod_{k\in\partial V\setminus f} \tilde{u}_{k\to V}^{(t)}(y), \tag{2.15}$$

and from functions to variables the update rule is given by:

$$\tilde{u}_{f\to V}^{(t+1)}(y) = \sum_{\boldsymbol{x}\in\Psi(\partial f\setminus V)} \left(\psi_f(\boldsymbol{x}, y) \prod_{K\in\partial f\setminus V} u_{K\to f}^{(t+1)}(x_k) \right), \tag{2.16}$$

where ∂ denotes the neighbors of a node, $\Psi(\cdot)$ all the possible combination of values for a set of variable nodes and $\psi_f(\cdot)$ is the compatibility function.[1] Overall, the message passed by a variable node to a function node (i.e. Equation 2.15) is the product of the messages it receives from the other edges and the initial $\mathsf{p}(\cdot|\cdot)$. The message passed by a function node to a variable node (i.e. Equation 2.16) is the sum of the messages received from the other edges.

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

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

attacks and provides a reasonable estimate of the complete marginal distribution [GS15, BCPZ16, GS18, GRO18, KPP20].

In this thesis, we only consider Boolean additions and multiplications operating as function nodes. For a generic function with two inputs, the straightforward evaluation of Equation 2.16 has complexity in $\mathcal{O}(2^{2b})$ where b is the number of bits of the variable nodes, 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.

2.5.3 SCALib

The Side-Channel Analysis Library (SCALib) is an open-source library that provides most of the tools needed to perform the evaluation methodology we detail in this thesis. SCALib is a python package available on PyPI that provides optimized implementations of a series of algorithms for side-channel analysis. The focus of this library is on performance (both single-core and multi-threaded) and usability. SCALib is hosted at `https://github.com/simple-crypto/SCALib` and the documentation lives at `https://scalib.readthedocs.io/en/stable/`. Gaëtan Cassiers is a co-author of SCALib.

Part I

Proof-based evaluations

Multivariate leakage detection

3

This chapter is based on the work *"Multi-Tuple Leakage Detection and the Dependent Signal Issue"* published in TCHES 2019 [BSS19].[a]

At the time of the publication, the goal of the paper was to point the limitations of Test Vector Leakage Assessment (TVLA) as a leakage detection method. However, the rest of the thesis questions the pertinence of leakage detection as a standalone evaluation metric.

I therefore change the main message and illustrate its interest in the case of proof-based evaluations. Namely, I put forward that leakage detection can benefit from an open-source and that it should be used in combination with masking schemes secure in abstract models and a quantitative noise evaluation.

[a] Co-authors: Tobias Schneider and François-Xavier Standaert.

3.1 Contextualization

Standalone leakage detection. Leakage detection has become a de facto standard for the fast preliminary assessment of cryptographic implementations against side-channel attacks. In contrast to (worst-case) attack-based and proof-based evaluations which quantify the difficulty to recover a key, detection methodologies try to answer a much simpler question: *do the measurements obtained by an adversary contain data-dependent information, independent of whether it can be efficiently exploited?* Cryptography Research's non-specific (fixed vs. random) t-test is the most popular example of this trend [GJJ+11, CMG+]. It is commonly used to evaluate the security order of block cipher implementations by comparing leakages obtained for a fixed plaintext (and key) to leakages obtained with a random plaintext (and fixed key).

In general, leakage detection is thought to accelerate the evaluation process by avoiding the need to conduct numerous different attacks which require expert knowledge [Wag12]. Furthermore, it helps to

reduce the evaluations' data complexity[1], since it relies on a simple statistical test to answer the aforementioned question, instead of conducting a complete key recovery. As discussed in [MOBW13, SM16, DS16], this gain largely stems from the fact that it is easier to compare two classes of leakages than 2^b ones for a side-channel attack targeting a b-bit sub-key.

This reduced data complexity naturally comes at a cost. As it is based on hypothesis testing, it comes with a risk of "false positive" and "false negative". The false positives correspond to implementations considered as insecure while it is not. This occurs either because the leakage is induced by a non sensitive variable (e.g., plaintext) either because of statistical artifacts. The false negatives correspond to cases where no leakage is detected with n measurements while the implementation is insecure. Drawing such a conclusion can be damaging for security analysis and raise the question of how to interpret negative outcomes (no leakage found) of leakage detection [Sta18, WO19a, WO19b].

Indeed, while it can be handy for early analysis, leakage detection cannot be used to assess the security of an implementation in a standalone solution as discussed in recent works [Sta18, GS18, BS21]. More precisely, because its relies on hypothesis testing, leakage detection is suited to draw conclusion that are "there is evidence of leakage" or "there is no evidence of leakage". It will never be able to "prove" that no leakage is present in an implementation [WO19b, WO19a]. One example of such a limitation will be highlighted in Chapter 6 where we exhibit successful attacks while leakage detection does not exhibit weaknesses.

Leakage detection and provable circuits. Eventually while hypothesis testing comes with the above inherent limitations, it can still be useful in combination with proof-based evaluations and careful (proven) designs with arbitrary masking order such as Hardware Private Circuits (HPC) [CGLS20, CS21] or (almost) key homomorphic primitives [BSS21, DMMS21]. There, the designer can first instantiate the implementation with low security parameters (i.e., $d = 2$). Then he can run leakage detection to (carefully) convince himself that the design is first-order secure meaning that he correctly instantiated that design. Thanks to proofs, the designer can then increase d while being heuristically convinced that the same independence properties are met

[1] The number of measurements needed to conclude the evaluation.

even tough he is not able to experimentally demonstrate it.[2]

Contributions. Based on this state-of-the-art, our contributions in this chapter are the following:

1. First, we show that the data complexity of non-specific tests can be further reduced thanks to multi-tuple (or multivariate) leakage detection. To this end, we rely on the well-established Hotelling's T^2-test which is the natural extension of the t-test for multivariate variables. We propose a general instantiation of such a multi-tuple detection using Hotelling's T^2-test.

2. Second, we demonstrate that dependencies within the traces can strongly affect the results of TVLA, which rely on the assumption that each of the point in the traces are independent. This makes simple conclusions such as "no leakages were detected with up to N measurements" difficult to fairly compare statistically without a proper assessment of the dependencies within the traces. Since Hotelling's T^2-test naturally captures such dependencies, it therefore becomes a tool of choice whenever applicable. Yet, it comes at the cost of a significantly higher computational complexity, since it essentially requires computing and inverting the leakage traces' covariance matrix.

3. Third, we provide experiments with both simulated data. It illustrates that leakage detection is easy for open-source implementations. Indeed, it allows to directly apply detection to relevant sample in the leakage traces. As a result, the previous computational overheads can be limited.

Terminology clarification. We clarify that leakage detection allows to empirically convince the evaluator of the independence condition in masking proofs. In this chapter, we will study the impact of dependences within leakages traces. That is, the covariance matrix within the traces is not diagonal. As clear from the context, we next refer to independence assumption as being the assumption that the covariance matrix is diagonal. That is, all the samples in the traces are independent.

[2] We note that coupling issues cannot (for now) be assessed in abstracts models [CEM18, LBS19]. Even tough less efficient, key homomorphic designs allow to physically separate the shares both in time and in space which gives them an advantage regarding coupling issues [BSS21].

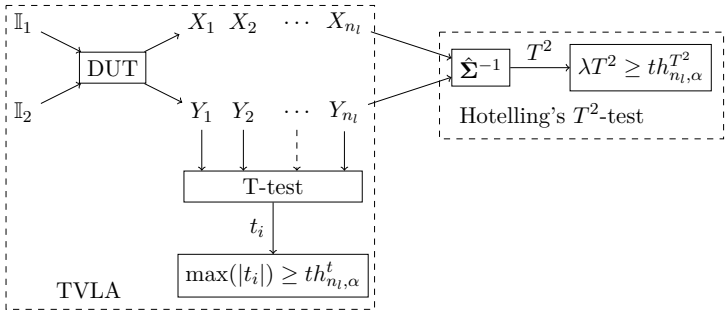


Figure 3.1: Leakage detection framework.

3.2 Preliminaries

In this section, we introduce the notations used in this chapter and explain the current state of leakage detection based on Welch's t-test [GJJ+11, CMG+] including its extension from [DZD+17]. It will serve as a frame of reference in our experiments to judge the potential of the newly-proposed methodology.

3.2.1 Notations

In this chapter, we take similar notations as Section 2.1 where random variables are denoted with capital letters, their realizations with low cases and vectors with bold symbols. In this context, n_l and N are the trace length and the number of available traces (i.e., measurements). We also use the full notation $\Pr[\cdot]$ to denote a probability to avoid confusion with the p-value.

We denote the mean and variance of a random variable X as μ_X and σ^2_X, while their sampled estimates from N_X measurements are written as $\hat{\mu}_X$ and $\hat{\sigma}^2_X$. For random vectors, $\mathbf{\Sigma}$ and $\hat{\mathbf{\Sigma}}$ represent the covariance matrix and its estimate. For statistical tests, α (resp., β) denotes the false positive (resp., negative) probability.

3.2.2 Leakage detection with Welch's t-test and extensions

Test Vector Leakage Assessment (TVLA) describes an efficient detection methodology based on Welch's t-test initially proposed by Cryptography Research [GJJ+11, CMG+]. TVLA tests the means of two different sets of measurements for equality and concludes the existence of leakages if a difference can be detected with a certain level of confidence. This basic procedure is split into two phases as depicted in Figure 3.1.

First, the device under test (DUT) is fed with two sets of inputs $\mathbb{I}_1$ and $\mathbb{I}_2$ while the leakage behavior during the computation is recorded. This results in a set of $N_{\boldsymbol{X}}$ (resp., $N_{\boldsymbol{Y}}$) measurements $\boldsymbol{X}$ (resp., $\boldsymbol{Y}$) for inputs $\mathbb{I}_1$ (resp., $\mathbb{I}_2$). In the widely-used non-specific fixed vs. random test, $\mathbb{I}_1$ consists of only one fixed plaintext and key while $\mathbb{I}_2$ corresponds to a fixed key with random plaintexts. In the following, our experiments are conducted in a *fixed vs. fixed* manner. As detailed in [DS16], it generally provides better detection rates. Yet, our findings are generic and transfer to the *fixed vs. random* strategy.

Second, Welch's t-test [Wel47] is applied to these measurements for each pair of random variables (X_i, Y_i) corresponding to one point in time separately. It is also a univariate metric as the SNR recalled in Section 2.4. TVLA tries to distinguish between the null hypothesis H_0 and its corresponding alternative hypothesis H_a with

$$H_0 : \quad \mu_{X_i} = \mu_{Y_i}, \qquad H_a : \quad \mu_{X_i} \neq \mu_{Y_i}. \tag{3.1}$$

To this end, it is required to estimate the mean and variance of each random variable and use them to derive the test statistic t_i and degrees of freedom v_i for each point in time as

$$t_i = \frac{\hat{\mu}_{X_i} - \hat{\mu}_{Y_i}}{\sqrt{\frac{\hat{\sigma}^2_{X_i}}{N_{X_i}} + \frac{\hat{\sigma}^2_{Y_i}}{N_{Y_i}}}}, \qquad v_i = \frac{\left(\frac{\hat{\sigma}^2_{X_i}}{N_{X_i}} + \frac{\hat{\sigma}^2_{Y_i}}{N_{Y_i}}\right)^2}{\frac{\left(\frac{\hat{\sigma}^2_{X_i}}{N_{X_i}}\right)^2}{N_{X_i}-1} + \frac{\left(\frac{\hat{\sigma}^2_{Y_i}}{N_{Y_i}}\right)^2}{N_{Y_i}-1}}. \tag{3.2}$$

Under H_0, t_i follows a Student's t-distribution $\mathcal{T}(\cdot, v_i)$ parameterized with v_i, where $\mathsf{F}_{\mathcal{T}}(\cdot, v_i)$ denotes the corresponding cumulative density function. If the observed t_i significantly differs from the expected distribution, H_0 is rejected and the alternative hypothesis H_a is accepted. To ensure that this rejection is done with sufficient confidence, the p-value can be computed from t_i and v_i as

$$p_i = 2(1 - \mathsf{F}_{\mathcal{T}}(|t_i|, v_i)), \tag{3.3}$$

where a small p-value indicates a high confidence to reject H_0. In this case, the evaluator concludes the existence of detectable leakages in the measurements.

Setting a threshold. Usually, the p-value is compared to α, which is set a priori depending on the desired detection accuracy, and if smaller

H_0 is rejected. This can be simplified for large numbers of observations for which $\mathcal{T}(\cdot, v_i)$ tends to a standard normal distribution. In this case, it is sufficient to directly set a threshold th^t_α for the t_i value, avoiding the computation of the degrees of freedom and p-values. For TVLA, the authors recommend a threshold of 4.5 for $|t_i|$, which if exceeded translates to $p_i < 0.00001$ assuming $v_i > 1000$. TVLA is usually employed using the min-p strategy to extend the test to traces with multiple points in time, i.e., the threshold $th^t_{10^{-5}} = 4.5$ is not affected by the number of points in each trace. However, as noted in [GJJ+11] and more recently in [DZD+17, WO19b, WO19a], this approach can lead to erroneous results for measurements consisting of large numbers of points. In [DZD+17] it is put forward to set an adjusted threshold based not only on the desired false positive rate α but also on the number of points per trace n_l (i.e., the number of separate Welch's t-tests) as

$$th^t_{n_l,\alpha} = \mathsf{F}^{-1}_{\mathcal{T}}(1 - \frac{1 - (1-\alpha)^{1/n_l}}{2}, v). \tag{3.4}$$

In this way, the authors ensure that the desired false positive rate is achieved even for large n_l assuming independence between the random variables. We use this approach to set the threshold for the combination of min-p and Welch's t-test in our experiments. By inverting the previous equation, the p-value is computed according to:

$$p = 1 - (1 + 2(\mathsf{F}_{\mathcal{T}}(\max(|t_i|)) - 1))^{1/n_l}. \tag{3.5}$$

Eventually, the false negative rate β which is the probability to not reject H_0 while $\mu_{\boldsymbol{X}} \neq \mu_{\boldsymbol{Y}}$ is given by:

$$\beta = \Pr[th^t_{n_l,\alpha} > \max(|t_i|)] \tag{3.6}$$

$$= \Pi_{i=1}^{n_l} \Pr[th^t_{n_l,\alpha} > |\mathcal{T}(v_i; \delta_i)|], \tag{3.7}$$

where:

$$\delta_i = (\mu_{X_i} - \mu_{Y_i}) / \sqrt{\frac{\sigma^2_{X_i}}{N_{X_i}} + \frac{\sigma^2_{Y_i}}{N_{Y_i}}} \tag{3.8}$$

is the non-central parameter. We observe that the false negative probability depends on the number of measurements as well as the samples' variance and the difference in mean between the two vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$. Namely, by decreasing the noise variance or by increasing the number of samples, the probability β decreases.

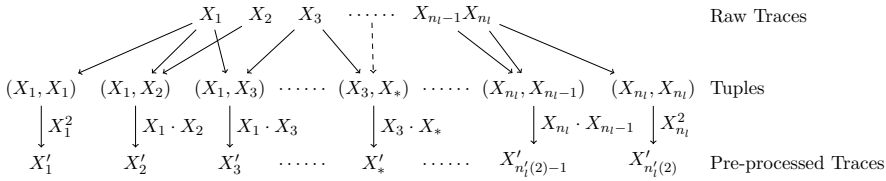


Figure 3.2: Exemplary multiplicative pre-processing of $\boldsymbol{X}$ for order $d = 2$.

Extension to higher orders. The previously-described procedure tests for a difference in the means between the random vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$. Next, we will illustrate how it can be extended to test for the d-th statistical moment. To do so, the traces need to be pre-processed before applying TVLA [CMG+].

A usual heuristic for this purpose is to combine every d-tuple of random variables multiplicatively, as shown in Figure 3.2. This produces the pre-processed trace sets $\boldsymbol{X'}$, $\boldsymbol{Y'}$ which we define as:

$$\boldsymbol{X'} = \left\{ \prod_{i \in \mathbb{T}} X_i, \forall\ \mathbb{T} \overset{d}{\subset} \{1, \ldots, n_l\} \right\},\ \boldsymbol{Y'} = \left\{ \prod_{i \in \mathbb{T}} Y_i, \forall\ \mathbb{T} \overset{d}{\subset} \{1, \ldots, n_l\} \right\},$$

for all considered sets of tuple indices $\mathbb{T}$ with cardinality d. Leakages can only be detected if such a d-tuple contains information about all d shares of the targeted sensitive variable. If there are dth-order leakages, the means of $\boldsymbol{X'}$ and $\boldsymbol{Y'}$ (resp., the dth-order moments of $\boldsymbol{X}$ and $\boldsymbol{Y}$) are different and TVLA will conclude the existence of leakages given enough samples. Note that sometimes, the centralized (mean-free) or standardized (normalized by standard deviation) moments are used which allow efficient and stable estimations with a one-pass algorithm [SM16].

This pre-processing quickly becomes very complex in a closed-source approach, since the evaluator does not know which d variables need to be combined to detect leakages. Thus, it is necessary to check all possible $n'_l(d)$ d-tuples[3], with:

$$n'_l(d) = \binom{n_l + d - 1}{d} = \frac{(n_l + d - 1)!}{d!(n_l - 1)!}, \tag{3.9}$$

[3] Other approaches than this pre-processing exist to deal with masked implementations, such as projection pursuits or dimensionality reductions [DSV+15, CDP16], but they are more applicable in an attack-based evaluation than in a detection-based one we consider here.

which we can bound with:

$$\frac{n_l^d}{d!} \leq n_l'(d) \leq \frac{(n_l + d)^d}{d!}\,. \tag{3.10}$$

This exponential increase strongly affects the density (i.e., the ratio of informative vs. non-informative tuples) which decreases exponentially in d.

Note also that samples pre-processed in this way are not Gaussian, and therefore the heuristic of testing the sample means relies on the central limit theorem in this case.

3.3 Multi-tuple / multivariate detection

Welch's t-test as applied in the TVLA methodology exploits only the difference in the means of two paired random variables (X_i, Y_i) as stated in its testing hypotheses (see Equation 3.1). Therefore, it does not take full advantage of jointly testing all the variables in the vectors $(\boldsymbol{X}, \boldsymbol{Y})$, and instead runs n_l separate tests in the hope that at least one of them gives large enough confidence to reject the null hypothesis. By taking an analogy with distance between two vectors in an n_l-dimensional space, this procedure only looks at the distances between all the dimensions separately.

As a solution to this problem, we propose to use *multivariate* – or equivalently multi-tuple – tests for leakage detection. Instead of Equation 3.1, such a multivariate detection methodologies consider the following testing hypotheses:

$$H_0: \quad \mu_{\boldsymbol{X}} = \mu_{\boldsymbol{Y}}, \qquad H_a: \quad \mu_{\boldsymbol{X}} \neq \mu_{\boldsymbol{Y}}. \tag{3.11}$$

H_0 assumes that there is no difference of means between the two vectors $\boldsymbol{X}$ and $\boldsymbol{Y}$ (instead of variables X_i and Y_i), whereas H_a corresponds to the presence of leakages.

3.3.1 Hotelling's T^2-Test

Hotelling's T^2-test is a natural candidate for our use case because it is inherently multivariate [Hot31] and corresponds to the statistical hypotheses described in Equation 3.11. The test statistic T^2 is computed according to:

$$T^2 = \frac{N_{\boldsymbol{X}} N_{\boldsymbol{Y}}}{N_{\boldsymbol{X}} + N_{\boldsymbol{Y}}} (\hat{\mu}_{\boldsymbol{X}} - \hat{\mu}_{\boldsymbol{Y}})' \hat{\boldsymbol{\Sigma}}^{-1} (\hat{\mu}_{\boldsymbol{X}} - \hat{\mu}_{\boldsymbol{Y}}), \tag{3.12}$$

and requires the estimated means $(\hat{\mu}_{\boldsymbol{X}}, \hat{\mu}_{\boldsymbol{Y}})$ of both random vectors and their estimated pooled covariance matrix $\hat{\boldsymbol{\Sigma}}$ defined as:

$$\hat{\boldsymbol{\Sigma}} = \frac{\sum_{i=1}^{N_{\boldsymbol{X}}} (\boldsymbol{x_i} - \hat{\mu}_{\boldsymbol{X}})(\boldsymbol{x_i} - \hat{\mu}_{\boldsymbol{X}})' + \sum_{i=1}^{N_{\boldsymbol{Y}}} (\boldsymbol{y_i} - \hat{\mu}_{\boldsymbol{Y}})(\boldsymbol{y_i} - \hat{\mu}_{\boldsymbol{Y}})'}{(N_{\boldsymbol{X}} + N_{\boldsymbol{Y}} - 2)}. \quad (3.13)$$

Hotelling's T^2-test assumes that the covariance of both random vectors $\boldsymbol{X}$, $\boldsymbol{Y}$ is the same and pools their estimates to achieve better accuracy. However, small differences in the covariances can be tolerated by the test without strongly affecting its performance (as discussed next).[4] Under the null hypothesis and up to some multiplicative factor λ, this test statistic follows a Fisher distribution $\mathcal{F}$ parameterized with $(n_l, N_{\boldsymbol{X}} + N_{\boldsymbol{Y}} - 2)$ such that:

$$\frac{(N_{\boldsymbol{X}} + N_{\boldsymbol{Y}} - 1 - n_l)}{(N_{\boldsymbol{X}} + N_{\boldsymbol{Y}} - 2) n_l} T^2_{H_0} = \lambda \, T^2_{H_0} \sim \mathcal{F}(n_l, N_{\boldsymbol{X}} + N_{\boldsymbol{Y}} - 2). \quad (3.14)$$

In contrast to Welch's t-test, the form of this distribution does not only depend on the number of measurements $(N_{\boldsymbol{X}}, N_{\boldsymbol{Y}})$, but also on the number of samples in a trace n_l. Based on this, the p-value and a detection threshold for T^2 can be computed as

$$p = 1 - \mathsf{F}_{\mathcal{F}}(\lambda T^2), \qquad th^{T^2}_{\alpha} = \mathsf{F}^{-1}_{\mathcal{F}}(1 - \alpha)/\lambda,$$

where $\mathsf{F}_{\mathcal{F}}$ denotes the CDF of the Fisher distribution. The threshold is, therefore, a function of the number of available measurements, the trace length, and the desired false positive probability α. This dependency is depicted in Figure 3.3. For fixed (exemplary) parameters $\alpha = 0.1$ and $N_{\boldsymbol{X}} = N_{\boldsymbol{Y}} = 64$, we report the PDF's and CDF's of the Fisher distribution for varying n_l. It is noticeable that the shape of the distributions is strongly affected by the length of the traces n_l and, therefore, the threshold $th^{T^2}_{\alpha}$ (represented by the dashed lines) also changes accordingly.

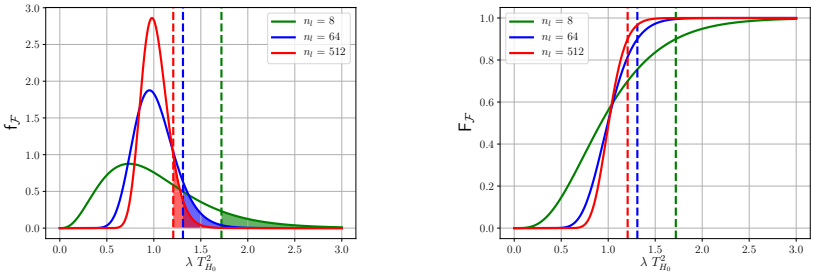
Under the alternative hypothesis, the test statistic follows a non-central Fisher distribution:

$$\lambda \, T^2_{H_a} \sim \mathcal{F}(n_l, N_{\boldsymbol{X}} + N_{\boldsymbol{Y}} - 2; \delta), \quad (3.15)$$

with the effect size δ as an additional parameter which is defined as:

$$\delta = \frac{N_{\boldsymbol{X}} N_{\boldsymbol{Y}}}{N_{\boldsymbol{X}} + N_{\boldsymbol{Y}}} (\mu_{\boldsymbol{X}} - \mu_{\boldsymbol{Y}})' \boldsymbol{\Sigma}^{-1} (\mu_{\boldsymbol{X}} - \mu_{\boldsymbol{Y}}) \cdot \quad (3.16)$$

[4] Note that there are versions of Hotelling's T^2-test which do not rely on this assumption. However, they have increased data complexity, and thus are not considered in the following.



(a) PDF **(b) CDF**

Figure 3.3: The PDF and CDF of different Fisher distributions with varying n_l and their corresponding threshold $th_\alpha^{T^2}$ (dashed lines) for $\alpha = 0.1$ and $N_{\boldsymbol{X}} = N_{\boldsymbol{Y}} = 64$.

Here, δ represents the distance between the null and the alternative hypothesis for any covariance, means and number of measurements. Based on this distribution, we can compute the false negative probability β (so under H_a with $\mu_{\boldsymbol{X}} \neq \mu_{\boldsymbol{Y}}$) as proposed in [Ren98]:

$$\beta = \Pr[\lambda\ th_\alpha^{T^2} > \mathcal{F}(n_l, N_{\boldsymbol{X}} + N_{\boldsymbol{Y}} - 2; \delta)]. \tag{3.17}$$

This gives the probability that the observed test statistic T^2 is smaller than the detection threshold even though the mean difference between $\boldsymbol{X}$ and $\boldsymbol{Y}$ is unequal zero.

3.3.2 Leakage detection with Hotelling's T^2-test.

We propose to follow the same procedure as TVLA (and all of its instantiations) to generate the measurements, and replacing the n_l independent Welch's t-test with one single Hotelling's T^2-test as depicted in Figure 3.1. This comes with two advantages:

1. Because it is inherently multivariate, it does not require an independence assumption. This allows a stable and fair control of the false positive and false negative (with a prior on the effect size δ) as mentioned in [WO19a].

2. In the case of dense traces, it allows to reject the null hypothesis will a smaller data complexity than the TVLA procedure (see Figure 3.5).

These improvements come at the cost of the estimation and inversion of the pooled covariance matrix $\hat{\boldsymbol{\Sigma}}$ (see Equation 3.13). Since $\hat{\boldsymbol{\Sigma}}$ is an

$n_l \times n_l$ matrix, storing and computing on it is roughly quadratic in the trace length. For more details about the computational cost of this methodology, we refer to Appendix A.2.

Extension with different covariance. Note also that even though the standard Hotelling's T^2-test assumes equal covariances between the sets, it can still be applied in the context of leakage detection for which this is not necessarily true for every scenario (e.g., in the case of masked implementations). As pointed out in [Wy92], the test is robust for unequal covariances under two conditions: *(i)* (N_X, N_Y) are large and equal, and *(ii)* N/n_l is large. The first condition (i.e., the equality in size of the two sets) can be trivially forced by evaluators. The second condition is typically fulfilled for protected designs which are the main target for leakage detection, for which (hundreds of) thousands of traces are necessary to detect. Indeed, n_l is anyway limited by the computational capabilities of the evaluator due to the cost computing the covariance matrix. For example, in our following experiments, we restricted it $n_l = 1,000$.[5]

3.4 Simulated experiments

The underlying idea of multi-tuple detection is to take advantage of multiple joint differences in the traces to reduce the data complexity required for detection. We next use simulated experiments to compare Hotelling's T^2-test and Welch's t-test with min-p.

In this respect, our starting observation is that (as already observed with the higher criticism [DZD+17]), these gains may depend on the target implementation. For example, it may happen that not all samples considered in a detection procedure are informative. These non-informative points might thwart the estimation of the multi-tuple test statistic by increasing the noise and, thus, cancel the aforementioned advantage. In order to fairly compare the different approaches (Welch's t-test with min-p and Hotteling's T^2-test), our simulations will therefore assess the influence of two main parameters on the detection performances:

1. **The density** refers to the ratio of (pre-processed) informative points in the traces.

[5] Note that despite Hotelling's T^2-test returns a single statistic, it does not lead to a complete loss of intuition on the leaking points in a measurement trace. Indeed, it still requires computing mean vectors $\hat{\mu}_X$ and $\hat{\mu}_Y$ which provide such an indication, yet with less confidence due to the faster detection.

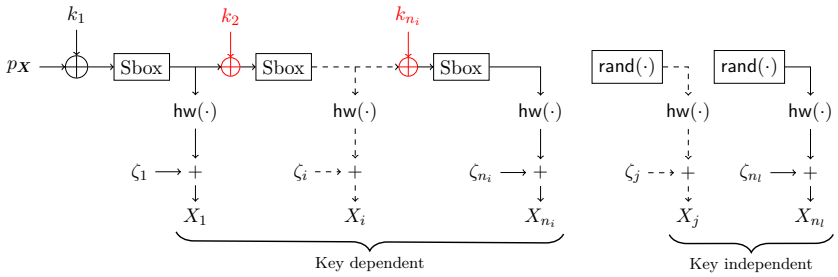


Figure 3.4: Simulation framework.

2. **The dependency** is the level of deviation of the covariance matrix from diagonal.

As usual, simulations are useful since they allow us to thoroughly assess the influence of these parameters in a controlled environment. Besides the density and dependency (which carry the most important intuitions), we also consider the SNR, β, and trace length n_l as additional parameters influencing our results (in a way that is essentially similar to what was observed in previous works on leakage detection).

In the following, we first introduce our simulation framework and a methodology to estimate the success rate of Welch's t-test and Hotelling's T^2-test. Secondly, we use simulations with independent leakage. Thirdly, we highlight the dependent signal issue and its effect on leakage detection methodologies.

3.4.1 Simulation framework

In our simulations, we use the common Hamming weight (denoted with $\mathsf{hw}(\cdot)$) model with additive Gaussian noise. To this end, we initially pick two fixed 8-bit inputs $p_{\boldsymbol{X}}$ and $p_{\boldsymbol{Y}}$.[6] These are run through a round-based structure consisting of the addition of an independent round key k_i followed by a bijective function Sbox : $\{0,1\}^8 \mapsto \{0,1\}^8$ as depicted in Figure 3.4. The value after each function is considered as an informative leakage sample, to which we apply $\mathsf{hw}(\cdot)$ and add noise $\zeta \sim \mathcal{N}(\mathbf{0}, \boldsymbol{\Sigma})$. The noise is sampled from a multivariate Gaussian distribution with the zero vector $\mathbf{0}$ as means and $\boldsymbol{\Sigma}$ as its covariance matrix. Depending on the scenario, this matrix is chosen to be either diagonal or with varying levels of dependency between its variables (see Figure 3.7). The diagonal is set to σ^2 depending on the desired SNR [Man04], which is equal to $2/\sigma^2$ assuming an 8-bit Hamming weight

[6] Note that these are picked as $p_{\boldsymbol{X}} \neq p_{\boldsymbol{Y}}$ to ensure leakage.

leakage model and equal covariances for both sets. In order to ensure the independence of the manipulated values independently of the Sbox, a key addition with independent keys $k_i \leftarrow \{0,1\}^8$ is performed at each round.

Density. The density in our simulations is controlled by repeatedly adding random values to the trace as shown in Figure 3.4. These generate leakage samples that are independent of the key and plaintext making them non-informative. In total, we generate traces of length $n_l = n_i + n_o$, where n_i denotes the number of informative points, that are padded with n_o points of noise. Based on this setting, we define the density of our simulated measurements as:

$$\phi = \frac{n_i}{n_l}\,, \tag{3.18}$$

where $\phi = 1$ denotes that every point is sensitive while $\phi = 0$ means none point is sensitive. In our simulated experiments, we will only generate first-order leakages and rather indirectly emulate the effect of this pre-processing by adapting the simulation parameters (e.g., the number of samples and density of informative points) – which allows simpler interpretation.

3.4.2 Simulation with independent signal

We start by comparing the tests assuming independent signals. To this end, only the diagonal elements of the covariance matrix are set to the desired noise level, while we fix the remaining elements to zero (cf. Figure 3.7a). After describing the evaluation methodology, we primarily focus on the impact of the trace length (n_l) and the density (ϕ) on the data complexity (N) for a fixed detection rate ($1 - \beta$).

Evaluation method. Next, the simulations are performed in three steps. First, the desired false positive probability α is set as well as the other implementation parameters (density ϕ, trace length n_l, covariance matrix $\mathbf{\Sigma}$ and SNR). If not mentioned, α is equal to the frequently assumed 10^{-5}. Secondly, the desired false negative probability β is set. Finally, the inputs for the two sets are selected at random, the mean vectors $\mu_{\boldsymbol{X}}$ and $\mu_{\boldsymbol{Y}}$ are obtained from Figure 3.4. Then, the number of measurements (N) needed to achieve the targeted detection rate is computed by leveraging β expression in Equation 3.17 for Hotelling's T^2-test and Equation 3.7 for t-test with min-p. We note that β depends

on the effect size δ (or δ_i) which depends on $\mu_{\boldsymbol{Y}} - \mu_{\boldsymbol{X}}$. Therefore, this last step is repeated and averaged to remove input dependencies. [7]

Influence of multiple-tuples. First, we set the density to $\phi = 1$ (i.e., all points in a trace are informative) and observe the influence of the remaining simulation parameters (SNR, n_l) on the data complexity N. The results are given in Figure 3.5 where the vertical axis denotes the number of measurements N, while the horizontal axis shows an increasing n_l for fixed SNR (on the left) and an increasing SNR for fixed n_l (on the right).

The most noticeable influence on the detection complexity stems from the number of points in a trace n_l. While both tests (which are equivalent for $n_l = 1$) benefit from increasing the trace length, the T^2-test improves more than Welch's t-test with min-p, as shown in Figure 3.5a. This behavior is expected given that the T^2-test combines the differences of all points into a single test statistic while the min-p approach only looks for one worst-case. Therefore for dense traces, multivariate tests can detect (much) faster than Welch's t-test in case of sufficiently long traces. In our example, for $n_l = 10^3$, Welch's t-test with min-p requires 698 measurements to achieve a false negative probability $\beta = 10^{-3}$, while the T^2-test needs only 229 measurements (which corresponds to an improvement by a factor 3). For $n_l = 10^6$, Welch's t-test requires 303 traces while T^2-test only 205.

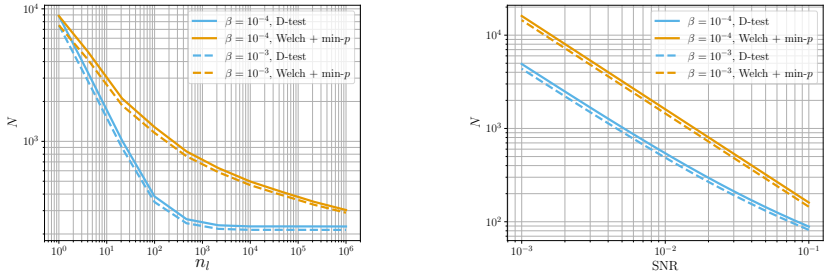
The other parameters affect the detection rate of both tests as expected. Decreasing β basically means a need for more measurements. The same can be observed for changes in the SNR given that more noise/less signal also increases N (as depicted in Figure 3.5b).

Influence of the density. We next study the influence of the density at a given noise level as shown in Figure 3.6. Now, the horizontal axis shows the density ϕ, while the vertical axis is still the number of traces required to detect N.

For a density equal to one, we observe that the T^2-test requires fewer traces than Welch's t-test with min-p, as expected from Figure 3.5. However, it is noticeable that both methods suffer from a decreasing the density.

For the T^2-test, the decreased density increases the number of zero elements in the vector $\mu_{\boldsymbol{X}} - \mu_{\boldsymbol{Y}}$. Therefore, the effect size δ (see Equation 3.16) decreases as well as the distance between the test statistic distribution under the zero and alternative hypothesis. This must be

[7] Where $N = N_{\boldsymbol{X}} + N_{\boldsymbol{Y}}$ and $N_{\boldsymbol{X}} = N_{\boldsymbol{Y}}$.



(a) $\mathsf{SNR} = 0.01$ **(b)** $n_l = 64$

Figure 3.5: Number of traces N required to detect for a fixed false negative rate β on simulated traces with a density of $\phi = 1$, and varying trace length n_l and SNR.

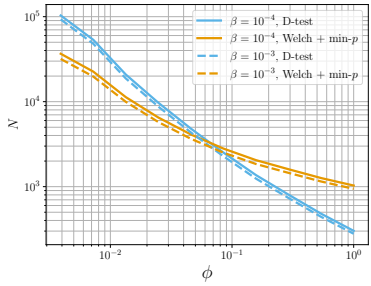


Figure 3.6: Number of traces N required to detect for a fixed false negative rate β on simulated traces with $\mathsf{SNR} = 0.01$, and varying density ϕ.

counterbalanced by increasing the number of samples to keep a constant δ and so the desired β. For Welch's t-test with min-p, decreasing the density reduces the number of t_i's which can potentially show leakages and, therefore, reduces the overall chance to detect leakages and increases the number of required measurements. More precisely, in Equation 3.7, an increasing proportion of δ_i's is equal to zero, making the overall detection rate smaller at a fixed number of measurements. Nevertheless, Figure 3.6 clearly shows that Welch's t-test with min-p suffers less from the reduced density than the multivariate test. For small densities, it even outperforms the T^2-test.

In a closed-source setting, the density is a priori not known by the evaluator. Hence, it is not possible to predict which test performs better a priori. Instead, both Welch's t-test with min-p and the T^2-test should be run in parallel. For high densities, the T^2-test will typically detect faster. For low densities, the opposite conclusion will hold. However in an open-source setting, he has an a priori knowledge of the points in the traces that are data dependent. Hence, he can preselect them to increase density. This is the first advantage of open-source in leakage detection.

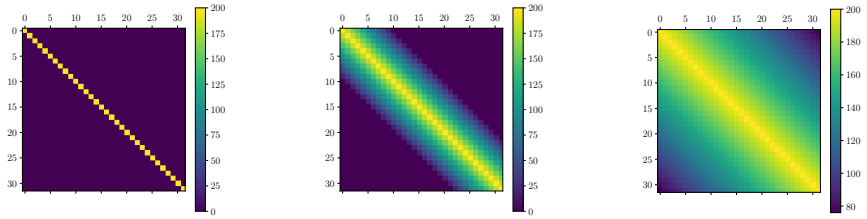
3.4.3 Simulation with dependent signal

We now explore the dependent signal issue and its effect on Welch's t-test with min-p and Hotelling's T^2-test. In particular, we consider three different noise covariance matrices $\mathbf{\Sigma}_1$, $\mathbf{\Sigma}_2$, $\mathbf{\Sigma}_3$ as depicted in Figure 3.7. $\mathbf{\Sigma}_1$ is a diagonal matrix like in the previous section (i.e., it corresponds to independent signals), while $\mathbf{\Sigma}_2$ and $\mathbf{\Sigma}_3$ represent increasing rates of dependency between the variables of $\boldsymbol{X}$ and $\boldsymbol{Y}$.[8]

In the following, we first show the influence of a non-diagonal covariance matrix on the distributions of the test statistics. Next, we highlight that it can lead to a wrong estimation of α, hence to a false sense of security based on sub-optimal detection performances. Thirdly, we confirm our intuitions with simulated experiments using $\mathbf{\Sigma}_2$ and $\mathbf{\Sigma}_3$.

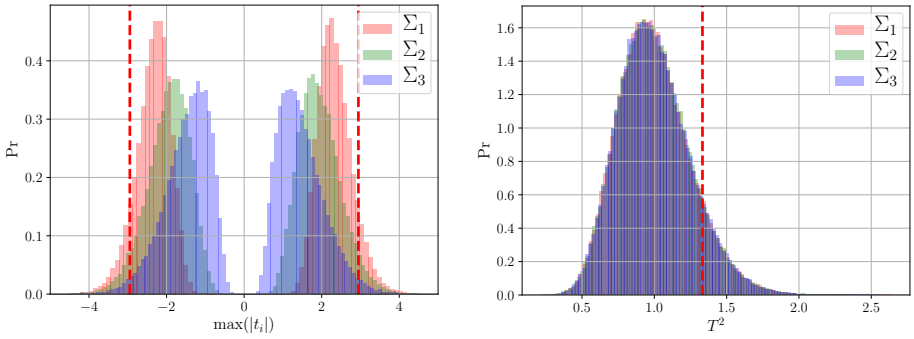
Evaluation method. Given independent signals, all the previous test statistics follow well-defined distributions under the null hypothesis. When considering non-diagonal covariance matrices, this is not necessary the case. Therefore, the following experiments rely on a sampling of the test statistics distributions for small (yet illustrative) examples. Namely, we will first generate $100,000$ times a test statistics under null hypothesis

[8] Further descriptions of each matrix are given in Appendix A.1.



(a) $\mathbf{\Sigma}_1$ (b) $\mathbf{\Sigma}_2$ (c) $\mathbf{\Sigma}_3$

Figure 3.7: Covariance matrices considered in our experiments (for $n_l = 32$).



(a) Welch + min-p. (b) Hotelling's T^2-test.

Figure 3.8: Null distributions with dependent noise and detection thresholds (dashed lines).

and derive an estimate of its distribution.[9] Then, we take the same approach for the alternative hypothesis.

Test statistic distributions under H_0. The sample test statistics distributions are reported in Figure 3.8. The distribution of Welch's t-test with min-p statistic is composed of two lobes (cf. Figure 3.8a) since only the largest $|t_i|$ from n_l different Welch's t-test is kept. For $\mathbf{\Sigma}_1$, we observe the expected distribution (in red) assuming independent signals. However, by adding slight dependencies in the traces as in the $\mathbf{\Sigma}_2$ case, the test statistic distribution (in green) is pushed towards the center. This trend is amplified with $\mathbf{\Sigma}_3$ (in blue). By contrast, Hotelling's T^2-test does not rely on the independence signal assumption and, therefore, the test statistic distributions remain the same in the different settings.

As detailed in Section 3.2 the estimation of the threshold $th_{n_l,\alpha}$ for a targeted α as well as the p-value estimation depends on the knowledge of the test statistic distribution under H_0. Yet if the leakage contains

[9] This is done for $N = 4,000$.

	$\mathbf{\Sigma}_1$	$\mathbf{\Sigma}_2$	$\mathbf{\Sigma}_3$
Welch's t-test with min-p	0.099	0.051	0.018
Hotelling's T^2-test	0.101	0.099	0.102

Table 3.1: Empirical false positive rate $\hat{\alpha}$ estimated from $100,000$ experiments.

dependencies (i.e., with $\mathbf{\Sigma}_2$ and $\mathbf{\Sigma}_3$), the latest is not known by the evaluator which can so not derive correctly $th_{n_l,\alpha}$ and the p-value. In order to illustrate this, we report in Table 3.1 the estimated false positive rate $\hat{\alpha}$ for the three considered covariance matrices. This $\hat{\alpha}$ corresponds to the actual false positive probability when an evaluator (possibly wrongly) assumes independence and sets a detection threshold $th_{n_l,\alpha}$ to reach one specific α. In the following table, we target an $\alpha = 0.1$.

In the $\mathbf{\Sigma}_1$ case, all $\hat{\alpha}$ converge to the expected value since the independent signal assumption is indeed fulfilled. By contrast, for $\mathbf{\Sigma}_2$ and $\mathbf{\Sigma}_3$, the derived threshold and p-value are no longer correct for Welch's t-test with min-p. Namely, it converges to values that are smaller than $\alpha = 0.1$ when increasing the dependencies. This confirms the intuition from Figure 3.8a, where the distributions are pushed towards the center. It results in a smaller fraction of the estimated test statistics exceeding the threshold, hence a lower $\hat{\alpha}$.

Hence, only Hotelling's T^2-test gives the correct false positive rate even for dependent signals, since it does not rely on the independent signal assumption.

Test statistic distributions under H_a. Intuitively, these results indicate that Welch's t-test with min-p may behave too *conservatively* if incorrectly assuming independence. In such cases, it will declare the presence of leakages with a confidence level higher than expected by the evaluator, leading to a reduced detection rate (increased β) as shown in Figure 3.9. In this figure, we estimate the correct detection thresholds for $\mathbf{\Sigma}_2$ by sampling which is feasible since we choose $\alpha = 0.1$, and use them to compare the detection rates with faulty and correct thresholds. It is obvious that the performances of the test suffer significantly from wrongly assuming independence. This effect is easily explained by the most extreme case of fully dependent signal and noise (i.e., if all entries of $\mathbf{\Sigma}$ are the same). In this scenario, all n_l tests would produce the same test statistic and should, therefore, be reduced to only one test. Nevertheless, if this test statistic is still compared to a threshold assuming n_l independent tests, it will result in an overestimation of the

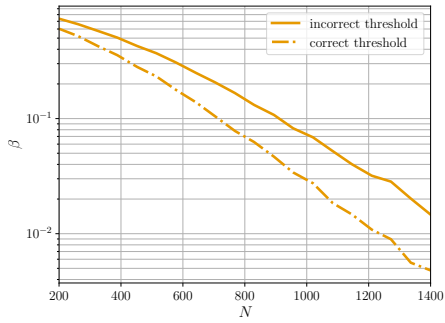


Figure 3.9: Detection rate errors due to incorrect signal independence assumption based on simulated traces with the covariance matrix $\mathbf{\Sigma}_2$, $\mathsf{SNR} = 0.004$ and Welch's t-test with min-p.

security, i.e., the evaluator may not detect existing leakages because of the erroneous threshold.

The second advantage of T^2-test is that it always enables a tight control of the false positive probability α. That is, by using T^2-test, the low efficiency of the statistical tests is due to a low difference difference between the means in two sets and not due to a badly parametrized test.

3.5 Towards a sound usage leakage detection

As mentioned in the introduction of this chapter, leakage detection cannot be a standalone solution to assess the security of an implementation [WO19b, WO19a]. Indeed, it only verifies the independence condition required for masking proofs and omits the noise condition. It should be used in combination with masking scheme secure at arbitrary order d. Then, the evaluator convinces himself heuristically that the design is well instantiated for low number of shares and then use the scalability of the design to extrapolate its conclusion for larger d.

Additionally, even if it remains qualitative, leakage detection with T^2-test benefits from the open-source requirement of worst-case evaluations. Indeed, it allows to use a priori knowledge of the possible leakage location to *i)* increase the density leading to more efficient detection and *ii)* tightly control false negative probability.

Bounding MI and modeling errors

4

This chapter is based on the work *"Leakage Certification Revisited: Bounding Model Errors in Side-Channel Security Evaluations"* presented at CRYPTO 2019 [BHM+19].[a]

I have been involved in the experimental part. Therefore, I will not present the proofs contained in the paper but rather depict the big picture and why these results are of interest in the context of my thesis. These will be illustrated with the experimental aspects I have been responsible of.

[a] Co-authors: Clément Massart, Julien Hendrix and François-Xavier Standaert.

4.1 Contextualization

As mentioned in the introduction of this thesis, proof-based evaluations allow to estimate the security of an implementation beyond the evaluator computational power. In the first step, it requires to make sure that the assumption of masking proofs are met. In Chapter 3, we have discussed how the independence assumption is assessed. In this chapter, we will focus on the quantification of noise/information within the leakage traces.

State of the art. The (worst-case) security evaluation of actual implementations and side-channel countermeasures requires estimating the amount of information leaked by a target device [SMY09]. Fair evaluations ideally require exploiting a perfect leakage model (i.e., a model that perfectly corresponds to the leakage distribution) with a Bayesian distinguisher. Yet, such a perfect leakage model is in general unknown. Therefore, side-channel security evaluators (and adversaries) have to approximate the statistical distribution of the leakages using density estimation techniques. It raises the problem

that security evaluations can become inaccurate due to estimation and assumption errors in the leakage model. Estimation errors are due to an insufficient number of measurements for the model parameters to converge. Assumption errors are due to incorrect choices of density estimation tools (e.g., assuming Gaussian leakages for non-Gaussian leakages).

The problem of ensuring that a leakage model is "good enough" so that it does not lead to over-estimating the security of an implementation has been formalized by Durvaux et al. as *leakage certification* [DSV14]. In the first leakage certification test introduced at Eurocrypt 2014, a leakage model is defined as good enough if its assumption errors are small with respect to its estimation errors. Intuitively, it guarantees that given the amount of measurements used by the evaluator / adversary to estimate a model, any improvement of his (possibly incorrect) assumptions will not lead to noticeable degradations of the security level (since the impact of improved assumptions will be hidden by estimation errors). In a heuristic simplification proposed at CHES 2016 [DSP17], a model is considered as good enough if the statistical moments of the model do not noticeably deviate from the statistical moments of the actual leakage distribution. In both cases, the certification tests are based on challenging the model against fresh samples in a cross-validation step. In both cases, the certification tests are qualitative and conditional to the number of measurements available to build the model. By increasing the number of measurements (and if the model is imperfect), one can make estimation errors arbitrarily small, which inevitably leads to the possible detection of assumption errors. As a result, a fundamental challenge in side-channel security evaluations (which we tackle in this chapter) is to *bound the information loss due to model errors quantitatively.*

We note that from an information theoretic viewpoint, the risk of under-estimating the leakages due to model errors in side-channel security evaluations can be captured with the notion of Perceived Information (PI) initially introduced in [RSV+11] to analyze model variability in nanoscale devices (see Subsection 2.2.2). Informally, the PI corresponds to the amount of information that can be extracted from some data thanks to a statistical model possibly affected by estimation or assumption errors. If the model is perfect, the PI is identical to Shannon's standard definition of Mutual Information (MI). Otherwise, the difference between the MI and the PI provides a quantitative view of the information loss. (Yet, at this stage not a usable one since the MI is unknown, just as the perfect model).

Contribution. The main contributions of the chapter are to provide simple and efficient information theoretic tools in order to bound the model errors in side-channel security evaluations, and to validate these tools empirically based on simulated leakages and actual measurements.

Our starting point for this purpose is a third information theoretic quantity that was introduced as part as a negative result on the way towards the CHES 2016 heuristic leakage certification test. Namely, the Hypothetical Information (HI), which is the amount of information that would be extractable from the samples if the true distribution was the statistical model. As discussed in [DSP17], as such the HI seems useless since in the case of incorrect model, it can be completely disconnected from the true leakage distribution (i.e., models with positive HI may not lead to successful attacks). Yet, we show next how it can be used in combination with the PI in order to enable quantitative leakage certification. In particular, our main results in this direction are twofold:

1. First, we show that – *under the assumption that the target random variable (e.g., the secret key) has constant (e.g., uniform) probability* – the empirical HI (eHI), which corresponds to the HI estimated directly based on the empirical leakage distribution, is in expected value an upper bound for the MI and that it converges monotonically towards the true MI as the number of measurements used in order to estimate the leakage model increases.

2. Second, we show that (under the same assumptions) the PI is a lower bound for the MI.

4.2 Definitions of IT metrics

Next, we recall the definitions of MI and PI (already introduced in Section 2.2) with the formalisms of [BHM+19] and define the HI. The relation between these definitions are graphically reported in Figure 4.1. There, distributions are represented in rectangular boxes, datasets in circular boxes and equations to estimate the metrics in dashed boxes.

4.2.1 Sampling process

In the continuous leakage setting, the true leakage distribution is a PDF and is denoted with $\mathsf{f}(\boldsymbol{l}|x)$. In the case of discrete leakage, it is a *Probability Mass Function* (PMF) and is denoted with $\mathsf{p}(\boldsymbol{l}|x)$. The true leakage distributions are generally unknown, but we can sample them in order to produce data sets for estimating leakage models and testing these models. We denote these sampling processes respectively

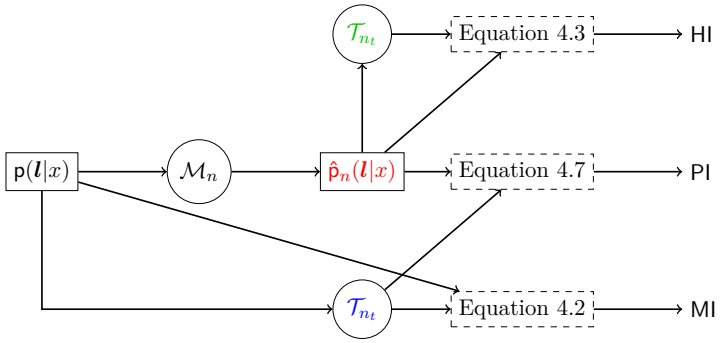


Figure 4.1: Graphical representation of IT metrics.

as $\mathcal{M} \leftarrow \mathsf{p}(\boldsymbol{l}|k)$ and $\mathcal{T} \leftarrow \mathsf{p}(\boldsymbol{l}|k)$ in the discrete case, and use $n(x)$ and $n_t(x)$ for the number of i.i.d. samples measured and stored per key in the multisets of samples $\mathcal{M}$ and $\mathcal{T}$ (which have repetitions). In the rest of the chapter, we assume an equal sampling for all keys so that $n(x) = n$ and $n_t(x) = n_t$. We replace $\mathsf{p}(\cdot|\cdot)$ by $\mathsf{f}(\cdot|\cdot)$ for the continuous case. When the true leakage distribution is sampled, we denote the resulting dataset as the empirical distribution. In Figure 4.1, the empirical distribution is denoted in blue with $\mathcal{T}_{n_t}$.

4.2.2 Computing MI by sampling

The MI is exactly defined in the discrete case with the equation:

$$\mathsf{MI}(X;\boldsymbol{L}) = \mathsf{H}(X) + \sum_{x\in\mathcal{X}} \mathsf{p}(x) \cdot \sum_{\boldsymbol{l}\in\boldsymbol{\mathcal{L}}} \mathsf{p}(\boldsymbol{l}|x) \cdot \log_2 \mathsf{p}(x|\boldsymbol{l}). \tag{4.1}$$

In case of a continuous leakage, the MI metric can also be computed directly thanks to integration by replacing the sum by an integral in the previous equation. Yet, this is not also practical for high dimensional leakages. It can also be estimated "by sampling" (for both discrete and continuous leakages) as:

$$\widehat{\mathsf{MI}}(X;\boldsymbol{L}) = \mathsf{H}(X) + \sum_{x\in\mathcal{X}} \mathsf{p}(x) \cdot \sum_{i=1}^{n_t} \frac{1}{n_t} \cdot \log_2 \mathsf{p}(x|\boldsymbol{l}_x(i)), \tag{4.2}$$

where $\boldsymbol{l}_x(i) \in \mathcal{T}$ is the i-th leakage sample observed for the value x. In the discrete case, it is easy to see that the blue part of the equation corresponds to the empirical distribution. So Equation 4.2 essentially replaces the true distribution $\mathsf{p}(\boldsymbol{l}|x)$ by the empirical one (hence the blue color), and the hat sign is used to reflect that the MI is computed by sampling. Since the empirical distribution converges towards the

real one as $n_t \to \infty$, $\widehat{\mathsf{MI}}(X;\boldsymbol{L})$ also tends towards $\mathrm{MI}(X;\boldsymbol{L})$. In the continuous case, the convergence requires more elaboration (details are given in Appendix of [BHM+19]). We note that the blue part of Equation 4.2 is the empirical in both the discrete and continuous cases.

Note that the PMF after the log in Equation 4.2 is fixed (i.e., it is not an estimate). So this equation does not describe an estimation of the MI in the usual sense, where the joint probability of two random variables has to be estimated: it only provides an alternative way to compute the MI of some known distribution. Hence it does not suffer from the bias issues discussed in [Pan03].

4.2.3 Model estimation

Given a set of $n \cdot |\mathcal{X}|$ modeling samples $\mathcal{M}$, we denote the process of estimating the conditional leakage distribution as $\hat{\mathsf{f}}_n(\boldsymbol{l}|x) \leftarrow \mathcal{M}$ for continuous leakage and $\hat{\mathsf{p}}_n(\boldsymbol{l}|x) \leftarrow \mathcal{M}$ for discrete leakage, where we use the red color to highlight the model and the hat sign to reflect that it is the result of a statistical estimation.

We will consider two types of models: *exhaustive models* where we directly estimate the empirical distribution (e.g., in the discrete case they correspond to histograms on the full support of the observations); *simplified models* which may for example correspond to histograms with reduced numbers of bins in the discrete case, or to parametric (e.g., Gaussian template described in Section 2.4) PDF estimation in the continuous case. Simplified models are aimed to converge faster (i.e., to require lower n values before becoming informative), possibly at the cost of some information loss when $n \to \infty$. In other words, exhaustive models (sometimes slowly) converge towards the real distribution as $n \to \infty$, while simplified models may be affected by assumption errors appearing for large n's (i.e., bad choices of parametric estimation such as assuming Gaussian noise for non-Gaussian leakages).

This is illustrated with more details in Chapter 5. There, we will present many distinguishers and classify them according to the information loss (distance between PI and MI), the assumption they do about leakage distribution and their convergence speed with n.

4.2.4 Hypothetical and perceived information

Given that the true distributions $\mathsf{p}(\boldsymbol{l}|x)$ or $\mathsf{f}(\boldsymbol{l}|x)$ are unknown, we cannot directly compute the MI from real measurements. One option to get around this impossibility is to estimate it, which is known to be a hard problem (i.e., there are no unbiased and distribution-independent

estimators [Pan03]). We next study an alternative approach which is to analyze the information that is revealed by estimated models thanks to two previously introduced and easy-to-compute quantities. First the PI, which is the amount of information that can be extracted from some data thanks to an estimated model, possibly affected by estimation or assumption errors [DSV14]. Second the HI, which is the amount of information that would be revealed by (hypothetical) data following the model distribution [DSP17].

Informally, the PI predicts the concrete success probability of a maximum likelihood attack exploiting an estimated model just as the (unknown) MI predicts the theoretical success probability of a worst-case maximum likelihood attack exploiting the true leakage distribution [GHRS15]. It can be negative if the estimated model is too different from the true distribution, and therefore can underestimate the information available in the leakages. By contrast, the HI is a purely hypothetical value that is always non-negative and can therefore overestimate the information available in the leakages. We next aim to formalize their properties, and in particular to show that they can be used to (lower and upper) bound the worst-case security level captured by the unknown MI.

Definition and estimation of HI. The HI is defined as follows in the discrete case (replace $\sum$ by $\int$ in the continuous case):

$$\mathsf{HI}_n(X;\boldsymbol{L}) = \mathsf{H}(X) + \sum_{x\in\mathcal{X}} \mathsf{p}(x) \cdot \sum_{\boldsymbol{l}\in\mathcal{L}} \hat{\mathsf{p}}_n(\boldsymbol{l}|x) \cdot \log_2 \hat{\mathsf{p}}_n(x|\boldsymbol{l}). \tag{4.3}$$

For a given estimated model $\hat{\mathsf{p}}_n(\boldsymbol{l}|x)$, the HI can be computed exactly based on Equation 4.3, or by sampling (just as for the MI). In the latter case, we use the notation $\widehat{\mathsf{HI}}_n(X;\boldsymbol{L})$:

$$\widehat{\mathsf{HI}}_n(X;\boldsymbol{L}) = \mathsf{H}(X) + \sum_{x\in\mathcal{X}} \mathsf{p}(x) \cdot \sum_{i=1}^{n_t} \frac{1}{n_t} \cdot \log_2 \hat{\mathsf{p}}_n(x|\boldsymbol{l}_x(i)), \tag{4.4}$$

with as main difference from the MI case that the test samples come from a set $\mathcal{T}_{\mathrm{m}}$ which has been picked up from the model distribution rather than the true distribution. We denote this process as $\mathcal{T}_{\mathrm{m}} \leftarrow \hat{\mathsf{p}}_n(\boldsymbol{l}|x)$, and use the green color to denote the empirical distribution of the model.

Note that, as in Equation 4.2, the model after the log in Equation 4.4 is fixed. Similarly to the MI estimation process, the value of the estimation $\widehat{\mathrm{HI}}(X;\boldsymbol{L})$ when $n_t \to \infty$ equals $\mathrm{HI}(X;\boldsymbol{L})$.

Definition and estimation of PI. Next, the PI is theoretically defined as follows in the discrete case:

$$\mathsf{PI}_n(X;\boldsymbol{L}) = \mathsf{H}(X) + \sum_{x\in\mathcal{X}} \mathsf{p}(x)\cdot\sum_{\boldsymbol{l}\in\mathcal{L}} \mathsf{p}(\boldsymbol{l}|x)\cdot\log_2 \hat{\mathsf{p}}_n(x|\boldsymbol{l}), \tag{4.5}$$

and as follows in the continuous case:

$$\mathsf{PI}_n(X;L) = \mathsf{H}(X) + \sum_{x\in\mathcal{X}} \mathsf{p}(x)\cdot\int_{\boldsymbol{l}\in\mathcal{L}} \mathsf{f}(\boldsymbol{l}|x)\cdot\log_2 \hat{\mathsf{p}}_n(x|\boldsymbol{l})\ dl. \tag{4.6}$$

In contrast with the HI, these equations cannot be computed directly since they require the knowledge of the true distributions $\mathsf{p}(\boldsymbol{l}|x)$ and $\mathsf{f}(\boldsymbol{l}|x)$ which are unknown. *So concretely, the* PI *will always be computed thanks to the following sampling process* (where we keep the red color code for the model and the blue color code for the true empirical distribution, as before):

$$\widehat{\mathsf{PI}}_n(X;\boldsymbol{L}) = \mathsf{H}(X) + \sum_{x\in\mathcal{X}} \mathsf{p}(x)\cdot\sum_{i=1}^{n_t}\frac{1}{n_t}\cdot\log_2 \hat{\mathsf{p}}_n(x|\boldsymbol{l}_x(i)). \tag{4.7}$$

This is feasible in practice since, even though the analytical form of the true distributions is unknown to the evaluator, he can sample these distributions, by measuring his target implementation.

Note again that, as in Equation 4.2, the model after the log in Equation 4.7 is fixed. So what the PI captures is the amount of information that can be extracted from some fixed model (usually obtained by estimation in an earlier phase). In other words, the PI computation is a two-step process: first a model is estimated, second the amount of information it provides is estimated. This is captured in our equations with the tilde and hat notations: the first one is for the estimation of the model, the second one for the computation of the information theoretic metrics by sampling.

Next, when considering an exhaustive model (i.e., histogram following the empirical distribution) PI and HI are respectively denoted with ePI and eHI. When a Gaussian model is considered, the PI is denoted with gPI.

4.3 Theorems, relations and bounds

Next, we report the two main theorems of [BHM+19] and refer to the original paper for technical details.

Theorem 1 *On average over the profiling sets $\mathcal{M}$ used to estimate the* eHI *and assuming that the target random variable X has (constant) uniform probability, we have:*

$$\mathop{\mathsf{E}}_{\mathcal{M}\leftarrow \mathsf{p}(\boldsymbol{l}|x)}\Big(\mathsf{eHI}_{n-1}(X;\boldsymbol{L})\Big) \geq \mathop{\mathsf{E}}_{\mathcal{M}\leftarrow \mathsf{p}(\boldsymbol{l}|x)}\Big(\mathsf{eHI}_{n}(X;\boldsymbol{L})\Big) \geq \mathsf{MI}(X;\boldsymbol{L}).$$

Moreover, $\lim_{n\to\infty} \mathsf{eHI}_n(X;\boldsymbol{L}) = \mathsf{MI}(X;\boldsymbol{L})$ *(i.e., the* eHI *monotonically converges towards the* MI*).*

Theorem 2 *Assuming that the target random variable X has (constant) uniform probability and given any model $\hat{\mathsf{p}}_n(\boldsymbol{l}|x)$ for the conditional probabilities $\mathsf{p}(\boldsymbol{l}|x)$, we have:*

$$\mathsf{PI}_n(X;\boldsymbol{L}) := \mathsf{H}(x) + \sum_x \mathsf{p}(x) \sum_l \mathsf{p}(\boldsymbol{l}|x) \log_2 \frac{\hat{\mathsf{p}}_n(\boldsymbol{l}|x)}{\sum_{x^*} \hat{\mathsf{p}}_n(\boldsymbol{l}|y^*)} \leq \mathsf{MI}(Y;\boldsymbol{L}).$$

Putting all together, $\mathsf{MI}(X;\boldsymbol{L})$ can be upper and lower bounding with eHI and PI such that:

$$\mathsf{PI}_n(X;\boldsymbol{L}) \leq \mathsf{MI}(X;\boldsymbol{L}) \leq \mathsf{eHI}_n(X;\boldsymbol{L}). \tag{4.8}$$

4.4 Experimental validation

4.4.1 Simulated experiments

In order to demonstrate the relevance of the previous metrics, we start by investigating a standard simulation setting where the evaluator / adversary exploits the leakages corresponding to several executions of the AES Sbox. We will next described three simulated scenarios similar to the one of Subsection 3.4.1.

1. **Univariate unprotected.** The first scenario corresponds to a univariate attack against an unprotected implementation of this Sbox, where the leakage samples are of the form:

   $$l = \mathsf{hw}\Big(\mathrm{Sbox}(x \oplus k)\Big) + \zeta,$$

   with hw the Hamming weight function, and ζ a Gaussian distributed noise sample with variance σ^2. The noise level is a parameter of our simulations. For convenience (and simpler interpretation) we report it as an SNR.

2. **Bivariate unprotected.** The second simulated scenario corresponds to a bivariate attack against the same unprotected implementation of the AES Sbox, where the leakage vectors are of the form:

$$\boldsymbol{l} = \Big[\mathsf{hw}(x \oplus k) + \zeta;\ \mathsf{hw}\Big(\mathrm{Sbox}(x \oplus k)\Big) + \zeta'\Big].$$

3. **Univariate Masked.** Finally, our third scenario corresponds to a univariate attack against a masked implementation of this Sbox, where the leakage samples are of the form:

$$\boldsymbol{l} = \Big[\mathsf{hw}\Big(\mathrm{Sbox}(x \oplus k) \oplus r\Big) + \mathsf{hw}(r) + \zeta\Big],$$

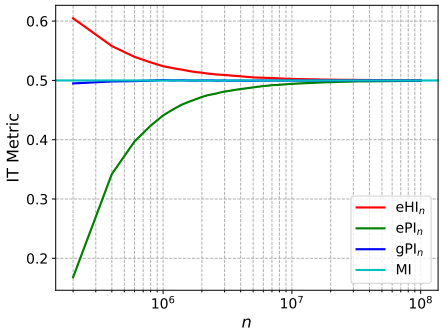
where r is a secret mask that is picked up uniformly at random by the leaking device.

Univariate unprotected. The results of our first scenario for high and medium SNR are in Figure 4.2, where we plot the MI (that is known since we are in a simulated setting), the eHI, the ePI both using exhaustive models and gPI which is the PI corresponding to a Gaussian leakage model. The IT metrics are plot in function of the number of traces in the profiling set n.[1] As expected, the eHI provides an average upper bound that converges monotonically towards the MI, and the ePI provides a lower bound. Besides, the gPI converges rapidly towards the true MI since in our simulations, the leakages are generated based on a Gaussian distribution. So making this additional assumption in such an ideal setting allows faster model convergence without information loss.

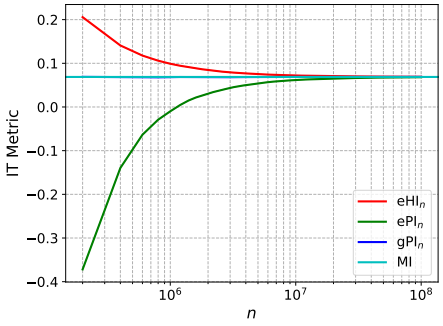
These results are confirmed with the similar plots given in Figure 4.3 for a lower SNR of 0.01. For readability, the right plot switches to a logarithmic scale for the Y axis. It illustrates a context where it is possible to formally bound the mutual information to values lower than 10^{-2}.

Figure 4.2 and 4.3 correspond to simple (unprotected, univariate) cases where the estimation of the empirical distribution (despite significantly more expensive than the one of a Gaussian distribution) leads to reasonably tight bounds for the MI.

[1] We use $n_t = n$, which leads to good estimations since the number of measurements needed to estimate a model is usually larger than the number of leakages needed to recover the key with a well-estimated model [SKS09].

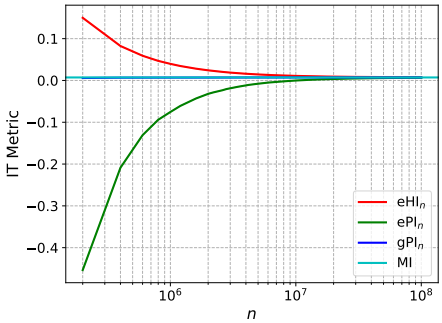


(a) SNR = 1 (lin. scale).

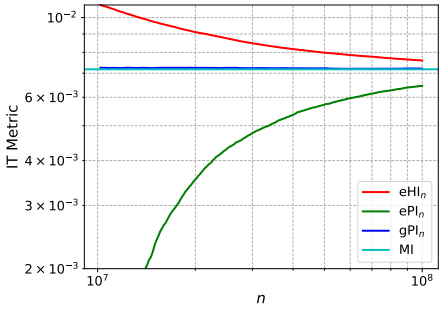


(b) SNR = 0.1 (lin. scale).

Figure 4.2: Simulated experiments, unprotected Sbox, high & medium SNR, univariate.

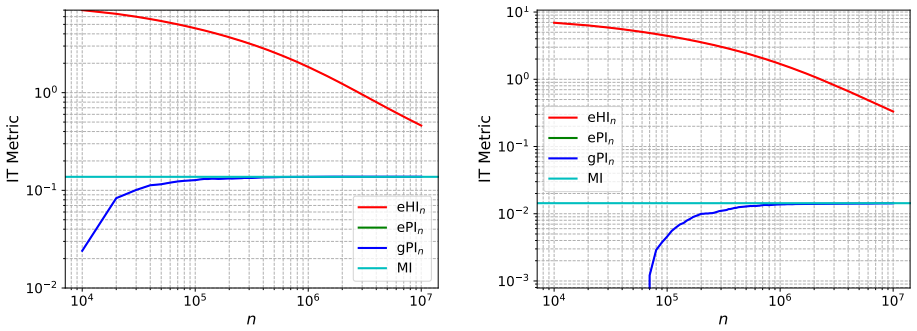


(a) SNR = 0.01 (lin. scale).



(b) SNR = 0.01 (log. scale).

Figure 4.3: Simulated experiments, unprotected Sbox, low SNR, univariate.



(a) SNR = 0.1 (log. scale). **(b) SNR = 0.01 (log. scale).**

Figure 4.4: Simulated experiments, unprotected Sbox**, medium & low** SNR**, bivariate.**

Bivariate unprotected. We complement this observation with experiments corresponding to our second scenario context. As illustrated in Figure 4.4 for medium and low SNR, this more challenging context leads to considerably less tight bounds, which can be explained by the (much) slower convergence of multivariate histograms. Note that we could not reach a positive ePI with $n = 10^7$ in this case (and the gPI still does it rapidly).

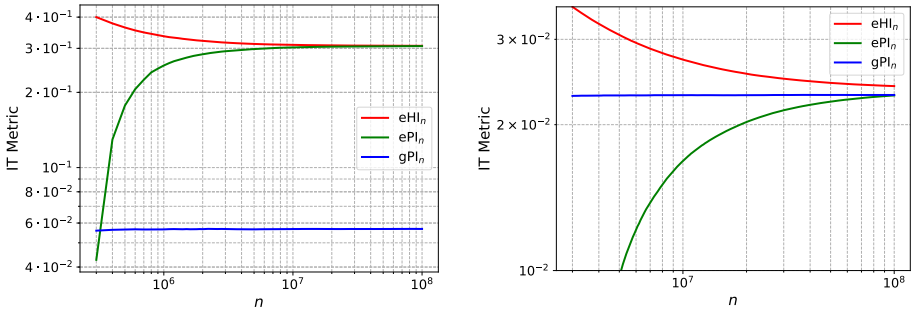
Univariate two shares. We finally report the results of the third scenario with simulated masked implementation in Figure 4.5 for very high and high SNR. The very high SNR case is intended to illustrate a context where the Gaussian assumption is not satisfied (since the masked leakage distribution is actually a Gaussian mixture), so that the gPI is considerably lower than the ePI. By contrast, and as observed (for example) in [GSP13], Figure 1 (right), this Gaussian approximation becomes correct and the gPI gets close to the ePI as the noise increases, which we also see on the right part of Figure 4.5.

An open source code allowing to reproduce these different results is available.[2]

4.4.2 Real traces experiments

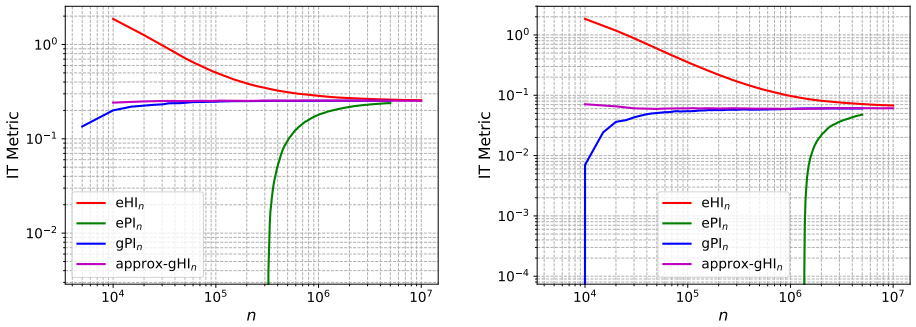
We complement the previous simulated experiments with analyzes performed on actual measurements obtained from an FPGA implementation of the AES Sbox. In order to instantiate a noise parameter as in our simulations, we consider different architectures

[2] `https://github.com/obronchain/Leakage_Certification_Revisited`



(a) SNR = 10 (log. scale). **(b) SNR = 1 (log. scale).**

Figure 4.5: Simulated experiments, masked Sbox**, very high & high** SNR**, univariate.**



(a) $\kappa = 0$ **(log. scale).** **(b)** $\kappa = 7$ **(log. scale).**

Figure 4.6: Actual measurements, unprotected Sbox**, univariate.**

for this purpose: the target Sbox is computed in parallel with $\kappa \in \{0, 3, 7, 11)$ other Sboxes whose computations (for random inputs) generate "algorithmic noise". We implemented our design on a SAKURA-X board embedding a Xilinx Kintex-7 FPGA. The target device was running at 4 MHz and sampled at 500 Ms/s (i.e., 125 leakage points per cycle).

The eHI/ePI bounds computed for the most informative univariate leakage points of our measurements for $\kappa = 0$ and $\kappa = 7$ are in Figure 4.6. We again observe that it is possible to obtain reasonably tight bounds (e.g., to bound the MI below 10^{-1} which is a sufficient noise for the masking countermeasure to be effective). Yet, as κ increases and the MI decreases, we also see that tightening the bounds becomes increasingly data-intensive.

4.5 Limitations and possible improvements

Such an upper bound for MI is of great interest for worst-case proof based evaluations. Indeed it allows to lower bound the worst-case attack data complexity. Yet, to be practical this bound must be relatively tight to reflect the actual security. Otherwise, the estimated worst-case attack complexity derived from the eHI will be significantly lower than the real one.

Unfortunately, the exhaustive models (based on histograms) used to derive eHI have a slow convergence compared to parametric models as illustrated with Figure 4.3. Additionally this convergence speed decreases exponentially with the number of dimensions considered. This can be observed by comparing the tightness of eHI with MI in Figure 4.3 and Figure 4.4.

Since side-channel adversaries usually exploit highly multivariate leakages, the upper bound (i.e., eHI) should ideally be reflecting such a possibility. While it is a good step in this direction, eHI does not completely fulfills this goal. To bind this gap, the evaluator should use a model that has both a fast(er) convergence and has the guarantee to converge to the true distribution, which is unlikely to be possible. An alternative would be to rely on a bound of the form $\mathsf{HI} > \mathsf{PI}$. Even if the relation to MI is lost, and so to the worst-case attacks, it would allow to bound the information that can possibly extracted by an adversary following a predefined strategy.

In the next chapters, we will illustrate the effectiveness of various leakage modeling based on these information theoretic metrics.

Part II

(Close to) worst-case adversaries

Why consider worst-case attacks?

5

This chapter is based on the works *"Efficient Profiled Attacks on Masking Scheme, Extended"* and *"Give Me 5 Minutes: Attacking ASCAD with a Single Side-Channel Trace"* [BCS21] that are under submission.[a]

Over the years, it appeared that the research community (and reviewers) do not always consider worst-case attacks as being of interest for defining the security of implementation. Therefore in this chapter, I take the time to classify various profiled adversaries. I then put forward that worst-case adversaries essentially ease the profiling phase while the asymptotic attacks (in term of profiling complexities) are equivalent.

Then, I illustrate that worst-case attacks are a useful shortcut by evaluating the ASCAD dataset which is used as a reference for deep-learning based attacks. As a result, my analysis reduces the profiling complexity (time and data) as well as the attack efficiency, setting a clear performance target for other types of adversaries such as deep-learning in closed-source settings.

[a] Co-authors: Gaëtan Cassiers, François Durvaux, Loïc Masure and François-Xavier Standaert.

5.1 Contextualization

Research problem. Worst-case side-channel attacks essentially require to have a perfect understanding of the target device, usually reflected in the literature as a *leakage model*. Indeed, in case an exact leakage model is available, it is possible to perform an attack that maximizes the likelihood of the target key given the observed leakages [DSV14, HRG14, BGHR14, BHM+19]. In this context, one recurrent source of discussion in the literature relates to the definition of relevant adversarial capabilities. To give one prominent example, side-channel attacks can be *profiled* — in which case the adversary

can use a device she controls in order to estimate an accurate leakage model from collected measurements, or non profiled – in which case the model is based on engineering intuition. Whenever trying to estimate the worst-case security level, the profiled setting is preferable, since there is no generic attack strategy that can succeed against any device without profiling [WOS14]. This has led Lerman and Markowitch to initiate a study of efficient profiled attacks against masking schemes as a central ingredient for the security evaluation of cryptographic implementations, and to propose a systematic comparison of parametric and non-parametric distinguishers for this purpose [LM19].

Contributions. In this chapter, we therefore extend the results of Lerman and Markowitch with the following contributions:

1. We use a PI approach in order to efficiently assess the profiling complexity and the attack complexity of different tools [SMY09, SKS09].

2. We contextualize the attack settings based on adversarial capabilities and consider profiled attacks with known or unknown masking randomness. We provide a classification of the profiled distinguishers according to the context and the hypothesis made about the leakage distribution. The methodology is similar to the one presented in Section 4.4 but with additional distinguishers.

   As a result, we first illustrate with simulations that the knowledge of the randomness does not necessarily change the asymptotic attack complexity but ease the profiling stage.

3. We confirm these observations on real traces by performing (close to) worst-case attacks on ASCAD and compare our results with deep-learning based attacks. That is, thanks to the knowledge of the randomness, we outperform state-of-the-art closed-source attacks.

5.2 Profiled distinguishers

We now detail various methods to estimate a model PMF $\hat{\mathsf{p}}(x|\boldsymbol{l})$ or PDF $\hat{\mathsf{f}}(l|x)$ from profiling samples $(\boldsymbol{l}, \{x\})$.[1] These estimations attempt to model as closely as possible the PDF of a masked implementation that is described with Equation 2.8.

[1] Recall that the PMF can be derived from the PDF using Bayes, Equation 2.13.

5.2.1 Gaussian mixture template attack (GMTA)

A first solution to build a model is simply to estimate all the PDFs in Equation 2.8. The complete estimated conditional PDF of x is then expressed as:

$$\hat{\mathsf{f}}(\boldsymbol{l}|x) = \sum_{\{x\}} \mathsf{p}(\{x\}) \cdot \hat{\mathsf{f}}'(\,\boldsymbol{l}\,|\,\{x\}\,), \tag{5.1}$$

where every $\hat{\mathsf{f}}'(\cdot|\cdot)$ is the estimated conditional PDF for one component of the mixture. In this work, we propose to use the usually considered Gaussian approximation for this PDF and next call the resulting distinguisher GMTA.

We note that this approach is computationally intensive. During the profiling phase, a template must be built for every possible encoding $\{x\}$. This increases the total number of templates to $|X|^d$ where $|X|$ is the number of possible values for x. During the attack phase, in order to recover $\hat{\mathsf{f}}(\boldsymbol{l}|x)$, the mixture must be explicitly computed and therefore requires $\mathcal{O}(|X|^d)$ operations per attack trace.

5.2.2 Multi-layer perceptron (MLP)

A Multi-layer perceptron aims at directly approximating the discriminative model $\hat{\mathsf{p}}(x|\boldsymbol{l})$ – seen as a multivariate function of l – thanks to a composition of elementary functions (a.k.a. *layers*), alternating between *linear* layers (denoted as λ), non-linear element-wise *activation functions* (denoted as σ), and a final *softmax* normalization layer (denoted as s). More precisely, in this study we use MLPs with only one intermediate layer, as follows:

$$\hat{\mathsf{p}}(x|l) = \mathrm{s} \circ \lambda_{|\mathrm{X}|} \circ \sigma \circ \lambda_C(l) \ , \tag{5.2}$$

where C denotes the number of neurons in the hidden layer, i.e., the output dimension of the first linear layer.

Each layer is fully described by inner parameters whose values are tuned during the profiling phase. This step is done by maximizing the likelihood of the model outputs given the profiling data providing the ground truth.

Such models are known to be very expressive, since the approximation error can be made arbitrarily small by increasing C [Pin99]. Thus, MLPs represent a powerful tool when no further hypothesis can be made on the true leakage model to approximate. As a drawback, they may require more profiling data to tune their parameters.

5.2.3 Encoding soft analytical side-channel attack (ESASCA)

In order to limit the computational complexity of GMTAs, an alternative solution is to express every $\hat{\mathsf{f}}'(\cdot|\cdot)$ from the (independent) leakage on each of the shares in $\{x\}$, leading to the following expression:

$$\hat{\mathsf{f}}(\boldsymbol{l}|x) = \sum_{\{x\}} \mathsf{p}(\{x\}) \cdot \prod_{j=0}^{d-1} \hat{\mathsf{f}}'(\boldsymbol{l} \,|\, x^j), \tag{5.3}$$

where $\hat{\mathsf{f}}'(\boldsymbol{l}, x_i)$ is the PDF estimated for the share x_i. The latter can be viewed as an application of Soft Analytical Side-Channel Attacks (SASCA) limited to an encoding rather than an entire circuit [VGS14]. This strategy has demonstrated its interest in the context of masked software implementations [BS21] (see Chapter 7). We next denote it as Encoding-only SASCA (ESASCA). In the specific case where $\hat{\mathsf{f}}'(\boldsymbol{l}, x_i)$ is estimated with a Gaussian distribution, we call it G-ESASCA; in the case MLPs are used, we call it MLP-ESASCA.

The main advantage of this strategy is that during profiling, the PDFs do not need to be estimated on the entire encoding but only on the shares independently. As a result, the number of templates is reduced down to $d \cdot |X|$. The advantages and disadvantages of ESASCA are further discussed in Section 7.3.

Its main drawback is that by making such an independent estimation of the leakage PDF for each component, ESASCA is unable to detect flaws due to physical defaults (like glitches or couplings) that could reduce the statistical security order of the implementation (defined as the highest statistical moment of the leakage distribution that is independent of the secret). We call such a strategy *order-preserving* to reflect the fact that it targets the maximum statistical security order of the masking scheme.

5.2.4 Expectation-maximization (EM)

The last approach we study is based on the EM algorithm [Moo96, LP07b]. It is an iterative procedure that allows estimating the parameters of a mixture of Gaussian PDFs and can therefore be used to model Equation 2.8. The parameters that the EM algorithm has to estimate are the means and the covariances of each of the components $\hat{\mathsf{f}}(\boldsymbol{l}|\{x\})$, the weight of each component being known and equal to $\{\mathsf{x}\}$. During the profiling phase, one mixture must be estimated per possible x meaning that EM must be executed $|X|$ times. Each EM execution

has to model a mixture with $|X|^{d-1}$ modes. During the attack phase, the mixture must then be explicitly computed as in a GMTA, leading to a computational complexity of $\mathcal{O}(|X|^d)$. However, the adversary / evaluator can decide to model a smaller number of components leading to more (computationally) efficient and heuristic attacks.

5.3 How to compare two distinguishers?

In this section, we detail the methodology we used in order to compare profiled side-channel distinguishers against masked implementations. We first describe our simulated attack framework and follow with a description about how we use the PI metric to compare distinguishers.

5.3.1 Simulation framework

Our experimental analyzes are based on simulated leakages. The rationale behind this choice is similar to the one of Lerman and Markowitch [LM19]. Since we aim to compare distinguishers, it allows us to evaluate them in different well-controlled scenarii (e.g., low-noise, high-noise), which considerably simplifies the interpretation of the results. We note however that all the distinguishers we consider have been applied to real traces in previous works (e.g., [PSDQ05] for GMTA, [MDP20] for MLP, [BS21] for ESASCA and [LP07a] for EM).

More precisely, we first consider a simulated masked (flawless) implementation as graphically represented in Figure 5.1. To simulate the leakage of a secret κ-bit variable x, $d-1$ first shares are drawn at random from the set of all κ-bit strings $\{0,1\}^\kappa$. The last share is set with $x^{d-1} = x \oplus \sum_{j=0}^{d-2} x^j$ in order to ensure that $x = \sum_{j=0}^{d-1} x^j$. The leakage vector $\boldsymbol{l}$ is then defined as the concatenation of all the leakage $l^{x^j} = \mathsf{hw}(x^j) + \eta_j$ where $\mathsf{hw}(\cdot)$ is the Hamming weight function and η_j is a Gaussian noise with variance σ^2.

Since some of the distinguishers we consider are order-preserving, we additionally consider the case of a flawed masked implementation, in order to determine the security gaps that may appear when targeting the maximum security order despite physical defaults. We use the setting of [DFS15] for this purpose, and assume a 2-share implementation with a first-order flaw. In this case, the simulated leakages additionally contain a value $l^x = f \cdot \mathsf{hw}(x) + \eta$. The noise η is again a Gaussian random variable with variance σ^2. The parameter f captures the amplitude of the flaw (or its SNR). If $f = 0$, the independence assumption is met. As f increases, it is expected that the first-order leakage will be more and more dominating over the second-order one.

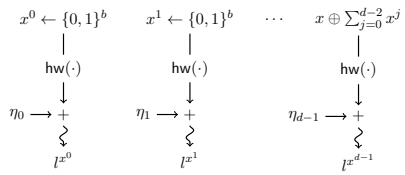


Figure 5.1: Simulation of masked a implementation.

Overall, the proposed simulations take three parameters. The number of shares d, the noise variance σ^2 and the first order-flaw magnitude f. When relying on simulations, these parameters are known and so is the true leakage PDF $\mathsf{f}(\boldsymbol{l}|x)$ that is defined by Equation 2.8. As discussed in Section 2.2, knowing this PDF allows us to compute the MI which is representative of the best-possible attack. In the next sections, it will enable us to compare the PI of the proposed distinguishers to this optimal MI value.

5.3.2 PI as a comparison metric

In this chapter, we make use of information theoretic metrics in order to compare profiled distinguishers. For this purpose, we denote as $\widehat{\mathsf{PI}}^{\mathsf{f}}_n(\mathrm{X}, L)$ the PI of the model for which n profiling measurements are used to estimate the PDF $\hat{\mathsf{f}}(l|x)$. We use a similar notation for the PMF $\hat{\mathsf{p}}(x|l)$. We then consider the following two criteria:

Profiling complexity. The first way to compare distinguishers is to evaluate their performances by fixing their profiling data complexity (i.e., the number of traces acquired during the profiling phase) to n. Comparing the PI values in this case answers the question: *"For a given data complexity n, what is the best model for the adversary?"*. That is, since having a larger PI implies an online attack with lower number of traces, we can deduce that if:

$$\widehat{\mathsf{PI}}^{\mathsf{f}_1}_n \leq \widehat{\mathsf{PI}}^{\mathsf{f}_2}_n, \tag{5.4}$$

then the best strategy for the adversary is to use $\widehat{\mathsf{f}_2}(\cdot \,|\, \cdot)$. We stress that this comparison is conditional to n: a tool may be better for some a small n and not for a larger one. A natural way to compare profiling complexity is to extract the n values needed to reach a positive PI (i.e., a model that is sufficiently accurate to allow key recoveries).

Online attack complexity. The second way to compare distinguishers is to analyse the asymptotic PI value that is reached by a model. For this purpose, we denote as $\mathsf{PI}^{\widehat{\mathsf{f}}}_{\infty}(X, \boldsymbol{L})$ the PI of a model for which n is sufficiently large for perfectly estimating all its parameters, which we assume to be the maximum reachable one. This value then indicates the complexity of the best online attack that can be performed with this model. Applied to a pair of models, it typically leads to inequalities like:

$$\mathsf{PI}^{\widehat{\mathsf{f}_1}}_{\infty} \leq \mathsf{PI}^{\widehat{\mathsf{f}_2}}_{\infty} \leq \mathsf{MI}. \tag{5.5}$$

From an evaluator's viewpoint, it answers the question: *"What is the best attack that can be performed if my profiling phase is sufficient for my model to converge?"*.

5.4 Distinguisher classifications

Before performing quantitative (experimental) comparisons in the next section, we propose a systematic contextualization of the distinguishers we study. We consider both the tools studied by Lerman and Markowitch in [LM19] and the new ones we list in Section 5.2. The resulting classification is summarized in Table 5.1. We next develop it by first highlighting the importance of (masking) randomness knowledge during profiling and then discussing assumptions on the PDF that may reduce the profiling complexity, possibly at the cost of less efficient online attacks.

	Known masks	Gaussian distri.	Gaussian comps.	Order-preserving
MLP	✗	✗	✗	✗
RF	✗	✗	✗	✗
KDE	✗	✗	✗	✗
EM	✗	✗	✓	✗
GMTA	✓	✗	✓	✗
G-ESASCA	✓	✗	✓	✓
MLP-ESASCA	✓	✗	✗	✓
HO-GTA	✗	✓	✗	NA

Table 5.1: Classification of distinguishers according to the *a priori* assumptions made on the leakage distribution. ✓ means that the given algorithm relies on the given assumption to work, whereas ✗ means that the given assumption is not mandatory to run the given algorithm.

5.4.1 Randomness knowledge

Allowing an adversary / evaluator to know the randomness used during the profiling stage of a side-channel attack generally allows estimating a model with a smaller profiling data complexity n. Yet, an important question is to determine whether such an assumption is only a useful shortcut for evaluators (as promoted in [ABB+20]) or if it also creates a complexity gap in the online attacks. In other words, does this randomness knowledge only speed up evaluations, leading to online attacks that could also be reached by determined adversaries without this shortcut (with more profiling), or are there realistic examples where profiling without masks knowledge cannot lead to the same online attacks as profiling with masks knowledge?

In order to contribute to this question, we evaluate two types of distinguishers. The first type does not require the knowledge of the randomness during profiling, and directly builds an estimate $\hat{\mathsf{f}}(\cdot|\cdot)$ for the full PDF $\mathsf{f}(\cdot|\cdot)$, or $\hat{\mathsf{p}}(\cdot|\cdot)$ for the full PMF $\mathsf{p}(\cdot|\cdot)$. The MLP, EM, Random Forest (RF) and Kernel Density Estimation (KDE) distinguishers typically fall in this category.[2] The same holds for Higher-Order Gaussian Template Attacks (HO-GTA) such as [PRB09, MS16]. The second type of methods does require the knowledge of $(l, \{x\})$ during profiling, in order to decompose $\mathsf{f}(\cdot|\cdot)$ in multiple simpler PDFs $\mathsf{f}'(\cdot|\cdot)$. Indeed estimating $\mathsf{f}'(\cdot|\cdot)$ for each of the components in the mixture requires to know which component is the one corresponding to the profiling trace l. The GMTA, G-ESASCA and MLP-ESASCA distinguishers are representatives candidates of this second category.

5.4.2 A priori PDF assumptions

Another natural way to speed up the profiling of a model (and possibly to make the online distinguishers more efficient) is to rely on good a priori assumptions on the leakage PDF $\mathsf{f}(\cdot|\cdot)$. As a counterpart, if these assumptions are not correct, the estimated distribution may not converge to true PDF so that $\mathsf{PI}_\infty < \mathsf{MI}$, indicating that an online attack using this model will be suboptimal. We next detail two assumptions that we will consider in our experiments.

Gaussian components. We first note that assuming a Gaussian distribution is common when targeting unprotected devices [CRR02]. However, when the masking countermeasure is implemented, it is expected that this assumption is not valid anymore, since the masking

[2] For the last two ones, we refer to [LM19].

randomness turns the distribution into a mixture. A natural extension is to assume that the distribution of each component in the mixture is Gaussian so that:

$$\mathsf{f}'(\boldsymbol{l}|\{x\}) \approx \mathcal{N}(\boldsymbol{\mu}_{\{x\}}, \boldsymbol{\Sigma}_{\{x\}}). \tag{5.6}$$

As in the unprotected case, the interest of this assumption is that it reduces the parameters that must be estimated during profiling to only a mean vector $\boldsymbol{\mu}$ and a covariance matrix $\boldsymbol{\Sigma}$. Estimating this covariance has a cost that is quadratic in the size of $\boldsymbol{L}$, which is significantly cheaper than non-parametric estimators based on histograms or Kernels (of which the cost grows exponentially in this size). The EM, GMTA and G-ESASCA attacks take advantage of this assumption.

Independent shares' leakages. Eventually, another hypothesis about the leakage PDF that speeds up the profiling of a masked implementation is to assume that all the shares leak independently. As a result, each component of the PDF $\mathsf{f}(\cdot|\cdot)$ can be approximated with:

$$\mathsf{f}'(\,\boldsymbol{l}\,|\,\{x\}\,) \approx \prod_{j=0}^{d-1} \hat{\mathsf{f}}'(\,\boldsymbol{l}\,|\,x^j\,), \tag{5.7}$$

where one single PDF $\hat{\mathsf{f}}'(\,\boldsymbol{l}\,|\,x^j\,)$ must be estimated per share. The main interest of this assumption is to scale gently with the number of shares. Namely, and as detailed in Subsection 5.2.3, the number of templates to estimate grows linearly with the number of shares, while it grows exponentially for methods like GMTA (which also makes its computation during the online attack computationally intensive as the number of shares increases) [GS18].

It is important to note that the independent leakage assumption is usually considered as a *design assumption* that engineers implementing masking have to fulfill [DFS15, NRS11, FGP+18]. Failing to meet this assumption may lead the worst-case security level of an implementation to be lower than expected. By contrast, we here consider it as a *distinguisher assumption*. So analyzing an implementation under this assumption when it is not fulfilled may lead to a false sense of security (i.e., to the PI extracted with a model exploiting this assumption being lower than the MI). In particular, if the shares of a masked implementation do not leak (sufficiently) independently due to a physical default like glitches or couplings, an adversary using this assumption will not be able to detect and exploit this flaw. For this reason, we use the terminology "order-preserving" for the distinguishers assuming independent shares' leakages in Table 5.1. In this work, only the

ESASCA-based attacks (i.e., G-ESASCA and MLP-ESASCA) are doing such an hypothesis.

5.5 Experimental (simulated) results

We now move to the presentation of our simulated experiments. For this purpose, we first selected representative distinguishers from the categories of Table 5.1, namely MLP, EM, GMTA, G-ESASCA and MLP-ESASCA. We excluded the RF and KDE distinguishers because they consistently led to significantly more computationally intensive attacks than the MLP-based one while not providing better results in terms of data complexity. We also did not consider HO-GTAs in our comparisons since they are based on the estimation of statistical moments rather than full distributions and we see their motivation as the assessment of a statistical security order rather than as efficient attacks.[3] We structure the section in two main parts. First, we compare the previously listed profiled distinguishers in a setting corresponding to a "properly implemented" masked implementation (i.e., without flaw) with 2 and 3 shares. We use this experiment to discuss the profiling complexity and online attack complexity of attacks against the masking countermeasure, in function of the implementation context and the assumptions used by the distinguishers. Next, we evaluate the extent to which the presence of a flaw in the masking can lead the order-preserving distinguishers to overstate the security level of an implementation in Subsection 5.5.2.

5.5.1 Flawless masked implementation

The convergence of the PI metric estimated for our investigated distinguishers is given in Figure 5.2 for a 2-share implementation, with SNR of 10, 1 and 0.1 and in Figure 5.3 for a 3-share implementation with SNR of 10 and 1. It leads to the following main three observations:

1. The value of the asymptotic PI reaches the value of the MI for all the investigated distinguishers. This implies that all these distinguishers can lead to worst-case attacks if profiled with a sufficient amount of traces. This observation is naturally explained by the fact that the assumptions exploited by some of these distinguishers are all fulfilled in this first simulated setting.

[3] It is for example shown in [Sta18] that they can require significantly more data to attack than a distribution-based distinguisher in low noise conditions, while it is shown in [SVO+10] that they reach the same data complexity as distribution-based distinguishers in high-noise conditions.

2. By contrast, the speed of convergence of the different distinguishers, and therefore their profiling complexity, significantly varies. As expected, the distinguishers that have the lowest profiling (data) complexity are also the ones that take advantage of more assumptions. Concretely, we observe that the G-ESASCA is the fastest, followed by MLP-ESASCA, GMTA, MLP and EM.

3. The ordering of the distinguishers in terms of profiling complexity is independent of the SNR and number of shares, but the quantitative gap between them increases as the SNR decreases and the number of shares increases (i.e., for better protected implementations).

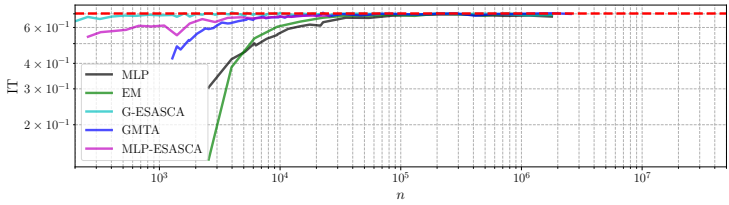
The first point is particularly important regarding the question raised in Subsection 5.4.1. Namely, it shows that for the flawless masked implementation simulated in this section, a determined adversary profiling without masks knowledge can reach the same attack efficiency as an evaluator leveraging this knowledge. So it confirms the masks knowledge as a useful shortcut for evaluations which does not affect the final security claims in terms of online attack complexity.

5.5.2 Flawed masked implementation

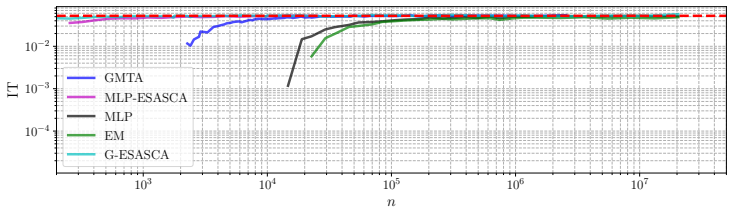
We complement the previous analysis with the case of a flawed masked implementation, in order to determine the extent to which the order-preserving distinguishers can overstate security. We focus on the online attack complexity reflected by the asymptotic PI value in this case, since a flaw in a masked implementation is not expected to affect our conclusions regarding the profiling complexity of the distinguishers.

We first illustrate the possibility of a security overstatement in Figure 5.4. It shows that while the order-preserving G-ESASCA and the MLP distinguishers both have an asymptotic PI equal to the MI independent of the SNR when there is no flaw in the masked implementation (in the left part of the figure), the presence of a flaw makes the PI of the G-ESASCA distinguisher significantly lower than the MI in the presence of a flaw (in the right part of the figure).

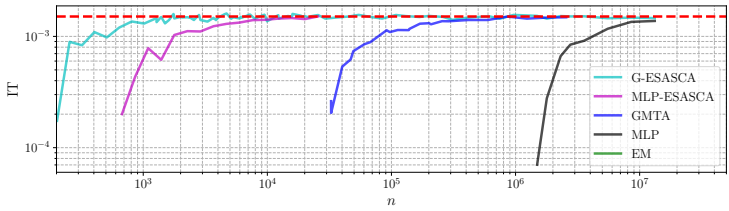
We then systematize this investigation in Figure 5.5, which shows how the security overstatement of the G-ESASCA distinguisher increases with the amplitude of the flaw (i.e., the f parameter), while the MLP distinguisher preserves an optimal PI value independent of the f parameter.



(a) SNR = 10

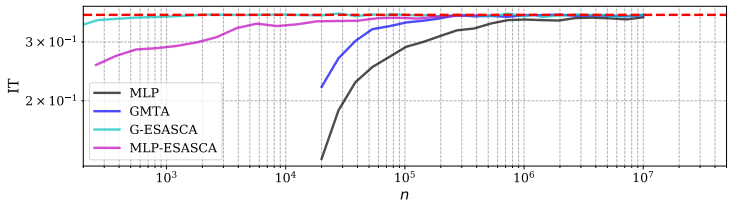


(b) SNR = 1

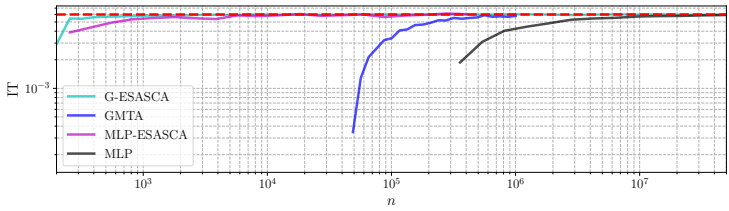


(c) SNR = 0.1

Figure 5.2: Flawless 2-share implementation profiling.

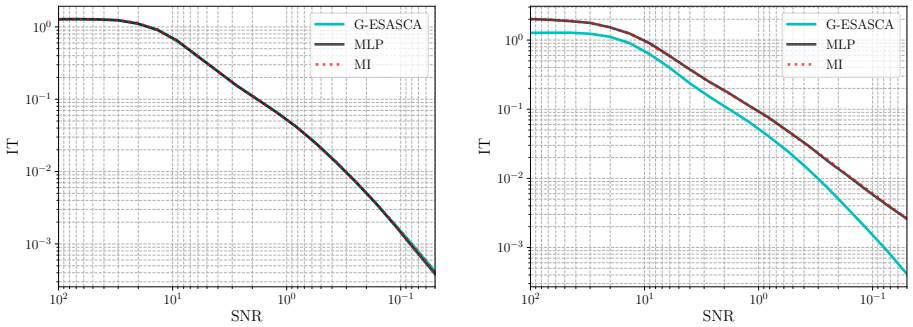


(a) SNR = 10



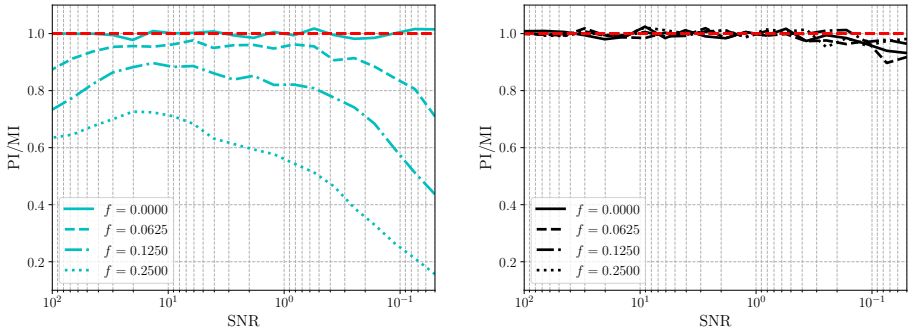
(b) SNR = 1

Figure 5.3: Flawless 3-share implementation profiling.



(a) $f = 0.0$ **(b)** $f = 0.25$

Figure 5.4: Flawed 2-share implementation: asymptotic PI.



(a) G-ESASCA **(b) MLP**

Figure 5.5: Flawed 2-share implementation: asymptotic PI/MI.

Overall, the results in this section confirm that while using the masking randomness in an evaluation is sound for the investigated case studies, the order-preserving assumption can lead to security overstatements. Whenever used in the evaluation of a leaking implementation, it is therefore important to confirm in parallel that it is sufficiently fulfilled (i.e., that the f parameter is low enough), for example using moment-based detections or attacks [SM16, MS16] or, more formally, by relying on leakage certification [BHM+19].

5.6 Exploiting randomness against ASCAD

Next, we illustrate the simulations of the previous section by putting forward the advantage of randomness knowledge on real measurements. That is, we propose a (close to) worst-case attack against ASCAD dataset with the evaluations tools developed in the next sections of this thesis.

5.6.1 ASCAD dataset

Since 2018, ASCAD has become one of the most used datasets for side-channel analysis benchmarking [BPS+20]. It contains power measurements of a two-share masked AES implementation running on an 8-bit micro-controller. As intended by the authors, most of the published attacks against ASCAD use Deep Learning (DL) – we refer to the publications using this dataset at CHES 2020 for illustration [ZBHV20, MDP20, WAGP20, WP20, PCP20, HHO20, ZZN+20, ZDF20]. Most of these works focus on recovering a single key byte by looking at a small (possibly desynchronized) part of the leakage traces, and do not exploit the knowledge of the masking randomness during their profiling stage. An exception is the recent work of Xiangjun Lu et al. [LZC+21].

5.6.2 Close to worst-case attacks against ASCAD

Attack strategy. Next, we apply the methodology presented in Section 7.3 implemented with SCALib to analyze ASCAD. The latest is a generalization of G-ESASCA exploiting multiple encodings at once. It will be applied in Chapter 7 against masked bitslice 32-bit software. This attack takes advantage of all the time samples in the leakage traces (next called raw traces) and leverages the knowledge of the masking randomness during the profiling phase. It mixes SNR computations, Linear Discriminant Analysis (LDA) [SA08] and SASCA [VGS14].

Results. We first ran our attack on the full traces (i.e., 250,000 time samples). We used 5000 traces for profiling (i.e., computing the SNR and building models for) the 86 different shares. The attack was then run 100 independent times using $N = 1$, then $N = 2$ attack traces. We then computed the resulting rank of the 14 byte masked key. The results are reported in Figure 5.6, and show that even with a single trace, the attack always succeeds. The total execution time (for both profiling and attacking) is less than 4 minutes on a 4-core laptop (16 GB RAM, Intel(R) Core(TM) i7-8650U CPU @ 1.90GHz).

For the sake of completeness, we also report in Figure 5.7 an attack on the third byte exploiting only a few samples (1400) of the traces, as usually considered in previous works. As before, we run the attack 100 times for each number of attack traces. This attack uses 100,000 traces for the profiling and the total execution time is 12 seconds. We can see that if all bytes were giving similar results, 8 attack traces would be enough to perform a full key recovery attack. As expected,

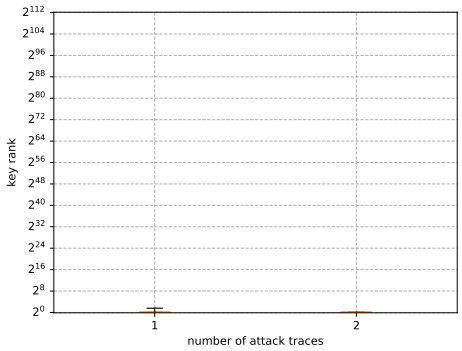


Figure 5.6: Attack against full ASCAD raw traces for the full (14 byte) key: boxplot of the rank of the correct key. Attacks with two traces always result in finding the correct key at rank 1. With only one attack trace, the correct key is always at very low (i.e., trivially enumerable) rank, and more than 0.75 % of the times at rank 1.

due to the reduction in the number of leakage points, this attack is less powerful than the one exploiting the full trace.

5.7 Implication for evaluators

The full key recovery attack described above significantly improves the state of the art, both in terms of profiling cost (data and time) and in terms of online attack complexity. At high level, efficient profiling is obtained thanks to a good exploitation of the masking randomness while the improved online attack is primarily due to an analytical strategy that takes advantage of the complete (raw) traces.

One of the long-standing open problems in the evaluation of cryptographic implementations against side-channel attacks is to determine what are the adversarial capabilities to consider. In the context of masked implementations, this question is typically reflected by a dilemma regarding the knowledge of the shares during profiling. On the one hand, knowing this randomness can significantly speed up the evaluations. On the other hand, and to the best of our knowledge, the question whether it leads to a gap between online attacks that can be performed in an evaluation context and more concrete attacks profiled without access to this randomness remains open. In this chapter, we first contribute to this issue and show that for realistic implementations, determined adversaries can reach the same (worst-case) attack complexities as evaluators, with only a penalty in profiling

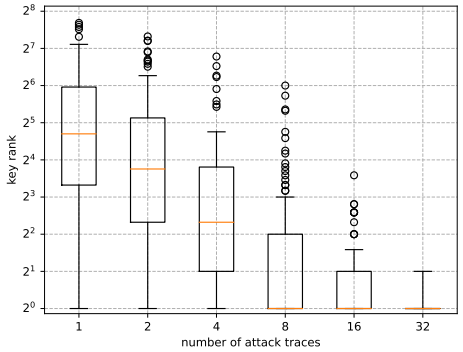


Figure 5.7: Attack against partial ASCAD traces for one key byte: boxplot of the rank of the correct key. Attacks with 32 traces almost always result in finding the correct key at rank 1. With only 8 traces, the correct key is at rank 1 more than half of the times.

complexity. Therefore in the rest of this thesis, we will then constantly exploit the knowledge of the randomness.

We complement this main conclusion with a systematization effort and evaluate the impact of other solutions that may simplify the evaluation problem, by positing realistic assumptions on the leakage distribution. We show that such assumptions are in general useful from the profiling complexity viewpoint but (as theoretically known), can lead to a false sense of security in case the actual leakages significantly deviate from the evaluator's assumptions. We therefore propose a classification of profiled distinguishers in function of the assumptions they make, which we hope can help evaluators selecting the appropriate statistical tools in function of the implementations to analyze.

Countermeasures's dissection

6

This chapter is based on the work *"Side-Channel Countermeasures Dissection and the Limits of Closed Source Security Evaluations"* published in TCHES 2020 [BS20].[a]

The initial motivation for this work was the preliminary report provided along an open-source masked implementation by the ANSSI who exhibits high SNR on some of the shares. It was my first evaluation of a masked software exploiting the knowledge of the randomness. It then became the main direction of my thesis as it appeared to be large gap between standard evaluations and the results obtained by my analysis.

Eventually, we note that since the publication of the research paper, the ANSSI released measurements of the implementation. An attack leveraging a similar (dissection) approach and using deep-learning for information extraction has also been published [MS21].

[a] Co-author: François-Xavier Standaert.

6.1 Contextualization

Motivation. Considering the general context of this thesis pushing for open-source implementations, the publication of an AES protected by a team from the French ANSSI (Agence Nationale de la Sécurité des Systèmes d'Information), was a very welcome initiative [BKPT19]. It contains a mix of countermeasures, purposed for an accessible COTS device, namely an `STM32` chip that is based on an ARM `Cortex-M4` architecture.

Besides the effort in publishing an open source library mixing state-of-the-art countermeasures aimed at a minimum security level, the ANSSI implementation also comes with a preliminary leakage assessment that we can hope is reflective of the first (minimum)

steps that an evaluation laboratory would perform, namely SNR computations [Man04] and basic (1st-, 2nd- and 3rd-order) divide-and-conquer attacks [SVO+10]. So while this implementation was not claimed to be designed to pass certification nor to be usable for commercial applications, it is at least a useful starting point for research purposes since it was developed and analyzed by a team of experts with both academic and industrial experience.

Goals. More specifically, this implementation can be used to guide research towards a better understanding of three important questions, namely (*i*) how to evaluate such a mix of countermeasure in a (close to) worst-case manner; (*ii*) which security levels can be reached by software implementations with low number of shares in (e.g., ARM-based) COTS devices (that are typical targets for deployment in IoT applications); (*iii*) to what extent the open-source approach helps in evaluating the security of the implementation.

Contributions. Based on this state-of-the-art, we tackle the aforementioned questions related to the security evaluation of protected cryptographic implementations. The contributions in this chapter are the following:

1. We use knowledge of the implementation to perform countermeasures' dissection. By dissection, we mean a detailed analysis of each ingredient of a protected implementation, in order to limit their (combined) impact as much as possible. We note that from a complexity viewpoint, a dissection attack is not equivalent to a divide-and-conquer one. Divide-and-conquer attacks turn a multiplication of complexities into a sum. This is what happens when performing 16 independent DPAs against the 16 key bytes of the AES (which has a time complexity $\propto 16 \cdot 2^8$) rather than performing an exhaustive search on the full key (which has a time complexity of 2^{128}). In a dissection attack, the complexities of different countermeasures are still multiplied, but the adversary aims at keeping the attack factors as close to one as possible.

2. We demonstrate that the implementation can be broken with $N = 900$ measurements which can be considered as a low complexity. This hints into the direction of a too low noise in the implementation. Indeed, the proposed dissection attack is order-preserving hence is not able to exploit flaws in the independence

assumption. As a result, the low security comes from the second noise assumption (or to a too low number of shares).

3. We finally illustrate the advantage of open-source evaluation. Indeed without the knowledge of randomness and the underlying affine masking scheme (hence in a black-box manner), we show that MLP-based adversaries cannot easily recover secrets. As discussed in the introduction of this thesis, this can lead to a false sense of security. Because this part has a similar conclusion as Chapter 5, we answer this question in Appendix B.

6.2 Target implementation

In this chapter, we describe the details of the masking and shuffling countermeasures implemented in the ANSSI's open-source library [BKPT19].

6.2.1 Affine masking

In this section, we first recall the masking countermeasures implemented in [BKPT19]. The targeted library implements the affine masking scheme proposed in [FMPR10] to protect the AES. All the operations are performed over the Galois Field $\mathsf{GF}(2^8)$. In such a masking scheme, the sensible value x (i.e., the Sbox output) is encoded thanks to:

$$C(x; r^m, r^a) = (r^m \otimes x) \oplus r^a, \tag{6.1}$$

where $C(\cdot)$ is the encoding function, r^m the multiplicative mask, r^a the additive mask and $\otimes$ a multiplication over $\mathsf{GF}(2^8)$ (instead of a bitwise AND considered in the rest of this thesis). The multiplicative mask is uniformly distributed over $\{1, \ldots, 255\}$ allowing to invert the multiplication. The additive mask is uniformly distributed over $\{0, \ldots, 255\}$.

The main challenge in such a masking scheme is to implement the AES Sbox. Before every encryption, an alternative Sbox$'$ is precomputed depending on two additional additive masks r^{in} and r^{out} as well as r^m. During the encryption, the Sbox is applied to the encoding of x following:

$$C(\mathrm{Sbox}(x); r^m, r^a) = \mathrm{Sbox}'(C(x; r^m, r^a) \oplus r^{in} \oplus r^a) \oplus r^a \oplus r^{out}. \tag{6.2}$$

That is, a look-up table to the alternative Sbox$'$ is made while the additive mask r^a is locally canceled and replaced by the input and

output masks r^{in} and r^{out} (that correspond to the pre-computed table). In the studied implementation, one r^a per byte is required and a single alternative Sbox$'$ is pre-computed for all the bytes, meaning that a single r^m, r^{in} and r^{out} triple is used, leading to randomness requirements per encryption of 19 bytes. In the rest of the chapter, $\boldsymbol{c}$ denotes the encoded 4×4 matrix of the AES state. Similarly, $\boldsymbol{r}^a$ corresponds to the 16 additives masks consistent with the encoding $\boldsymbol{c}$. The `MixColumns` operation can be performed on each column of $\boldsymbol{c}$ and $\boldsymbol{r}^a$ independently.

6.2.2 Shuffling

On top of the previously exposed affine masking, the targeted implementation is using operations' shuffling from [VMKS12] and described in Subsection 2.3.2. Therefore, some sets of independent operations are performed in a randomized order. For the Sbox executions and for `ShiftRows`, permutations $\boldsymbol{\theta}^{\boldsymbol{c}}$ and $\boldsymbol{\theta}^{\boldsymbol{r}^a}$ over the set $\{0, \ldots, 15\}$ are generated based on a 16-bit random seed. For `MixColumns`, smaller permutations $\boldsymbol{\theta}'^{\boldsymbol{c}}$ and $\boldsymbol{\theta}'^{\boldsymbol{r}^a}$ over the set $\{0, \ldots, 3\}$ are generated based on a 2-bit random seed. In the following, we will assume that the first permutations cannot be enumerated while the second ones can trivially (making them similar to the weaker random start index shuffling discussed in [VMKS12]).

A high-level view of how the affine masking and shuffling countermeasures are combined in the ANSSI implementation is given in Figure 6.1. Additionally to the previous notations, $\boldsymbol{p}$ denotes the 16 bytes plaintext.

6.3 Code inspection & countermeasures' dissection

In this section, we discuss different attack vectors that are exploitable in the protected implementation from [BKPT19]. First, a worst-case distinguisher strategy is proposed. Second, the worst-case equation is simplified based on reasonable hypotheses (that we discuss in this section and will validate experimentally in the next sections). This allows a significant improvement of the attacks' time complexity. Third, we dissect the combination of countermeasures and describe how the different ingredients of the ANSSI implementation can be analyzed separately, in order to reduce the attacks' data complexity. Finally, four different attack strategies are proposed for comparison purposes.[1]

[1] We also introduce a fifth one that is using MLP in combination with countermeasure's dissection.

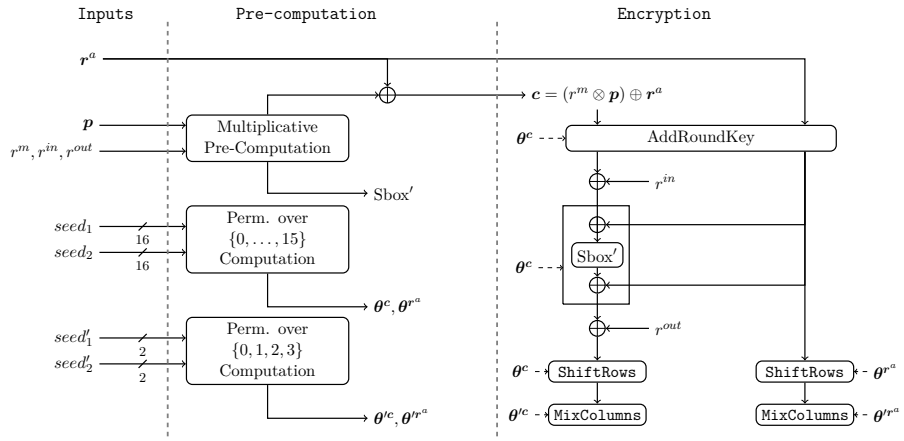


Figure 6.1: Combined (affine masking + shuffling) countermeasures: high-level view.

6.3.1 Worst-case distinguisher

In order to mount a successful attack, a side-channel distinguisher is interested in the PDF of a (key dependent) intermediate random vector $\boldsymbol{X}$ conditioned to the leakage $\boldsymbol{L}$. We first start by the description of the full PDF. Then we describe divide-and-conquer attack strategies.

Full PDF. In the case of affine masking, the secret vector $\boldsymbol{X}$ is protected with two random masks r^m and $\boldsymbol{r}^a$. Similarly to the conditional PDF in Boolean masking (Equation 2.8), the conditional PDF for affine masking is defined with:

$$\mathsf{f}_{msk}(\boldsymbol{l}|\boldsymbol{x}) = \sum_{r^m, \boldsymbol{r}^a} \mathsf{f}(\boldsymbol{l}|r^m, \boldsymbol{r}^a, \boldsymbol{c} = (\boldsymbol{x} \otimes r^m) \oplus \boldsymbol{r}^a) \cdot \mathsf{p}(\boldsymbol{r}^a) \cdot \mathsf{p}(r^m). \quad (6.3)$$

This equation sums over all the possible masks weighted by their (uniform) probability and all the indexes. In the context of [BKPT19], shuffling is combined to affine masking. Similarly to the generic shuffling PDF (Equation 2.10), the true conditional PDF also requires to sum over all the possible set of permutations resulting in:

$$\mathsf{f}(\boldsymbol{l}|\boldsymbol{x}) \propto \sum_{r^m, \boldsymbol{r}^a, \boldsymbol{\theta'}s} \mathsf{f}(\boldsymbol{l}|r^m, \boldsymbol{r}^a, \boldsymbol{c} = (\boldsymbol{x} \otimes r^m) \oplus \boldsymbol{r}^a, \boldsymbol{\theta'}s), \quad (6.4)$$

where the proportional factor allows to omit the uniform probabilities and $\boldsymbol{\theta'}s$ denotes all the four involved permutations (see Figure 6.1).

D&C strategy. However, the above equation can not be used in practical settings. Indeed, it requires to sum over all the possible randomness used in the implementation that is 19 bytes.[2] This forces any adversary to use a D&C strategy which induces a small loss in extracted information compared to the optimal attack (see Subsection 8.3.2, Figure 8.1). In such a D&C attack, the adversary extracts all the bytes x_i in the implementation independently. Namely, first only considering affine masking, it uses the following equation:

$$\hat{\mathsf{f}}(\boldsymbol{l}|x_i) \propto \sum_{r^m, r^a} \hat{\mathsf{f}}(\boldsymbol{l}|r^m, r^a, c = (x_i \otimes r^m) \oplus r_i^a), \tag{6.5}$$

where the exponent i denotes a byte index and once again omitting constant factors. To take shuffling into consideration, the probabilities of the shuffled states also have to be computed by summing over the shuffled cycles (and possibly permutations). A generic expression for estimating the corresponding conditional PDF was given in [VMKS12]. Using our notations, it can be written as:

$$\begin{aligned}\hat{\mathsf{f}}(\boldsymbol{l}|r^m, r_i^a, c_i) \propto & \sum_{o_1} \mathsf{w}(\boldsymbol{\theta}^{\boldsymbol{c}}_{o_1} = i|\boldsymbol{l}^{\boldsymbol{\theta}^c}) \\ & \cdot \sum_{o_2} \mathsf{w}(\boldsymbol{\theta}^{\boldsymbol{r}^a}_{o_2} = i|\boldsymbol{l}^{\boldsymbol{\theta}^{r^a}}) \cdot \mathsf{f}(\boldsymbol{l}^{\boldsymbol{c}}_{o_1}, \boldsymbol{l}^{\boldsymbol{r}^a}_{o_2}|r^m, r_i^a, c_i, o_1, o_2),\end{aligned} \tag{6.6}$$

where o_1 and o_2 respectively correspond to the cycles/operations' indexes where c_i and r_i^a can be manipulated. The leakages about both permutations are denoted as $\boldsymbol{l}^{\boldsymbol{\theta}^{\boldsymbol{c}}}$ and $\boldsymbol{l}^{\boldsymbol{\theta}^{\boldsymbol{r}^a}}$. The leakage about c_i (resp. r_i^a) at this index is written as $\boldsymbol{l}^{\boldsymbol{c}}_{o_1}$ (resp. $\boldsymbol{l}^{\boldsymbol{r}^a}_{o_2}$) when manipulated at index o_1 (resp. o_2).

The function $\mathsf{w}(\cdot|\cdot)$ assigns a weight to every possible index locations o_1 and o_2. Multiple $\mathsf{w}(\cdot|\cdot)$ exist and the previous notation remaining generic. In [VMKS12] the authors illustrate that its choice depends on the shuffling scheme, permutation size and computational resources. In this chapter, we make use of the *direct permutation leakage* methodology for $\mathsf{w}(\cdot|\cdot)$ as defined in [VMKS12]. However in Section 8.3 (which is based on a more recent work), we propose an improved weight function ($\mathsf{w}_{\mathsf{New}}(\cdot|\cdot)$ from Equation 8.5) that could improve the following results.

6.3.2 Time complexity improvements

Even when considering the previous D&C adversary, the previous expression implies a need to estimate a large (and hardly realistic)

[2] The randomness is reused between the masking and shuffling

number of PDFs (aka templates). Indeed, a direct/exhaustive profiling requires to estimate a template for every combination of r^m, r_i^a and c_i as well as for the indexes when they are manipulated, leading to a total of $255 \cdot 2^8 \cdot 2^8 \cdot |o_1| \cdot |o_2|$. Hereafter, we show how the profiling and attack time complexities can be reduced by exploiting an independence assumption in Equation 6.5. For this purpose, we first assume that the leakage l is composed of three leakages with independent noise (which is natural in the context of software implementations). The previous equation then becomes:

$$\hat{\mathsf{f}}(\boldsymbol{l}|x_i) \propto \sum_{r^m, r^a} \hat{\mathsf{f}}(\boldsymbol{l}^{r^m}|r^m) \cdot \hat{\mathsf{f}}(\boldsymbol{l}^{r_i^a}|r_i^a) \cdot \hat{\mathsf{f}}(\boldsymbol{l}^{c_i}|c_i)\,, \tag{6.7}$$

where $\boldsymbol{l}^\star$ is the leakage about the share $\star$. This simplification allows one to target the permutation on each of the shuffled shares independently and not jointly as performed in Equation 6.6. This results in equations:

$$\hat{\mathsf{f}}(\boldsymbol{l}^{c_i}|c_i) = \sum_{o_1} \mathsf{w}(\theta_{o_1}^{\boldsymbol{c}} = i|\boldsymbol{l}^{\boldsymbol{\theta}^{\boldsymbol{c}}}) \cdot \hat{\mathsf{f}}(\boldsymbol{l}_{o_1}^{\boldsymbol{c}}|c_i, o_1), \tag{6.8}$$

and:

$$\hat{\mathsf{f}}(\boldsymbol{l}^{r_i^a}|r_i^a) = \sum_{o_2} \mathsf{w}(\theta_{o_2}^{\boldsymbol{r}^a} = i|\boldsymbol{l}^{\boldsymbol{\theta}^{\boldsymbol{r}^a}}) \cdot \hat{\mathsf{f}}(\boldsymbol{l}_{o_2}^{\boldsymbol{r}^a}|r_i^a, o_2), \tag{6.9}$$

Therefore, the profiling data complexity is greatly simplified (with direct impact on the time complexity). Namely, the number of templates to estimate is reduced to only $255 + 2^8 \cdot |o_1| + 2^8 \cdot |o_2|$. Additionally, in the direct/exhaustive profiling, a fresh profiling trace $\boldsymbol{l}$ can only be used to estimate a single template since it corresponds to a single combination of secret values (i.e., shares and permutations). Thanks to the independence assumption, that trace can be separated in independent $\boldsymbol{l}^\star$'s, each being used to update a template. Consequently, for a fix number of available profiling traces, each of the PDFs will be estimated with more samples, leading to more accurate templates (i.e., less estimation errors) with less data complexity.

The introduced independence assumption also has a (more limited) positive impact on the complexity of the online attack. Namely, the computation of Equation 6.7 is slightly improved compared to Equation 6.5. Indeed, the impact of the two permutations can be estimated separately, reducing the number of sums it induces from $|o_1| \cdot |o_2|$ down to $|o_1| + |o_2|$. In Subsection 6.3.3, we show how the attack complexity can be further (and significantly) improved by exploiting countermeasures' dissection.

As already mentioned, the proposed independence assumption is natural in the context of software implementations where the shares

are manipulated in different clock cycles. Yet, it requires that the noise variables of the different shares are uncorrelated to a sufficient extent, which may of course not be perfectly fulfilled. Our following experimental results will show that this assumption leads to excellent attack results (yet that a machine learning based attack escaping this assumption leads to mild improvements) even tough the attack is order-preserving the sense of Chapter 5.

6.3.3 Countermeasures' dissection

Multiplicative mask. Based on the previous abstractions, we can now easily describe the goal of countermeasures' dissection. Namely, and in order to reduce the data complexity of the attacks, the goal of such a dissection is to try biasing the weights of each sum in Equation 6.7, ideally to the point where there is a single possible event (i.e., mask, permutation) to consider. To do so, sub-attacks are performed against each single trace in order to (at least partially) recover ephemeral secrets such as the multiplicative mask r^m. For example, the previous equation can be rephrased by using Bayes's law to the r^m PDFs such that:

$$\hat{\mathsf{f}}(\boldsymbol{l}|x_i) \propto \left(\frac{1}{|r^m|} \cdot \sum_{r'^m} \hat{\mathsf{f}}(\boldsymbol{l}^{r'^m}|r'^m)\right) \cdot \sum_{r^m, r_i^a} \hat{\mathsf{p}}(r^m|\boldsymbol{l}^{r^m}) \cdot \hat{\mathsf{f}}(\boldsymbol{l}^{r_i^a}|r_i^a) \cdot \hat{\mathsf{f}}(\boldsymbol{l}^{c_i}|c_i) \tag{6.10}$$

$$\propto \alpha \cdot \sum_{r^m} \hat{\mathsf{p}}(r^m|\boldsymbol{l}^{r^m}) \cdot \sum_{r_i^a} \hat{\mathsf{f}}(\boldsymbol{l}^{r_i^a}|r_i^a) \cdot \hat{\mathsf{f}}(\boldsymbol{l}^{c_i}|c_i). \tag{6.11}$$

There, the factor α does not need to be computed since it is independent of x and will therefore be canceled in Equation 2.13 (i.e., when the attack is performed). Expressed in that form, it appears that the probability of the multiplicative mask r^m given the leakage sample is first computed. This is equivalent to a partial attack on that secret value of which the output is used to attack the complete scheme.

In the case this sub-attack is successful, $\hat{\mathsf{p}}(r^m|\boldsymbol{l}^{r^m})$ will be close to one for the correct r^m and close to zero for all the remaining possibilities. In that case, only the sum over r^a must be performed for the correct r^m. This reduces the number of sums by a factor 255 and completely cancels the impact of this mask on the data complexity. In the following, we will show that such a partial attack on r^m is successful with an accuracy of 100% (thanks to the heavily leaking pre-computation that manipulates this mask).

Permutations. A similar approach can be used to skip the sum over $|o_1|$ and $|o_2|$ in Equation 6.8 and Equation 6.9. The main observation in this case is that the ANSSI implementation contains two permutations: one over the 16 Sboxes ($\boldsymbol{\theta}$), one over the 4 `MixColumns` ($\boldsymbol{\theta'}$). Using the terminology of [VMKS12], we can anyway exploit a direct permutation leakage in both cases. As an example, we directly apply to the proposed improved $\mathsf{w}_{\mathsf{New}}(\cdot|\cdot)$ strategy from Equation 8.5. Eventually, the permutations $\boldsymbol{\theta'}$ is weaker since seeded with only two random bits. Therefore, one could simple use the following equation:

$$\hat{\mathsf{f}}(\boldsymbol{l}^{c_i}|c_i) = \sum_{o_1} \hat{\mathsf{p}}(\theta'^{o_1} = \mathsf{col}(i)|\boldsymbol{l}^{\boldsymbol{\theta'}}) \cdot \hat{\mathsf{f}}(\boldsymbol{l}|c_i, o_1), \tag{6.12}$$

where $\mathsf{col}(i)$ denotes the column in which the byte i is contained (according to the AES specifications). The probability $\hat{\mathsf{p}}(\theta^{o_1} = \mathsf{col}(i)|\boldsymbol{l}^{\boldsymbol{\theta'}})$ can be simply derived from 4 estimated conditional PDFs $\hat{\mathsf{f}}(\boldsymbol{l}^{\boldsymbol{\theta'}}|seed'_1)$ since $seed'_1$ is bijectively linked to a single permutation $\boldsymbol{\theta'}$ over the four columns. Taking advantage of these observations, we will show in the following that a sub-attack – similar to Equation 6.11 – against the seed of the permutation reaches a 98% accuracy, only leading to a very limited increase of the attack data complexity. As for the time complexity, the only remaining sum after the dissection presented here is over the 256 additive masks.

One may note that an extreme version of the countermeasures' dissection proposed here would be to directly express the ANSSI implementation as a factor graph and to take advantage of algebraic/analytical attacks [RSV09, VGS14, GS15, GRO18, BS21], where a similar independence assumption and a similar knowledge of the implementation details are also exploited, yet with more complex tools. We leave the investigation of such attacks as an interesting scope for further investigation.

6.3.4 Exemplary attack strategies

Before moving to concrete experiments, we finally specify a few representative (more or less powerful) attacks against the ANSSI implementation. Concretely, they are all based on the computation of Equation 6.7 exploiting the leakage of different operations.

Looking at Figure 6.1, it appears that an adversary at least has to target one operation involving $\boldsymbol{c}$ and one operation involving $\boldsymbol{r}^a$. Depending on this choice, the signal available from the leakage samples as well as the effectiveness of the shuffling countermeasure (e.g., if

only `MixColumns` is considered) may change, leading to different attack complexities.

Our four instantiated adversaries are:

- $\mathcal{A}_1$ obtains information about $\boldsymbol{c}$ by targeting the XOR of the state with r^{out}. This operation is not shuffled making the estimation of $\hat{\mathsf{f}}(\boldsymbol{l}^{c_i}|c_i)$ trivial (no shuffling). Information about $\boldsymbol{r}^a$ is extracted from the shuffled `MixColumns` with the complete sum of Equation 6.12. We note that this attack path could be easily avoided by shuffling the XOR operation. The number of sums to perform to compute Equation 6.7 is $2^8 \cdot 4$.

- $\mathcal{A}_2$ targets `MixColumns` for both shares but does not exploit the permutation leakage. Therefore, she has to perform the sum of Equation 6.8 and Equation 6.9 with $\mathsf{w}(\cdot|\cdot) = 1/4$. Hence, the number of sums to perform is equal to $2^8 \cdot (4 + 4)$.

- $\mathcal{A}_3$ is similar to $\mathcal{A}_2$ but does exploit the `MixColumns` permutation leakage. To do so, she explicitly computes the sum over all the possible indexes with Equation 6.12.

- Finally, $\mathcal{A}_4$ is equivalent to $\mathcal{A}_3$ but only considers one (i.e., the most likely) event in Equation 6.12 in order to limit the number of sums to 2^8.

All these attacks will be performed, compared and discussed next. First, in Subsection 6.4.3, the PDFs will be estimated under a Gaussian assumption (possibly exploiting PCA for dimensionality reduction).

Eventually in Appendix B, we show how a MLP can used to estimate the leakage on each of the shares individually, leading to slightly improved performances. In between, we discuss the limitations of black box security evaluations.

6.4 Experimental results

In this section, we show how practical attacks can be performed against the ANSSI library. First, the measurement setup is described. Then, the profiling phase is explained during which the PDFs of each secret value are estimated. Finally, the performances of all the adversaries are reported and commented.

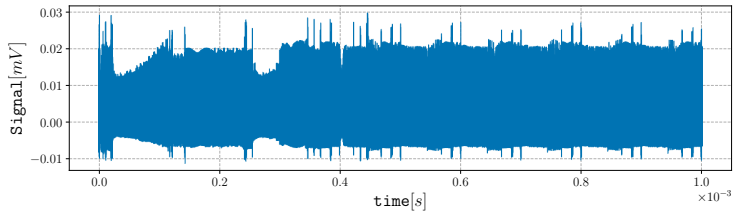


Figure 6.2: Mean trace obtained from the studied target.

6.4.1 Measurement setup

The targeted library is running on a `ATSAM4LC4CA` that is a `Cortex-M4` based micro-controller. The circuit under test is placed on a commercial board `SAM4L Xplained Pro` that has been slightly modified by removing decoupling capacitors. The target is running at its maximum clock frequency of 48[MHz].[3]

Its power consumption is measured through an electromagnetic near field probe `R&S HZ-15` placed directly above the package of the micro-controller. The pre-amplified signal with a matched `R&S HZ-16` is sampled at 1[GSamples/s] with a `PicoScope 5244d` and an 8-bit resolution. The resulting averaged power trace is shown in Figure 6.2 where only the pre-processing and the 6 first rounds of the AES are recorded. The traces are 10^6 samples long and are obtained at a rate of about 35 traces per second (because of the interface with the scope). For this chapter, $5 \cdot 10^5$ traces have been recorded with known random masks, seeds, keys and plaintexts for profiling purpose. In addition, $2 \cdot 10^5$ traces were also recorded with a fixed key in order to mount the attacks.

We stress that particular care was paid to the quality of the measurements. More precisely, a large configuration space has been explored including probe position, sampling frequency/resolution as well as decoupling network. The traces used in this work are recorded in the setup maximizing the SNR on the multiplicative mask r^m. This choice is made because of its importance in countermeasures' dissection. No engineering on frequency filtering was performed, which could possibly lead to improved attacks.

Note that the implementation and setup described in this section are not exactly the same as used in the preliminary leakage assessments

[3] It turned out that the library was not constant time because of cached memory access on this platform. The experiment are performed with a table free `xtime` making the encryption constant time.

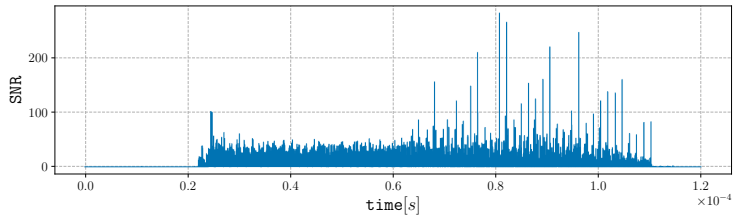


Figure 6.3: SNR on r^m during the multiplicative table pre-computation.

performed by the ANSSI team. The impact of these differences on our conclusions will be discussed in Section 6.4.6.

6.4.2 Templates' estimation

Our template estimation uses the tools from Section 2.4 and proceeds in two steps. First the POIs are found with SNR and then gT are built. The projection $\mathbf{W}$ is derived with a PCA [APSQ06]. The size of the linear subspace n'_s is set to 10 for 8-bit variables and to 3 for 2-bit variables (i.e., $seed'_1$ and $seed'_2$).

We next highlight how a PDF can be estimated for all the secret variables, starting with the multiplicative mask, followed by the permutations' seed and the additive shares. We also provide intuition about the origin of the leakage.

Multiplicative mask. In order to build a gT to obtain $\hat{\mathsf{f}}(\boldsymbol{l}^{r^m}|r^m)$, the pre-computation of the multiplicative table by r^m is targeted. Indeed, this is the only pre-computation that only involves r^m and no other secret value. The corresponding SNR is shown in Figure 6.3, and exceeds 200 on the peaks value. Such a high value is due to type of memory accesses targeted: words of 32 bits are stored in memory for each of the four bytes corresponding to four multiples of r^m. The stored values are 32-bit wide but only depend on 8 bits. This leads to a variance between the signal of the 255 classes that dominates the noise in our experimental setup. The resulting model allows to perfectly re-identify the multiplicative mask with a single trace (i.e., with an accuracy of 100% and on all the $2 \cdot 10^5$ attack traces).

Permutations. Next, we described the conditional PDF estimation $\hat{\mathsf{f}}(\boldsymbol{l}^{\boldsymbol{\theta}'}|seed'_1)$ used in Equation 6.12 where $seed'_1$ is a 2-bit value used to generate the permutation (again, a similar equation holds with o_2, l'_2 and $seed'_2$). The SNR of both permutations is shown in Figure 6.4. The first

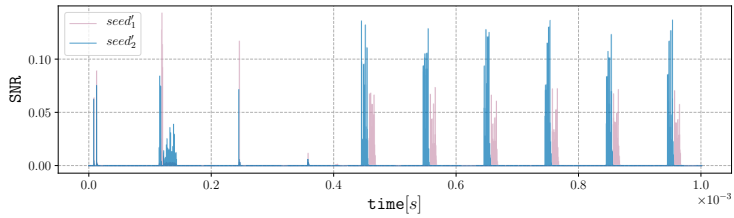
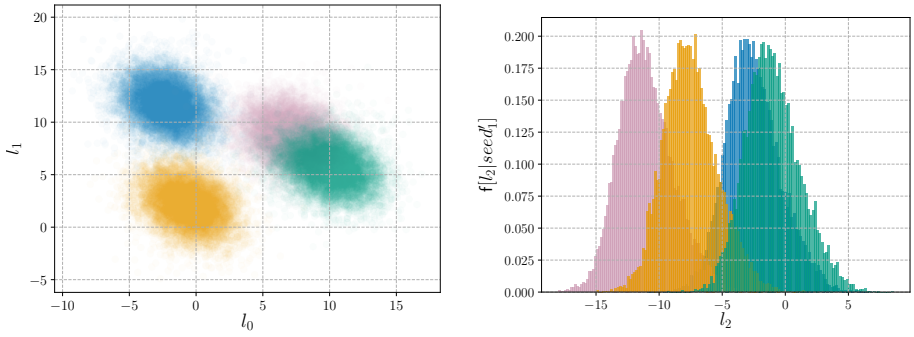


Figure 6.4: SNR **on** `MixColumns` **permutations' seeds.**



(a) Dimensions 1-2. **(b) Dimension** 3.

Figure 6.5: PCA **output on** $seed'_1$ **for the three studied dimensions.**

peak shows the loading of the random bits. The second ones correspond to the permutations' computation. Then, the pattern repeated six times is the actual `MixColumns`. It clearly appears that the two shares are processed serially starting with $\boldsymbol{r}^a$ followed by $\boldsymbol{c}$. Each of them are respectively shuffled with $seed'_1$ and $seed'_2$. In this operation, the leakage depends on the memory address accessed, which depends on the studied seeds.

Because of the large dimensionality of the leakage, we first selected $3,000$ dimensions showing evidence of leakage based on the SNR. These are reduced using a PCA to only 3 dimensions (leveraging the small sample size variant of PCA proposed in [APSQ06]). Figure 6.5 shows the projected samples for the four possible $seed'_1$ values. Already by looking to the two first dimensions (Figure 6.5a), most of the classes are well separated. Only the pink and the green clusters are overlapping. By taking the third dimension into consideration, these two are also well separated. Four gTs are then fitted based on these clusters, and allow retrieving the correct seed with an accuracy of 98%.

Intuitively, this accuracy can be explained by the fact that the same permutation is used at every round, increasing the number of leaking

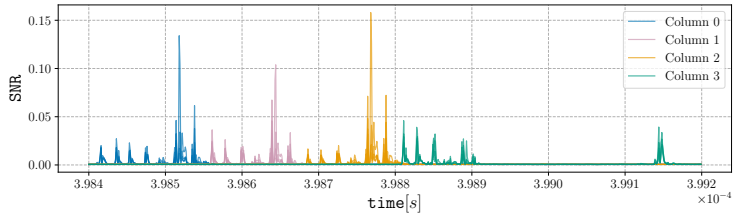


Figure 6.6: SNR **on** $\boldsymbol{c}$ **during** r^{out} **addition.**

dimensions about the seeds. These distant samples have potentially uncorrelated noise while manipulating the same data and therefore allow reducing the noise thanks to averaging. Such an averaging is implicitly performed by the PCA that is a weighted sum of samples. Additionally, the seed (and its bijectively derived data) are involved in different operations such as loads, permutation computations or memory accesses, each of them leaking in a different way, therefore providing additional independent information about the seed as well.

Additive shares. We finally highlight how information can be extracted on the additive shares given a permutation. For this purpose, one single template is computed for each index o_1, omitting the dependency to the actual manipulated byte i. In total, 16 gTs are estimated independently of the shuffling strategy. Because the leakage about a share is typically contained in a few cycles, only the SNR is used to isolate the POIs. In the following, we restrict our profiling to the operations exploited by the previously mentioned adversaries.

First, in Figure 6.6, the SNR on each byte of $\boldsymbol{c}$ is reported and color-grouped by column during the addition with r^{out}. This operation is implemented with a few sequential instructions. An entire column is loaded at once from memory into a register, then it is bitwise XORed with r^{out} (duplicated 4 times) and finally it is stored in memory. Therefore, all the four bytes of the same columns leak at the same time. To our understanding, the first peaks correspond to the XOR operation while the larger (and last) peak corresponds to the memory write. The resulting gTs are used only by $\mathcal{A}_1$.

Second, information about $\boldsymbol{c}$ can also be extracted from the `MixColumns` operation as exploited by $\mathcal{A}_2$,$\mathcal{A}_3$ and $\mathcal{A}_4$. Because this operation is shuffled, a profile for each c_{o_1} is built. The SNR is plotted in Figure 6.7 where the color-grouping corresponds to a constant o_1. Contrarily to the previous case, the bytes of the columns are serially loaded from memory and stored in registers. Then, these bytes are

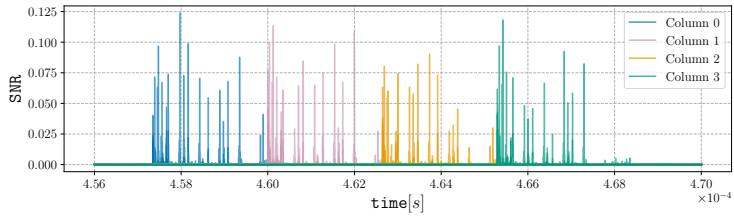


Figure 6.7: SNR **on** c **during** `MixColumns`**.**

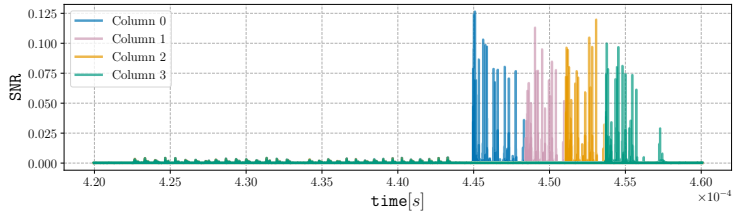


Figure 6.8: SNR **on** r_a **during** `MixColumns`**.**

combined one with each other within registers to perform the matrix multiplication of `MixColumns`. Therefore, the four bytes of the processed column do not always leak at the same point in time.

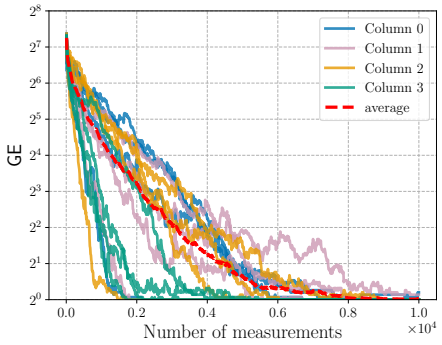
A similar behavior is observed for the template corresponding to $\boldsymbol{r}^a$ in Figure 6.8. Additionally, we see smaller peaks before `MixColumns`, generated by `ShiftRows`. Each of them corresponds to the shuffled loading of a byte or the store of one of them. These smaller SNR values would typically be the ones that would be exploited by an adversary targeting the (well) shuffled Sbox executions in a black box manner.

6.4.3 Adversary comparison

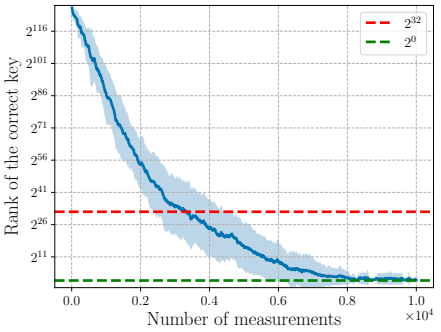
Based on the previous profiling efforts, we can now compare the different adversaries introduced in Subsection 6.3.4. For each of them, the Guessing Entropy (GE) for each subkey byte is first reported [SMY09]. Then, the key rank of the full key is also estimated thanks to the rank estimation algorithm of [PSG16].

The first adversary $\mathcal{A}_1$ exploits jointly the leakage from the addition of $\boldsymbol{c}$ with r^{out} and the leakage of `MixColumns` on $\boldsymbol{r}^a$. As shown on Figure 6.9, such an adversary requires about $10 \cdot 10^3$ measurements to recover the full encryption key. The data complexity is reduced to $3 \cdot 10^3$ if key enumeration is performed.

The second adversary $\mathcal{A}_2$ targets `MixColumns` for both shares but does not exploit information about the permutation used. The resulting

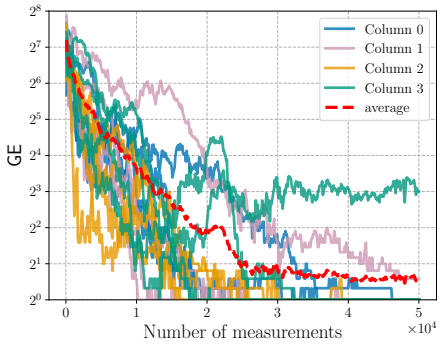


(a) GE of all the bytes.

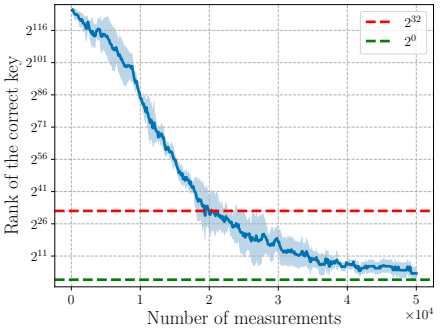


(b) Rank of the correct key.

Figure 6.9: Attack results for $\mathcal{A}_1$.



(a) GE of all the bytes.



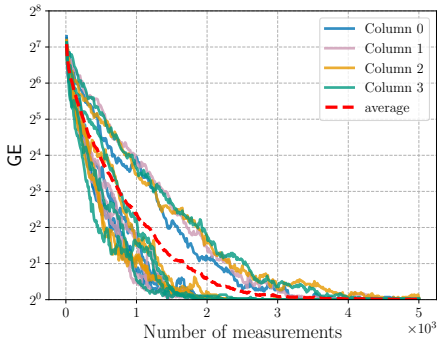
(b) Rank of the correct key.

Figure 6.10: Attack results for $\mathcal{A}_2$.

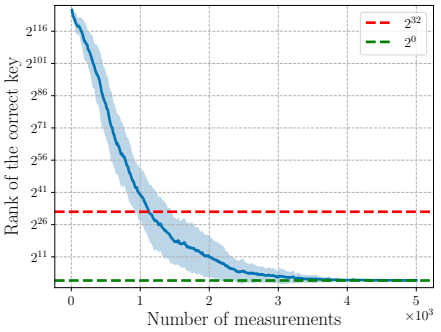
attack performances are given in Figure 6.10. With a total amount of $50 \cdot 10^4$, the full key is not recovered without post-processing. The latter is due to some "harder to recover" subkey bytes, as illustrated in Figure 6.10a. Yet, key enumeration still allows recovering the full key with $20 \cdot 10^3$ traces.

The third attack performed by $\mathcal{A}_3$ exploits the `MixColumns` leakage for both shares as well as the leakage on the permutations. The attack results are shown in Figure 6.11. With around $1.1 \cdot 10^3$ traces, the entropy of the full key is reduced to less than 32 bit, while it is directly recovered with $3 \cdot 10^3$. One can also notice that one byte per column is more difficult to recover, slowing down the full attack.

The last adversary, $\mathcal{A}_4$, is reported in Figure 6.12. It requires about $1.2 \cdot 10^3$ traces with key enumeration and $4 \cdot 10^3$ without. Note that the GE of some of the bytes is not to 1 (but close to it) with that amount of traces (so the outputted key is not necessarily ranked first but closed

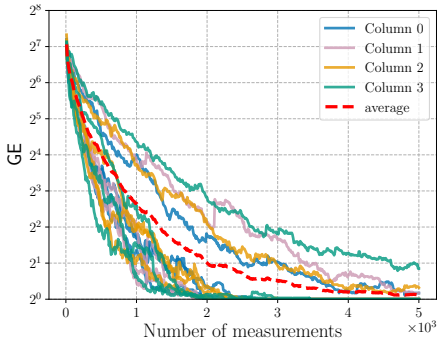


(a) GE of all the bytes.

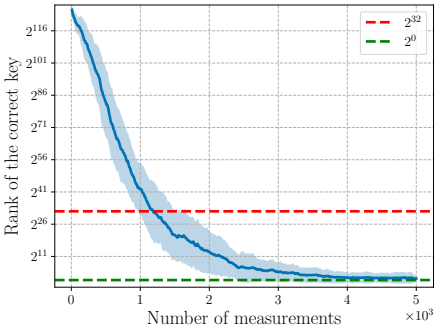


(b) Rank of the correct key.

Figure 6.11: Attack results for $\mathcal{A}_3$.



(a) GE of all the bytes.



(b) Rank of the correct key.

Figure 6.12: Attack results for $\mathcal{A}_4$.

to it).

6.4.4 Discussion

Our different results are summarized in Table 6.1, which allows evaluating the influence of the shuffling and targeted operations on the attacks' complexity.

First, by moving from $\mathcal{A}_1$ to $\mathcal{A}_3$, only the targeted operation to gain knowledge about $\boldsymbol{c}$ is changed. Hence, the reduced complexity is inherent to the targeted operation. Namely, the information exploited by $\mathcal{A}_1$ is coming from a few cycles with 24 random bits manipulated in parallel (Figure 6.6). The information exploited by $\mathcal{A}_3$ comes from multiple operations in `MixColumns`, leading to more extractable and independent signal.

The effect of shuffling can be measured by moving from $\mathcal{A}_2$ to $\mathcal{A}_3$.

Adversary	Direct key recovery	Key enumeration
$\mathcal{A}_1$	$1 \cdot 10^4$	$3 \cdot 10^3$
$\mathcal{A}_2$	$> 5 \cdot 10^4$	$2 \cdot 10^4$
$\mathcal{A}_3$	$3 \cdot 10^3$	$1.1 \cdot 10^3$
$\mathcal{A}_4$	$4 \cdot 10^3$	$1.2 \cdot 10^3$
$\mathcal{A}_5$	$2 \cdot 10^3$	$9 \cdot 10^2$

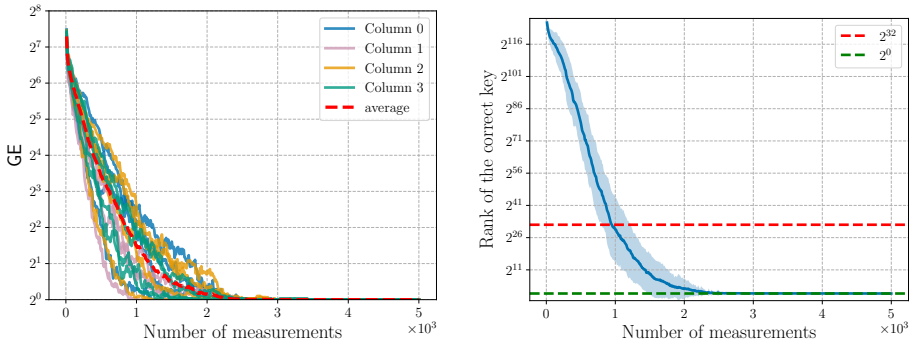
Table 6.1: Summary of all data complexity for all the investigated adversaries.

If no information is exploited about the permutation, the information per share is (roughly) divided by the number of possible indexes (i.e., $|o_1|$ and $|o_2|$). As a result, the noise on each share is increased by a factor 4, leading to a data complexity 16 larger (because of masking). This factor 16 fits the difference between both attacks (that respectively require $1.1 \cdot 10^3$ and $2 \cdot 10^4$ traces) reasonably well.

Eventually, omitting some terms in the sums (i.e., moving from $\mathcal{A}_3$ to $\mathcal{A}_4$) only marginally increases the data complexity since the secret permutation is recovered with 98% of accuracy in our experiments. So it mostly improves the attack time complexity by a factor 8. (In case of less accurate information extraction about the permutation, this change would be more of a tradeoff between an improved time or data complexity).

We recall that all the attacks described in this section are exploiting the (full) knowledge of the multiplicative mask r^m that is recovered in a single trace. With this knowledge, the affine masking is turned into a Boolean masking with two shares (given a multiplication by r^m in $\mathsf{GF}(2^8)$ is performed). Overall, the success of these attacks is mostly due to the limited amount of noise on the target device, which naturally implies that the "noise amplification" that masking provides only has a limited impact on the final complexities.

We insist that such a low noise level was already pointed out in the technical report of [BKPT19] where the analysis was made on a different target (`STM32` with a `Cortex-M4`). For example, given the noise level that we measured, extrapolations based on [DFS15] show that extending the same masking scheme as used in the ANSSI library would require 18 shares to prevent attacks with up to 2^{64} measurements. So our results mostly show that ensuring security in such low-cost devices is challenging, and a first step would probably be to try increasing the – algorithmic or physical – noise so that masking becomes more effective (which would also have a positive impact on shuffling [VMKS12]).



(a) GE of all the bytes. **(b) Rank of the correct key.**

Figure 6.13: Attack results for $\mathcal{A}_5$.

6.4.5 Dissection with MLP

As discussed in Section 6.3, performing an efficient countermeasures' dissection based on gT requires a few (independence) assumptions. Despite the experiments in Subsection 6.4.3 show that these assumptions hold to a sufficient extent so that powerful attacks can be mounted based on them, it remains that avoiding such hypotheses may lead to improved attacks (e.g., a better modeling of the leakages).

In the following, we run a MLP on each of the shares individually, so that the MLP replaces the computation of Equation 6.8 and Equation 6.9 given the same leakage samples. As a result, it estimates independently the probability of each share while taking into account the leakage of the permutation on `MixColumns`. This adversary is denoted as $\mathcal{A}_5$.

The resulting attack performances are reported in Figure 6.13 where only $2{\cdot}10^3$ traces are needed to recover the full 128-bit key. Furthermore, the correct key is ranked in the 2^{32} most probable ones with only 900 measurements, which improves over $\mathcal{A}_3$. This gain presumably comes from the absence of leakage assumption for the MLP. In particular, it does not assumes independence between the leakage of the permutation and the data, nor a Gaussian distribution. Interestingly, we note from Figure 6.13a that all the bytes are now roughly equivalent in front of $\mathcal{A}_5$, while it was not the case for $\mathcal{A}_3$ (Figure 6.11a), so we assume that some bytes were hard to recover due to assumption errors by $\mathcal{A}_3$.

This last attack confirms that machine learning algorithms can be used in complement with other approaches to evaluate protected implementations in a close to worst-case manner. It also shows that claims of better automation do materialize in advanced scenarios, since the manipulation of the permutation and shares leakages in this section

is simpler than in Section 6.4, which is in line with previous results (e.g., [CDP17]). Yet, these positive results have to be moderated by the inherent limitations of black box security evaluations that experiments such as in Section B.2 illustrate.

6.4.6 Comparison with the ANSSI measurements

In view of the previous results, a natural question is whether they are due to implementation and setup differences with the ones used in the ANSSI leakage assessments. After we communicated our results to them, the ANSSI team kindly accepted to share their measurements so that we could answer this question. It turns out their (`STM32`) implementation has some minor difference with the one we analyzed (in particular the non-constant time behavior mentioned in Footnote 3). But for the rest, it gives rise to the same POIs as we exploit in our attacks. We therefore repeated the same attacks as in this section using the traces from [BKPT19] and could recover the full key with approximately 5,000 traces and an enumeration power of 2^{32}. The only noticeable difference is a slightly lower SNR, especially for the same four bytes as we observed (so enumeration is actually more critical in the case of the measurements used by the ANSSI team). We assume this difference is mostly due to an improved measurement setup.

6.5 How not to interpret our results

Considering the low-complexity of the attacks put forward in this paper, a tempting conclusion could be that the open approach chosen for the ANSSI implementation is too risky. In mine opinion, the correct conclusion is the exact opposite. Namely, the ANSSI implementation puts forwards the (significant) risks of overstated security that closed source designs evaluated in a black box manner carry. This is similar to the conclusions of Chapter 5.

Concretely, the fine grain analysis we provide was only possible (under the time constraints of evaluators and researchers) because of the knowledge of the target implementation. Hence, as our understanding of side-channel countermeasures progresses, preventing such a fine grain analysis can still postpone the finding of some attacks (by the time needed to reverse engineer a single implementation), but it is more and more at the cost of the long-term security guarantees that an open approach leveraging sound physical assumptions and mathematical amplification can offer (see [Sta19] for a discussion).

As far as designs are concerned, we believe it is an important long-term goal to evolve towards physical security based on well-understood solutions surviving public audits. The collaboration of public agencies is of great help for this purpose, in order to gradually increase the security level of public implementations and identify relevant targets.

Evaluation of 32-bit masked bitslice software

7

This chapter is based on the work *"Breaking Masked Implementations with Many Shares on 32-bit Software Platforms or When the Security Order Does Not Matter"* that has been published in TCHES 2021 [BS21].[a]

Next, I present a methodology to evaluate bitslice masked implementations. The initial motivation was to evaluate the software implementation I developed (with other PhD students from UCLouvain) for the CHES2020 CTF. While the literature corresponding to the design of such software is deep, not much was done about dedicated practical evaluations pushing for new tools.

Interestingly, while I did not release our security analysis, the BSI came up with a similar worst-case approach leveraging deep learning instead of Gaussian templates and without SASCA. Their approach was similar to the one presented in Subsection 6.4.5.

[a] Co-author: François-Xavier Standaert.

7.1 Contextualization

Motivation and goals. In this chapter, we perform a (close to) worst-case attack to estimate the security that comes with masked bitslice software implementations. 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 (see Chapter 6). 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] or the onces produced by Tornado [BDM+20] have never been evaluated against advanced (e.g., horizontal) side-channel attacks.

Next, 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 as proposed in Equation 2.12), and analytical strategies enabling the exploitation of all the shares' manipulations, for multiple intermediate computations such as SASCA (see Subsection 2.5.2). 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:

1. We show that despite the fact that bitslice masking provides better opportunity for secure implementation than the table-based solutions considered in [BKPT19, 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.

2. 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.

3. 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.[1]

4. 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.

5. In the spirit of proof-based evaluations, 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.

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 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.

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

7.2 Targets' description

We now detail the higher-order ($d > 2$) bitslice masked implementations that we analyse in this chapter. 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].

7.2.1 Micro-controllers and measurement setup

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. Therefore, we selected two targets with 32-bit architecture from ARM. Namely, we study a Cortex-M0 and a Cortex-M3.[2]

Regarding the measurement setup of the chapter, 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 as well as 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

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

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.

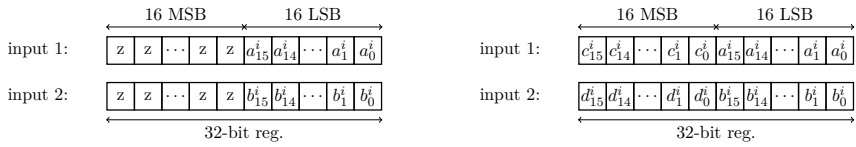
7.2.2 Goudarzi and Rivain's bitslice masked implementations

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 as detailed in Subsection 2.3.1. To do so, each input bit of the corresponding circuit is stored in different native words of the MCU, that we next call slices.

AES. 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 C.1. All the 16 Sboxes of the AES can be evaluated in parallel based on that circuit by using 8 slices of size 16. 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.

In this representation, the native slices of size 32 are not fully used and therefore Goudarzi and Rivain proposed dedicated optimizations to the masked AES. We detail in Figure 7.1 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 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 7.1a where the encodings are placed on the 16 LSBs of the slices. The 16 MSBs are not specified (z) 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



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

Figure 7.1: 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.

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.

Clyde. As for the Clyde tweakable block cipher, its non-linear layer consists in applying 32 independent 4-bit Sboxes [BBC+20a]. The representation of the Clyde Sbox given in Appendix C.2, Table C.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].

Implementations details. We note that we consider the randomness generally out of scope in this thesis. 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.

7.3 (Close to) worst-case evaluation methodology

In this section, we present our generic 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 of the attack.

7.3.1 Factor graph for masked Boolean circuits

Limitations of the full graph. When performing a SASCA, the first step is to define the factor graph (see Subsection 2.5.2). A direct and intuitive solution is to include all the intermediate values processed by the target in the factor graph as done in [GS18]. However for a masked implementation (e.g. using ISW multiplications), this solution rapidly becomes out of reach for large number of shares. 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 (Section 2.5). Additionally, such a graph includes many loops that make the BP heuristic.

New construction. 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 7.3.3 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 7.2 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 (e.g., an ISW gadget is

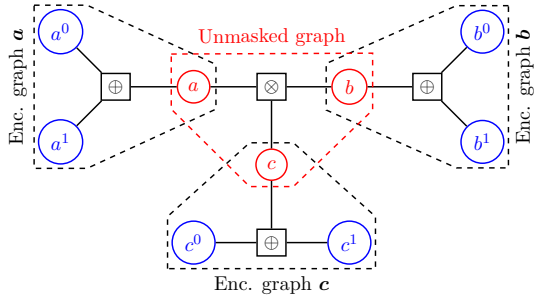


Figure 7.2: Factor Graph around AND gate for two shares implementation.

replaced by a $\otimes$). The probabilities on these unmasked inputs/outputs are taken from their respective encoding graphs.

Advantages. 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.

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.

7.3.2 Factor graph for (tweakable) block ciphers

The factor graph for the masked Sboxes is built similarly to Figure 7.2. To collect information about a single trace, the unmasked graph contains all the function and variable nodes described in Appendix C.1 for the AES and in Appendix C.2 for Clyde. 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].

AES specificities. For the AES, we link multiple trace graphs through the input linear layer as depicted in Figure 7.3. 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.

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.

Clyde specificities. 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 have full implementation

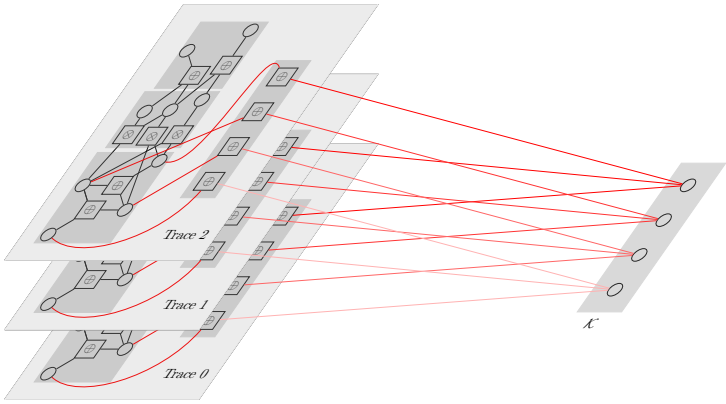


Figure 7.3: Unprotected graph for a AES SBox like circuit.

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.

7.3.3 Profiling and attack complexities

We now discuss the time and memory complexities of our analysis for each of the steps. Namely we describe the cost of the profiling and attack phase illustrated respectively in Figure 1.2 and Figure 1.3. 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. These steps are technically described in Section 2.4. Here, the focus is on their complexities in the context of masked bitslice software.

(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 $n_s = 3,000$ time samples that are (the most) significant ones, in order to limit the complexity of the final profiling step.

(c) The LDA + gT profiling are taken for PDF estimation as given in Section 2.4 where only the POIs of step 3 are modeled. 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. For each of the templates, the parameters μ_x, $\mathbf{\Sigma}$ and $\mathbf{W}$ have to be estimated. In this chapter, $\mathbf{W}$ is derived with LDA.

Overall, the bottleneck of the profiling is the SNR computation with its cubic cost. One could reduce this complexity by exploiting the prior 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 N 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 $\mathcal{O}(N \cdot d)$.

(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 $\mathcal{O}(N \cdot d \cdot \log d)$. 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 N trace graphs and the nodes in $\mathcal{K}$ which requires a memory growing in $\mathcal{O}(N)$ as well as its corresponding time complexity. We note that for large N, 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. For Clyde, 4 independent graphs are exploited since it is bitsliced on 32 bits. Eventually, the above methodology is fully implemented with SCALib and the code is available at `https://github.com/obronchain/BS21_ches2020CTF`.

Comparison with full graph. 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 $\mathcal{O}(N \cdot d^2)$. 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 $\mathcal{O}(N \cdot d^2)$, 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 time complexity is only in $\mathcal{O}(N \cdot d \cdot \log d)$, with a smaller constant (i.e. $8 \cdot 2^8$).

7.4 Higher-order attacks in worst-case settings

In this section, we perform attacks against the software implementations of the AES and Clyde presented in Section 7.2, 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.

7.4.1 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 +gT" method, she has to carefully choose the number of dimensions to keep after doing the LDA projection (i.e. n'_s). This choice depends on her available profiling data. If n'_s is too small, she may miss information hidden in other dimensions. If n'_s is too large, the estimation converges more slowly and the collected data may not be sufficient. In order to select this n'_s parameter, we study the PI of the resulting PDF estimations as defined in Subsection 2.2.2.

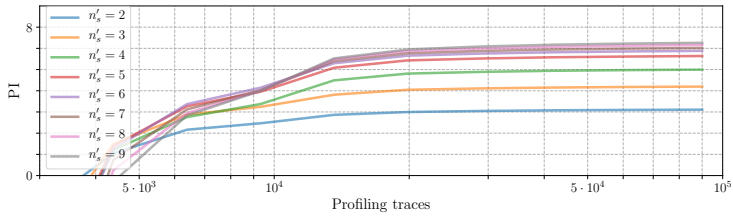
Figure 7.4 exhibits the PI (see Subsection 2.2.2) extracted for an exemplary variable node, according to the number of profiling traces used to build the model, and for various n'_s values. For both targets, the PI curves increase with n'_s until a saturation effect appears, meaning that a larger n'_s 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 $n'_s = 6$ for Cortex-M3 and $n'_s = 11$ 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.[3]

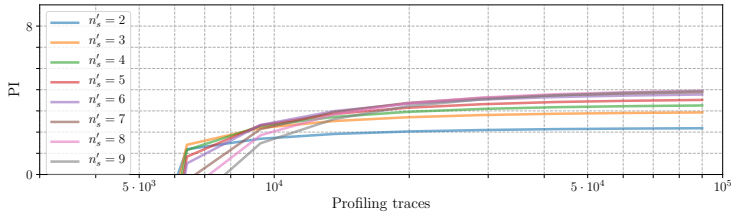
7.4.2 Attack data complexity

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

[3] 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.

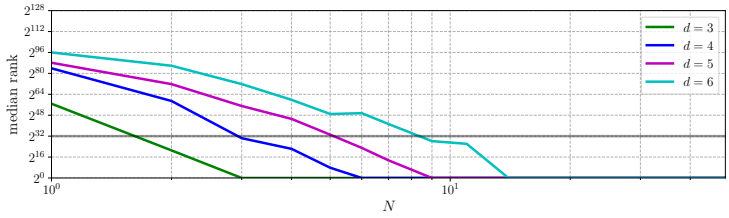


(b) Cortex-M3.

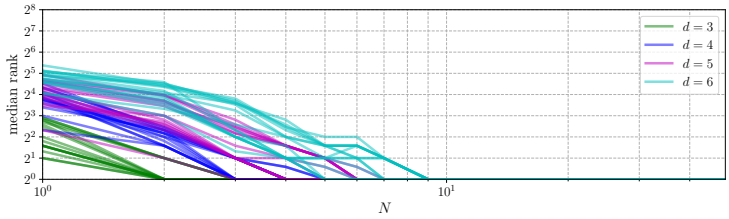
Figure 7.4: PI convergence for various dimensions after projection n'_s.

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.

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. As we will do in Chapter 6, 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.



(a) Full 128-bit key.



(b) Individual key bytes.

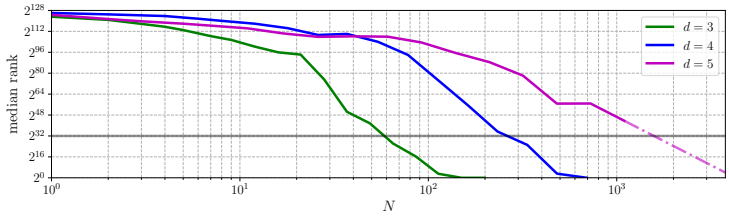
Figure 7.5: Attacks against the Cortex-M0, AES

Concrete attacks. The results of concrete attacks against the Cortex-M0 are given in Figure 7.5. As shown in Figure 7.5a, the attacks succeed in less than 10 traces with a probability large than 1/2. 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 7.6, 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.[4] 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 7.7.

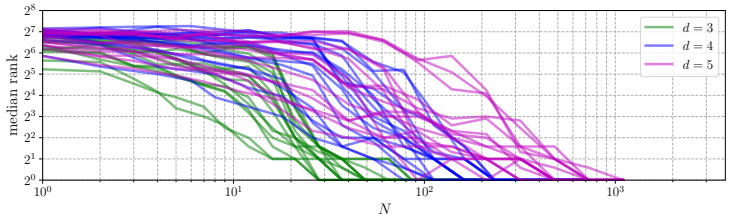
In both cases, we also report the data complexity to recover each byte. We observe that some bytes are more difficult to recover than others, which is in contrast with divide-and-conquer attacks against (e.g. unprotected) AES implementations with table lookups. We assume these differences are due to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

Extrapolated attack data complexity. The previous results allow us to extrapolate the attack data complexities to a larger number of

[4] 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 7.6: Attacks against the Cortex-M3, AES

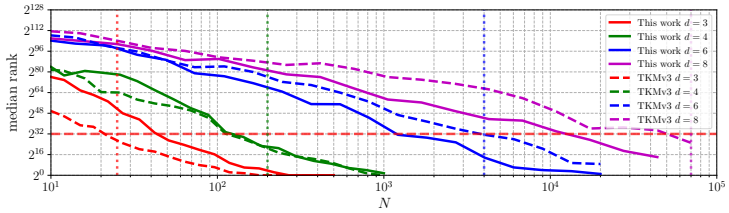


Figure 7.7: Attacks against the Cortex-M0, Clyde. The vertical dashed lines are the numbers reported for the best attack reported in the CHES 2020 CTF [BBC+20b].

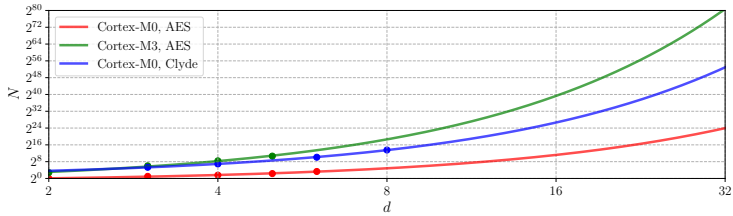


Figure 7.8: Data complexity of our attacks in function of the number of shares assuming a 2^{32} key enumeration. Markers are for real attacks. Lines are for extrapolations.

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 that for the considered adversary, the low noise is amplified by the number of shares d as expected from Equation 2.7. 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 2.7 resulting in the curves reported in Figure 7.8.[5] 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.

7.4.3 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.

7.4.3.1 Impact of the number of leaking dimensions

Next, we look at the impact of the number of POIs (n_s) obtained in Step 3 of Subsection 7.3.3 on the extracted information. On Figure 7.9, a bullet corresponds to a single share within the circuit. Its position

[5] These are estimated by minimizing the mean squared error between the extrapolation and the experimentally observed data complexities in Figures 7.5a and 7.6a and 7.7.

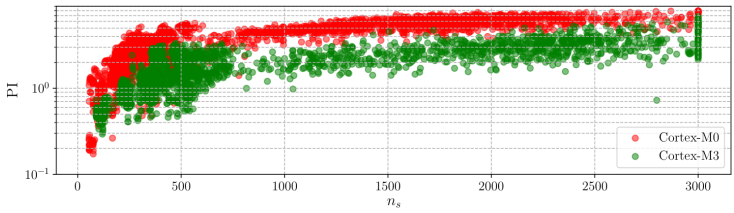


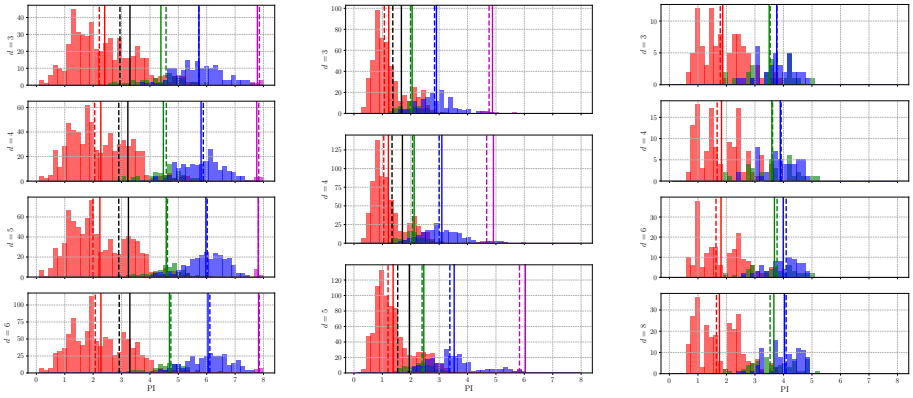
Figure 7.9: Relation between the number of profiled dimensions and the extracted information per share. Each point is for a single share in the AES implementation.

depends on n_s (x-axis) and the PI obtained on that share by exploiting these n_s time samples (y-axis) with Equation 2.12 . Overall, we observe that a larger n_s, 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.

7.4.3.2 Impact of the circuit structure

Since the previous experiments show that the noise level of the Cortex-M 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) 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.

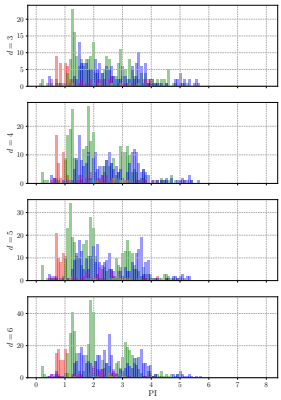
# of multiplications. We start by studying the impact of the number of multiplications m in Figure 7.10, which contains histograms of the PI measured for all the profiled shares within the encoding graphs. The color codes correspond to the value m of the share. For example, red histograms contain shares 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



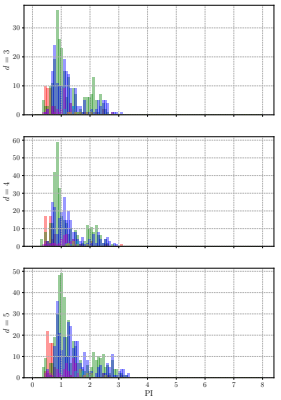
(a) Cortex-M0, AES (b) Cortex-M3, AES (c) Cortex-M0, Clyde

Figure 7.10: 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.

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 7.1a). 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.



(a) Cortex-M0, AES.



(b) Cortex-M3, AES.

Figure 7.11: 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 $a = 3$.

# of additions. 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 7.1a. 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 7.11, 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.[6]

# of shares. 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, 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

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

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.

7.5 Inter-device profiling portability

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 study the increase in attack data complexity induced by profiling on one device and attacking another one. 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:

$$\gamma_{i,j} = \mathsf{E}\left[\frac{\min(0, \mathsf{PI}_{j,i})}{\mathsf{PI}_{j,j}}\right], \tag{7.1}$$

where $\mathsf{PI}_{j,i}$ is the PI available when profiling a device j and attacking a device i.[7] The denominator corresponds to the best 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 2.7, we extrapolate that the data complexity is also growing exponentially such that:

$$N_{j,i} \approx N_{j,j} \cdot \left(\frac{1}{\gamma_{j,i}}\right)^d, \tag{7.2}$$

[7] 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.

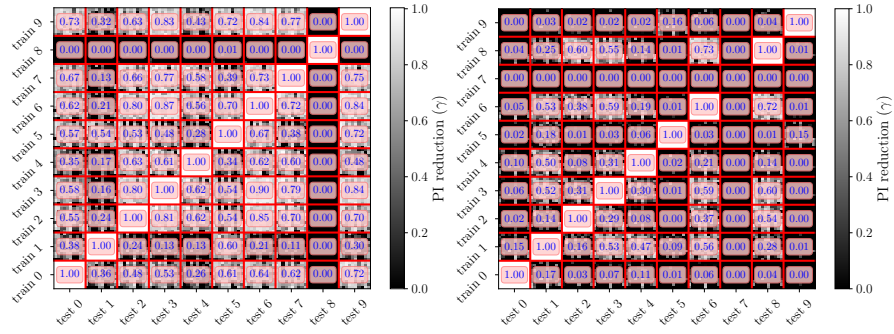
where $N_{j,j}$ is the data complexity in the worst-case setting estimated in Subsection 7.4.2. The term $N_{j,i}$ is the data complexity of the relaxed adversary.

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 7.2.1 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 7.12.[8] For the Cortex-M0 on Figure 7.12a, 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 7.12b, 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.

Overall, the results from Figure 7.12 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

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



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

Figure 7.12: Estimated PI loss factor ($\gamma_{j,i}$) for various training and testing devices. Estimation is done on 100 randomly selected intermediate variables.

.

device based on a pool of profiling devices is an interesting question that we leave for further research.

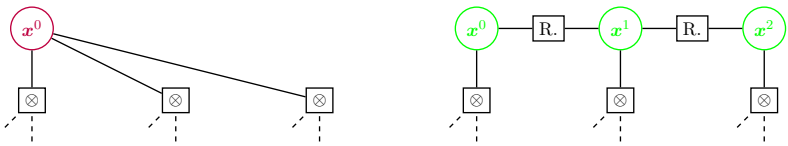
7.6 Extrapolated countermeasures impact

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.

7.6.1 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 7.10. 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.[9]

[9] Since the circuit is already probing secure without the refresh gadgets, these additional refresh gadgets do not require strong composability properties like strong



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

Figure 7.13: refreshing strategy to reduce the m parameter.

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 7.10 to the green one ($m = 1$). The refreshing strategy we used in our analyzes is described in Figure 7.13 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 7.10). 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 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 7.14. 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

non-interference.

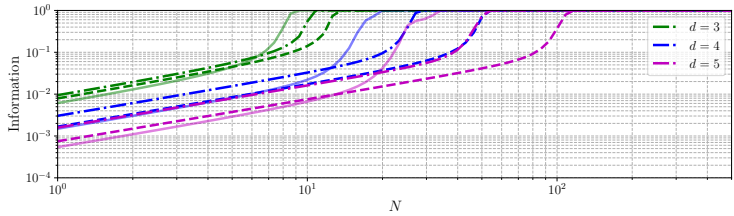


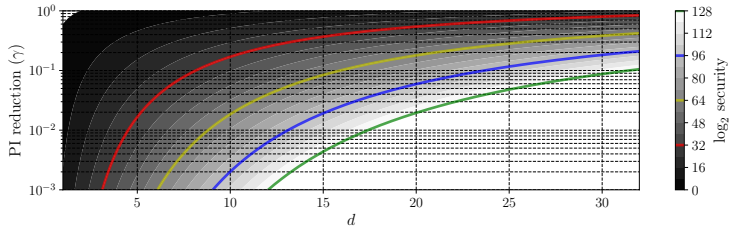
Figure 7.14: LRPM analysis of the Cortex-M3. − for actual PI at each node, −· for mean PI according to m and −− for mean PI with additional refresh gadgets inserted.

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 to be more significant (e.g. lightweight ciphers like Clyde).

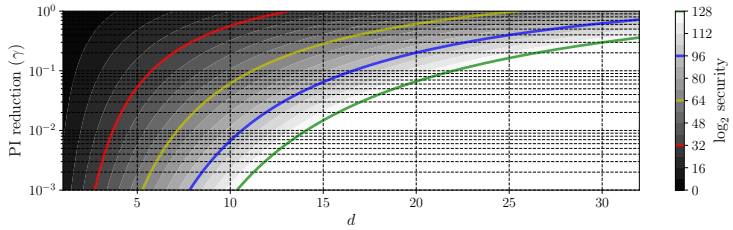
7.6.2 Information reduction countermeasures

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-alone countermeasure, while it could potentially be combined with other countermeasures.

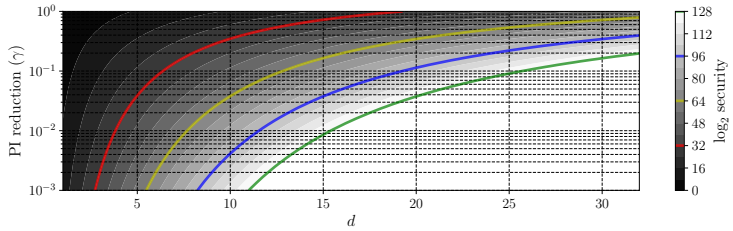
Yet, and as a first step in the directions of designing software implementations with higher security at lower cost, we next extrapolate the impact of combining masking with a (generic) countermeasure reducing the information available on each share. Namely, for a given information reduction factor γ, we report the estimated attack data complexity based on Equation 7.2. 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



(a) Cortex-M0, AES



(b) Cortex-M3, AES



(c) Cortex-M0, Clyde

Figure 7.15: Extrapolated attack data complexity with 2^{32} enumeration power according to the information reduction on a single share compared to our adversary.

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]. The case of shuffling is further studied in Chapter 8 with a practical evaluation of γ in Section 8.6. This shows that the impact of shuffling remains small on such Cortex-M targets. It finally covers the case of a less efficient measurement setup used by the adversary.

For example, Figure 7.15 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.

Part III

Countermeasure design

How to combine masking and shuffling ? 8

This chapter is based on the work *"Bitslice Masking and Improved Shuffling: How and When to Mix Them in Software?"* that is under submission [ABG+21].[a]

During the publication process of the results presented in Chapter 7, the reviewers were regularly arguing that to increase the security of masked software, designers should rely on a combination of masking with hiding/shuffling. This was the initial motivation for this project especially since the state of the art does not allow to mix masking and shuffling with an exponential effect.

Therefore, the last part of this thesis is dedicated to the design of mixed countermeasures. I next show that such combination of masking and shuffling is feasible but is not a "magic trick". Indeed, these solutions require sufficient noise, which is not always available.

[a] Co-authors: Melissa Azouaoui, Olivier Bronchain, Vincent Grosso, Kostas Papagiannopoulos, François-Xavier Standaert.

8.1 Contextualization

Ever since the introduction of Differential Power Analysis (DPA) by Kocher et al. [KJJ99], the idea that side-channel countermeasures must be combined to be effective has become a mantra. For example one general conclusion of the DPA book is that "*implementing a combination of several cheap countermeasures typically leads to a much better protection than one expensive countermeasure*" [MOP07]. By "much better protection", one implicitly means that the complexity of an attack against a combination of countermeasures should be significantly higher than the sum of the complexities to attack each countermeasure separately.

Following this intuition, any pair of countermeasures could

potentially be combined, raising the question whether they indeed lead to concrete benefits in terms of security vs. performance tradeoff. In this respect, it was also put forward that the most promising countermeasures to combine are the ones that provide complementary improvements. One popular example of complementary countermeasures is hiding and masking [MOP07]. Hiding aims at reducing the attack's Signal-to-Noise Ratio (SNR) [Man04], by increasing the measurement noise or reducing the side-channel signal, as for example happening when shuffling an implementation [HOM06]. Masking aims at reducing the data-dependencies of the leakages by randomizing the intermediate computations of a cryptographic implementation thanks to secret sharing, so that side-channel analysis becomes hard [CJRR99]. The statistical confirmation that this combination is sound was then put forward in an important paper by Rivain et al. [RPD09]. Assuming a shuffled execution of η independent operations (e.g., Sbox), each of them masked with d shares, they showed that for a sufficient noise variance σ^2 in the leakages, the complexity to attack the corresponding implementation grows in $\mathcal{O}(\eta \cdot (\sigma^2)^d)$.

Contributions. Based on this state-of-the-art, the main contribution of this chapter is to improve the security, the applicability and the evaluation of this important combination of countermeasures, by following three complementary tracks:

1. In terms of security under a sufficient noise regime, the proposal of Rivain et al. increases the complexity of attacks against masked implementations by a factor η.[1] Given that shuffling can be viewed as noise emulation and masking as noise amplification, a natural question is whether one could amplify the noise emulated by the shuffled operations thanks to masking and improve this gain to a factor η^d? We answer the question positively and describe an efficient solution for this purpose, with a systematic investigation of the options to combine masking and shuffling. At high level, this improvement is obtained by shuffling noisy shares rather than noisy tuples of shares.

2. In terms of applicability, several recent advances in the analysis and design of secure and efficient software masking exploit the bitslicing concept [GR17, BGR18, BDM+20] (see Subsection 2.3.1). The latter has not been studied by Rivain et

[1] Precisely, they have this factor for non-linear operations while having better results for linear operations – see the discussion in Section 8.4 for the details.

al. and raises new design questions. Namely, say you want to securely implement 128 independent AND gates in a 32-bit (ARM Cortex) device. What is the best combination of bistlice masking and shuffling given that both of them "utilize" the independent operations differently? We show that the good strategy is to mask first and shuffle next. In other words, fill the 32-bit bus with shares and shuffle what remains (i.e., four 32-bit operations in this example).

3. Eventually, the security evaluations of Rivain et al. rely on simple (correlation-based) side-channel attacks [PRB09]. This led them to the conclusion that shuffling remains useful even in low-noise regimes. Yet, the results in Chapter 6 showed that such evaluations can significantly overestimate the worst-case security level of an implementation. We therefore conclude the chapter with an experimental evaluation of the "masking + shuffling" combination in an ARM Cortex-M3 and we show that profiled multivariate attacks can cancel the impact of the shuffling in this case (while masking still offers a mild security improvement).[2] The latter can be explained by the quite leaky memory accesses that a shuffled implementation uses intensively, and raises the question of how to limit their leakage as an interesting scope for further research.

We combine these results with a technical/consolidating contribution. Namely, we describe an improvement of the multivariate attacks against shuffled implementations proposed in [VMKS12] that we use in our practical security evaluations.

8.2 Background

The AC12 attack. At Asiacrypt2012, Veyrat-Charvillon et al. proposed an approach to recover $\boldsymbol{y}$ by exploiting the leakage on the permutation indexes efficiently [VMKS12], as reflected by the following equation:

$$\mathsf{AC12}(\boldsymbol{l}|y_s) = \sum_{c=0}^{\pi-1} \underbrace{\mathsf{w}_{\mathsf{AC12}}(\theta_c = s|\boldsymbol{l}^{\boldsymbol{\theta}})}_{\text{perm. leak.}} \cdot \underbrace{\mathsf{f}(l^{y_{\theta_c}}|y_s)}_{\text{data leak.}}, \tag{8.1}$$

where $\mathsf{w}_{\mathsf{AC12}}(\theta_c = s|\boldsymbol{l}^{\boldsymbol{\theta}})$ is the weight assigned to a cycle c. Its goal is to indicate the probability that the targeted value y_s is manipulated at the

[2] Similar observations have been made for shuffled table recomputations [BGNT18].

cycle c. They propose many solutions to derive $\mathsf{w}_{\mathsf{AC12}}$ with different time and data complexities.[3] We focus on the so-called *"direct permutation leakages"* (DPLeak), for which this weight is computed as:

$$\mathsf{w}_{\mathsf{AC12}}(\theta_c = s|\boldsymbol{l}^{\boldsymbol{\theta}}) = \frac{\mathsf{f}(l^{\theta_c}|\theta_c = s)}{\sum_{c'=0}^{\pi-1} \mathsf{f}(l^{\theta_{c'}}|\theta_{c'} = s)} \cdot \tag{8.2}$$

Putting things together, the AC12 attack recovers the full vector $\boldsymbol{y}$ by applying Bayes on each of the elements of the vector independently, using a model:

$$\tilde{\mathsf{m}}_{\mathsf{AC12}}(\boldsymbol{y}|\boldsymbol{l}) = \prod_{s=0}^{\pi-1} \frac{\mathsf{AC12}(\boldsymbol{l}|y_s)}{\sum_{y_s^*} \mathsf{AC12}(\boldsymbol{l}|y_s^*)} \cdot \tag{8.3}$$

Note that since this model is imperfect, using it leads the adversary to exploit some perceived information rather than the whole mutual information. Note also that despite this attack is not summing over all the permutations (which makes it suboptimal from the data complexity viewpoint), it is not a divide-and-conquer one since the leakages of the permutation are still exploited jointly.

8.3 Improving the AC12 attack strategy

We next propose an improved computationally efficient strategy in order to attack shuffled implementations, and use simulated information theoretic evaluations to demonstrate the gains it provides over the AC12 approach.

8.3.1 Attack specification

In principle, the attack we propose is similar to the AC12 DPLeak one. Yet, it uses a slightly different model that is expressed by the following equation:

$$\tilde{\mathsf{m}}_{\mathsf{New}}(\boldsymbol{y}|\boldsymbol{l}) = \prod_{s=0}^{\pi-1} \sum_{c=0}^{\pi-1} \underbrace{\mathsf{w}_{\mathsf{New}}(\theta_c = s|l^{\theta_c})}_{\text{perm. leak.}} \cdot \underbrace{\frac{\mathsf{f}(l^{y_{\theta_c}}|y_s)}{\sum_{y_s^*} \mathsf{f}(l^{y_{\theta_c}}|y_s^*)}}_{\text{data leak.}}, \tag{8.4}$$

where the first term processes the permutation leakages and the second term processes the data leakages. Our modifications compared to Equation 8.3 are twofold. First, the weighted sum over c is not performed

[3] Which, for large enough noise, are equivalent from the data complexities viewpoint.

on continuous PDFs anymore but on the probabilities obtained after the application of Bayes (as reflected by the right term in the above equation). This is the usual approach taken for advanced attacks such as [VGS14]. Second, we notice that the weights in the sum estimated with Equation 8.2 depend on the full permutation leakage. Since each term in the sum corresponds to the leakage at cycle c, we modified the weights such that they give the probability that the index manipulated at cycle c is equal to s. Formally, this leads to:

$$\mathsf{w}_{\mathsf{New}}(\theta_c = s | l^{\theta_c}) = \frac{\mathsf{f}(l^{\theta_c} | \theta_c = s)}{\sum_{s'=0}^{\pi-1} \mathsf{f}(l^{\theta_c} | \theta_c = s')} \cdot \tag{8.5}$$

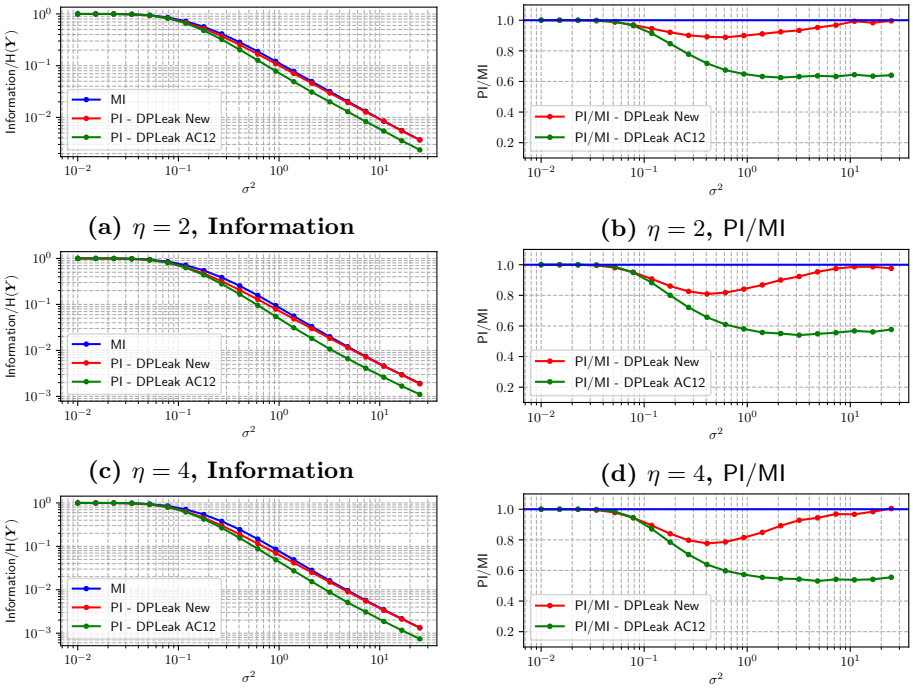
8.3.2 Models comparison

Methodology. Next, we simulate a shuffled implementation (i.e., Algorithm 3) where the leakages are distributed according to Equation 2.10. From these leakages and the knowledge of the true PDF, we first estimate the MI according to Equation 2.5 which represents the best attack possible against the implementation. We do that for small permutation sizes (since for large π values the direct computation of the MI is computationally hard). For the same implementations, we extract the PI thanks to Equation 2.6 for both models $\tilde{\mathsf{m}}_{\mathsf{AC12}}(\cdot|\cdot)$ (Equation 8.3) and $\tilde{\mathsf{m}}_{\mathsf{New}}(\cdot|\cdot)$ (Equation 8.4). It allows discussing which model is the best and how far it is from the optimal attack enumerating all the permutations.

Practically, the simulations take two parameters. The first one is the noise variance σ^2 in Equation 2.3. It represents the amount of noise that is intrinsic to the implementation. The second one is the number of independent operations on which we shuffle, which corresponds to the size η of the secret vector $\boldsymbol{y}$. As a result, we have $\mathsf{H}(\boldsymbol{Y}) = \pi \cdot \mathsf{H}(Y_i) = \eta \cdot \mathsf{H}(Y_i)$. Due to the aforementioned computational limitations, we take values $\eta \in \{2, 4, 6\}$. However, we note that the evaluation of the AC12 attack and our improvement do not suffer from such a limitation, meaning that PI could be evaluated also for a larger η.[4]

Results and discussion. The results of these simulations are reported in Figure 8.1. On the left, we observe the resulting MI and PIs according to the noise parameter σ^2: DPLeak AC12 is the label of the model $\tilde{\mathsf{m}}_{\mathsf{AC12}}(\cdot|\cdot)$ and DPLeak New is the label of the model $\tilde{\mathsf{m}}_{\mathsf{New}}(\cdot|\cdot)$. On the right, we report the ratio between the MI and the PIs. This part

[4]The simulations performed in this work are available to the reviewers as supplementary material and will be published under an open-source license.



(a) $\eta = 2$, **Information** **(b)** $\eta = 2$, PI/MI

(c) $\eta = 4$, **Information** **(d)** $\eta = 4$, PI/MI

(e) $\eta = 6$, **Information** **(f)** $\eta = 6$, PI/MI

Figure 8.1: IT analysis of shuffled implementation Algorithm 3. Normalized IT metrics are on the left. The ratios between the PIs and the MI are on the right.

of the plot is used to highlight the gap between the efficient adversaries' models and the worst-case attack. Based on these simulations, we first observe that our new model improves over the AC12 one. Indeed, the PI for the DPLeak AC12 adversary is always (sometimes significantly) lower than DPLeak New. Second, we observe that for high noise levels, our new model offers a good approximation of the MI while the AC12 one suffers from a bias.

In the rest of this chapter, such small-scale information theoretic evaluations (based on the worst-case MI) will be quite systematically used in order to evaluate and compare different combinations of masking and shuffling. We therefore use this first application to discuss their main pros and cons as follows.

On the negative side: (*i*) these small scale examples only correspond to attacks considering a representative leakage function, which is in not equivalent to proving security in general, and (*ii*) they do not directly apply to the typical sizes of concrete implementations (e.g., $\mathsf{H}(Y_i) = 8$ and $\eta = 16$ for the AES).

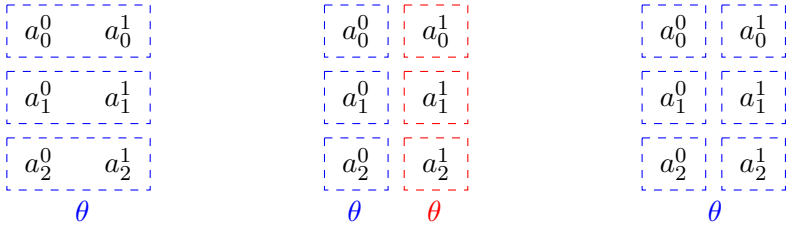
On the positive side, (*i*) information theoretic evaluations as we propose have quite systematically been shown to be excellent indicators of the bounds that can be obtained in masking security proofs: see for example the sequence of papers [SMY09, SVO$^+$10, PR13, DDF14, DFS15] for an illustration, and (*ii*) the same holds (up to constant factors) for the extrapolation from small variables to larger ones, and the concrete attacks we propose do not suffer from computational limitations.

Since all our observations are confirmed for a few (growing) values of d and π, we therefore believe they provide a necessary first step towards the understanding of the masking + shuffling combination of countermeasures which, in particular, improves over the one of Rivain et al. [RPD09], both in terms of the security levels we claim and in terms of the considered attacks' coverage. Proving the bounds we observe in a formal model is an interesting scope for further research.[5]

8.4 Systematic information theoretic analysis

As discussed in Subsection 2.3.1, a masked d-probing secure circuit is composed of two types of of operations: linear ones can be applied

[5] We note that the DPLeak New could possibly be further improved with analytical attacks such as [VGS14]. However, Figure 8.1 shows that for high noise levels on which we will focus, DPLeak New is already close to the worst-case attack, which is in line with the observation in [ADP$^+$20] that such analytical attacks only lead to minor improvements when aiming at recovering ephemeral secrets (like a permutation).



(a) Tuples. **(b) Shares.** **(c) Everything.**

Figure 8.2: Shuffling configurations of a linear encoding for $d = 2$ and $\eta = 3$.

share-by-share, non-linear ones require to mix the shares in a secure manner. We now analyze different approaches to combine masking and shuffling for these two types of operations. We start with paper-and-pencil intuitions to express the security gains such combinations provide in simple terms and/or to revoke some possible options. Then, we evaluate some relevant combinations with an information theoretic analysis. For now we focus on the high-noise regime, where exploiting the permutation leakages does not improve the attacks [VMKS12]. Some simulations taking into account the permutation leakages are reported in Appendix D.1. The case of low noise regime will be discussed based on a real-world case study in Section 8.6.

8.4.1 Linear operations

Based on the notations of Section 2.1, applying a linear operation $\mathsf{op_l}(\cdot)$ to an encoding of the secret vector $\boldsymbol{a}$ of size η consists in applying $\mathsf{op_l}(\cdot)$ independently to all the elements of its share vectors $\boldsymbol{a}^i$. Namely, the output encoding of $\boldsymbol{b}$ is derived such that $b_s^i = \mathsf{op_l}(a_s^i)$ for all $0 \leq i < d$ and $0 \leq s < \eta$. The challenge when combining masking and shuffling for linear layers is to identify in what order should pairs of indexes (i, s) be used to load a_s^i. We consider three possible shuffling configurations that can be applied to a linear layer, as summarized in Figure 8.2 where a color denotes a permutation and a box the pair(s) (i, s) that are accessed deterministically at each cycle of that permutation.

8.4.1.1 Shuffling tuples.

A first straightforward possibility is to shuffle tuples of shares. Only the indexes s of the vectors are shuffled and the shares i of that index are accessed sequentially and deterministically. Algorithm 4 is an instantiation of such a combination and a graphical representation is

given in Figure 8.2a.

Algorithm 4 Masking and shuffling tuples.

Input: $\boldsymbol{a}$ such that $a_s = \sum_{i=0}^{d-1} a_s^i$ and $\mathsf{op}_\mathsf{l}(\cdot)$
Output: $\boldsymbol{b}$ such that $b_s = \mathsf{op}_\mathsf{l}(a_s) = \sum_{i=0}^{d-1} b_s^i$.

$\boldsymbol{\theta} \leftarrow \boldsymbol{\Theta}$ ▷ Perm. generation of size $\pi = \eta$
for c in $[0, \ldots, \eta - 1]$ **do**
  $s \leftarrow \theta_c$
  for i in $[0, \ldots, d - 1]$ **do**
    $b_s^i = \mathsf{op}_\mathsf{l}(a_s^i)$

In terms of security, this shuffling and masking combination is similar to the case where we only shuffle an implementation from Algorithm 3. Namely, because the shares are accessed in order, at the cycle c information on all the shares $a_{\theta_c}^i$ can be combined together to get information on a_{θ_c}, without being impacted by shuffling. So similarly to a shuffled-only implementation, the security of this combination is linear in the number of operations η on which we shuffle:

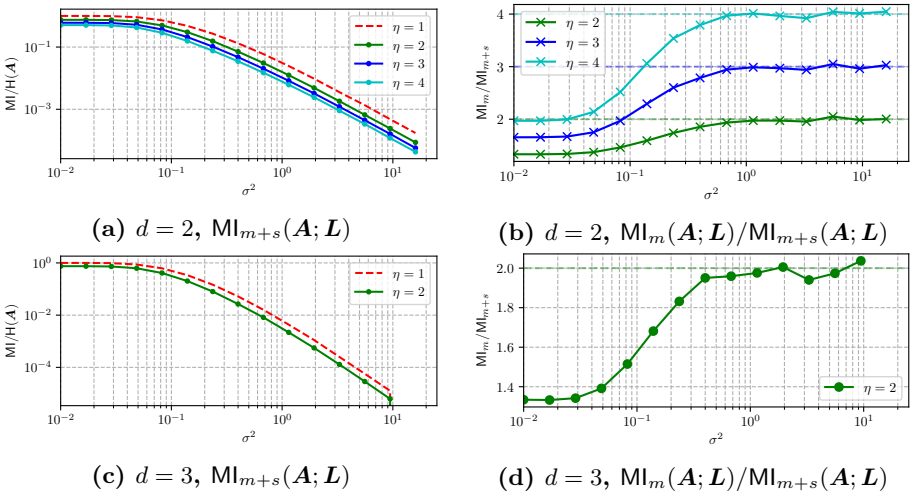
$$N^{\text{tuples}} \geq \frac{cst \cdot \eta}{\prod_{i=0}^{d-1} \mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})}. \tag{8.6}$$

This equation is confirmed by the IT analysis of Algorithm 4 given in Figure 8.3. On the left, we report the MI of a masked and shuffled implementation $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$, for various parameters d and η. On the right, we report the ratio between the MI of a masked-only implementation $\mathsf{MI}_m(\boldsymbol{A}; L)$ and $\mathsf{MI}_{m+s}(\boldsymbol{A}; L)$. Based on Equation 2.1, this ratio is the increase of the (worst-case) attack data complexity that shuffling and masking provide compared to masking only.

We first observe that, as expected from Equation 8.6, the gain is equal to the permutation size $\pi = \eta$ for sufficiently large noise variance. We additionally observe that this gain is independent of d, which confirms that there is no non-trivial interaction between shuffling and masking in this case. Indeed for $\eta = 2$, the gain is equal to 2 for the two and three shares implementations.

8.4.1.2 Shuffling shares.

A natural option to improve the interaction between masking and shuffling is to shuffle the shares of independent variables instead of their tuples. As illustrated in Figure 8.2b and described formally in Algorithm 5, it consists in processing the shares sequentially and



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(b) $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(c) $d = 3$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(d) $d = 3$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Figure 8.3: Shuffling tuples (Algorithm 4): IT analysis.

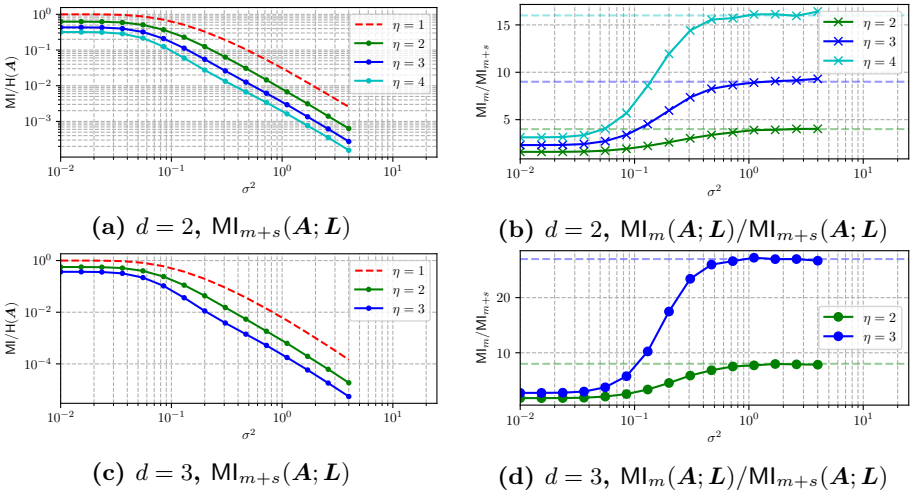
deterministically and in picking up a random permutation for each vector of shares $\boldsymbol{a}^i$. For each share index, the operations $\mathsf{op}_\mathsf{l}(a_s^i)$ are performed with s selected according to a fresh permutation. As a result, the permutation is always applied on independent values.

Algorithm 5 Masking and shuffling shares.

Input: $a_s = \sum_{i=0}^{d-1} a_s^i$ and $\mathsf{op}_\mathsf{l}(\cdot)$
Output: $b_s = \mathsf{op}_\mathsf{l}(a_s) = \sum_{i=0}^{d-1} b_s^i$.

for i in $[0, \ldots, d-1]$ **do**
  $\boldsymbol{\theta} \leftarrow \boldsymbol{\Theta}$ ▷ Perm. generation of size $\pi = \eta$
  for c in $[0, \ldots, \eta - 1]$ **do**
    $s \leftarrow \theta_c$
    $b_s^i = \mathsf{op}_\mathsf{l}(a_s^i)$

Such an approach is beneficial to side-channel security since to obtain information about a secret element a_s, an adversary now has to retrieve at which cycle c the d shares a_s^i are manipulated. Without knowledge of the permutations (e.g., because of sufficiently noisy leakages as we assume for now), she succeeds for a share with probability $\frac{1}{\eta}$, and so with probability $\left(\frac{1}{\eta}\right)^d$ for the d shares. As a result, the shuffling shares method can be interpreted as providing an increase of the noise on each share by a factor equal to the permutation size $\pi = \eta$. Masking then amplifies this emulated noise exponentially. The impact of this



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ **(b)** $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(c) $d = 3$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ **(d)** $d = 3$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Figure 8.4: Shuffling shares (Algorithm 5): IT analysis.

combination on the (worst-case) attack data complexity is therefore given by:

$$N^{\text{shares}} \geq \frac{cst}{\prod_{i=0}^{d-1} \mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})/\eta} = \frac{cst \cdot \eta^d}{\prod_{i=0}^{d-1} \mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})} \cdot \tag{8.7}$$

This equation is confirmed by the IT analysis of Algorithm 5 in Figure 8.4. Indeed for large noise, the $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ of the shuffled shares implementation is η^d times lower than the $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})$ of the masked only implementation. For example, for $\eta = 4$ and $d = 2$, this ratio is of $4^2 = 16$. For $\eta = 3$ and $d = 3$, this ratio is of $3^3 = 27$. Therefore, by using d permutations of size $\pi = \eta$, this solution provides an exponential amplification of the noise emulated thanks to shuffling. For completeness, we report similar results by assuming a leaking permutation in Appendix D.1, Figure D.1. Slightly more noise is required to hide information on the permutation indexes, but we keep the same asymptotic improvement.

8.4.1.3 Shuffling everything.

The two previous options shuffled either over the shares or over the tuples. A last solution is to shuffle jointly all the shares corresponding to all the pairs (i, s) by using a single permutation on $\pi = d \cdot \eta$ elements. This combination is illustrated in Figure 8.2c and formally defined in Algorithm 6.

In terms of side-channel security, recovering information about a secret element a_s without knowledge of the permutation (e.g., because

Algorithm 6 Masking and shuffling everything.

Input: $a_j = \sum_{i=0}^{d-1} a_j^i$ and $\mathsf{op}_\mathsf{l}(\cdot)$
Output: $b_j = \mathsf{op}_\mathsf{l}(a_j) = \sum_{i=0}^{d-1} b_j^i$.

$\boldsymbol{\theta} \leftarrow \boldsymbol{\Theta}$ ▷ Perm. generation of size $\pi = d \cdot \eta$
for c in $[0, \ldots, (d \cdot \eta) - 1]$ **do**
  $(j, i) \leftarrow (\theta_c // d, \theta_c \mod d)$
  $b_j^i = \mathsf{op}_\mathsf{l}(a_j^i)$

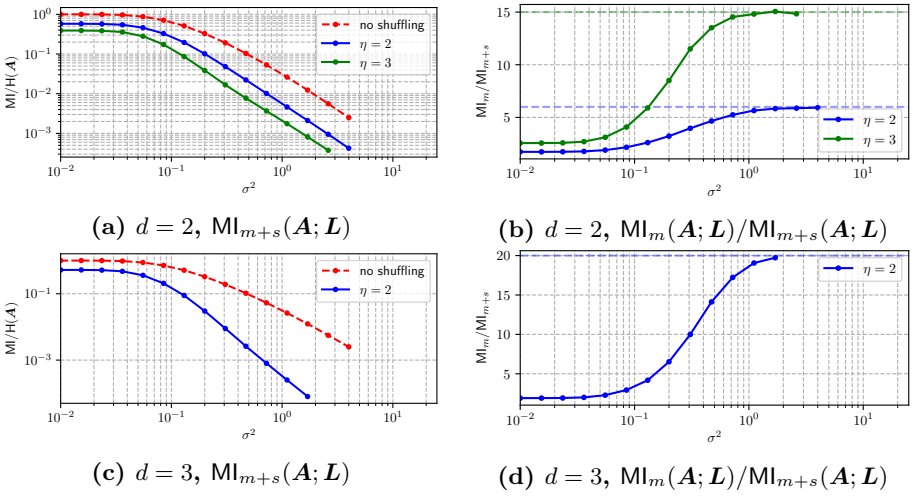
of sufficiently noisy leakages) now requires that the adversary exploits the leakage of all the d shares a_s^i. This implies to find the cycles c where the d pairs (i, s) with $0 \leq i < d$ are used. To do so, she chooses the first share out of the set of $d \cdot \eta$ shuffled values. Since d of these values correspond to a share a_s^i, she succeeds with probability $\frac{d}{d \cdot \eta}$. The second share can be guessed correctly with probability $\frac{d-1}{d \cdot \eta - 1}$ by excluding the first selected share. Eventually, the adversary can obtain information on a_s if she selects correctly all the d shares which happens with probability $\prod_{i=0}^{d-1} \frac{d-i}{d \cdot \eta - i} = \frac{1}{\binom{d \cdot \eta}{d}}$. Hence, the attack data complexity is growing as:

$$N^{\text{everything}} \geq \frac{cst \cdot \binom{d \cdot \eta}{d}}{\prod_{i=0}^{d-1} \mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})}. \tag{8.8}$$

Once again, we confirm this equation with an IT analysis of a simulated implementation of Algorithm 6. For large noise, the security improvement $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L}) / \mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ is equal to $\binom{2 \cdot 2}{2} = 6$ for $d = 2$ and $\eta = 2$, equal to $\binom{2 \cdot 3}{2} = 15$ for $d = 2$ and $\eta = 3$ and finally equal to $\binom{3 \cdot 2}{3} = 20$ for $d = 3$ and $\eta = 2$. This confirm the combinatorial security amplification of this combination of shuffling and masking.

8.4.1.4 Discussion.

We conclude from the previous analyses and experiments that $N^{\text{tuples}} \leq N^{\text{shares}} \leq N^{\text{everything}}$. However, these successive improvements come at the cost of more or larger permutations. Hence, they raise the question of which option provides the best cost vs. security tradeoff, which we will discuss in Section 8.5. For this purpose, a preliminary is to generalize our results from linear operations to non-linear ones, which we tackle in the next section. We note that Rivain et al. used the *shuffling everything* method for their linear layers (and derived a correlation-based bound for this purpose), but they only used the *shuffling tuples* one for the AES Sbox [RPD09]. To the best of our knowledge, the *shuffling shares*



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ **(b)** $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

(c) $d = 3$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ **(d)** $d = 3$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Figure 8.5: Shuffling everything (Algorithm 6): IT analysis.

method is new. As will be shown next, it is also the one leading to the best cost vs. security tradeoff. In particular, it allows avoiding the imbalance between the security levels of linear an non-linear operations, which is caused by the use of two types of shuffling, and forced Rivain et al. to artificially increase the permutation size of the non-linear layers by using dummy operations.

8.4.2 Non-linear operations

The second building block for d-probing secure circuits are non-linear operations. We will consider the standard ISW multiplication for this purpose.[6] In this case, all the pairs (i, j) have to be accessed in order to compute the cross products $a^i \otimes b^j$. We assume a setting where η independent ISW multiplications $c_s = a_s \otimes b_s$, with $0 \leq s < \eta$, have to be computed. In order to perform such operations, it implies that all the triplets (s, i, j) have to be accessed to compute all the $a_s^i \otimes b_s^j$ cross-products. Next, we list different ways to shuffle the computation of these triplets and discuss the different security levels they lead to. We note that such combinations of masking and shuffling for non-linear layers are more complex than for linear ones: there are more possibilities to be considered and their security is sometimes hard to assess. We therefore start by presenting the simplest solution of *shuffling tuples* used by Rivain et al. We then introduce a generic taxonomy of shuffling configurations which allows describing the design space of the masking

[6] Our conclusions apply to other masking schemes based on a similar structure.

+ shuffling combinations, and we illustrate this taxonomy with the shuffling tuples approach. Finally, we prune this design space and focus on a number of solutions that can be viewed as the counterparts of the *shuffling shares* and *shuffling everything* approaches previously described for linear operations. We also argue why this pruning is practically relevant.

8.4.2.1 Shuffling tuples.

This first method is similar to the *shuffling tuples* for linear layers. It is presented in Algorithm 7 where only the accesses to the tuples $\boldsymbol{a}_s$ and $\boldsymbol{b}_s$ are shuffled with a permutation of size $\pi = \eta$. Then the pairs (i, j) are accessed deterministically within these tuples.[7] Similarly to the shuffling tuples for linear layers, this allows an adversary to obtain information on the unshared secrets a_s, b_s and c_s with probability $\frac{1}{\eta}$. The resulting security guarantee is then linear with the size of the permutation similarly to Equation 8.6. This is confirmed by our IT analysis of Algorithm 7 reported in Figure 8.6. There, the ratio between the MI of a masked only ISW multiplication $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})$ and the one of shuffled and masked ISW multiplication $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ is always equal to η. This combination is a straightforward adaptation of the Rivain et al. implementation of masked and shuffled AES SubBytes to the case of ISW multiplications.

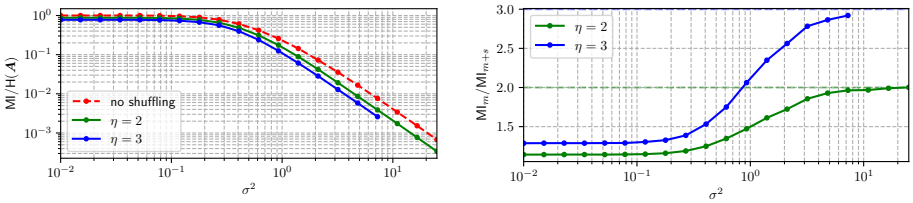
Algorithm 7 Shuffling tuples ISW (configuration $(1^s, 0^i, 0^j)$).

Input: inputs $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\eta-1}\}$ and $\{\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\eta-1}\}$ and randomness $r_s^{i,j}$ defined as: $\forall s, \forall i, \forall j$, such that $i < j$, $r_s^{i,j} \leftarrow \{0, 1\}$ and $r_s^{j,i} = r_s^{i,j}$ and $r_s^{i,i} = 0$.

Output: outputs $\{\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\eta-1}\}$ such that $\forall s \in \{0, 1, \ldots, \eta-1\}, c_s = a_s \otimes b_s$.

```
θ^s ← Θ^s                                   ▷ permutation over η elements
for s in θ^s do
    for i in [0, ..., d − 1] do
        for j in [0, ..., d − 1] do
            c_s^i ← c_s^i ⊕ r_s^{i,j} ⊕ (a_s^i ⊗ b_s^j)
```

[7] The sequence of operations in Algorithm 7 are slightly different than in Algorithm 2. Namely both i and j range from 0 to $d - 1$ and the first loop on $a^i \otimes b^i$ is omitted. The constraints on $r_s^{i,j}$ make these two algorithms equivalent.



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ **(b)** $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Figure 8.6: Shuffling ISW multiplication tuples (Algorithm 7): IT analysis.

8.4.2.2 Shuffling more.

As in the previous subsection, the next step is to shuffle over more operations in order to reach a better security gain. For example, one natural goal would be to amplify the effect of shuffling with masking so that a gain factor of η^d can be maintained for full implementations. For this purpose, we first describe all the possible shuffling and masking combinations which handle the indexes s, i and j independently, with the next shuffling configurations.

Shuffling + masking configurations. At a high level, we describe all the possible combinations of masking and shuffling with Algorithm 8 along with a configuration of the form $(x^\alpha, x^\beta, x^\gamma)$. The algorithm is composed of three nested loops where each loop is responsible for shuffling (or not) one of the indexes s, i or j. In the configuration, the superscripts correspond to the index manipulated by the loop and the position in the triplet corresponds to the order of the loops. For example, the first superscript designates the outermost loop and the third one the innermost loop. Additionally, the x value is a bit set to 1 if the loop is shuffled and 0 otherwise. We note that swapping the indexes i and j leads to the same gadget since the multiplication is commutative. As a result, the configuration $(0^s, 0^i, 0^j)$ denotes an unshuffled implementation, and the previous *shuffling tuples* solution given in Algorithm 7 corresponds to the configuration $(1^s, 0^i, 0^j)$, where only the outer loop on s is shuffled. Finally, the Greek letters in Algorithm 8, namely $\alpha \in A, \beta \in B$ and $\gamma \in \Gamma$, are uniquely set to s, i or j. The corresponding capital letter is the set of values that the index can take. For example, if $\alpha = s$ then $A = [0, \ldots, \eta - 1]$ and if $\alpha = i$ or $\alpha = j$ then $A = [0, \ldots, d - 1]$. Hence, $\boldsymbol{\theta}^\alpha \xleftarrow{x^\alpha} \boldsymbol{\Theta}^\alpha$ is a fresh permutation of the elements in A if x^α is set to one and is A otherwise.

Algorithm 8 Generic shuffled & masked ISW multiplications.

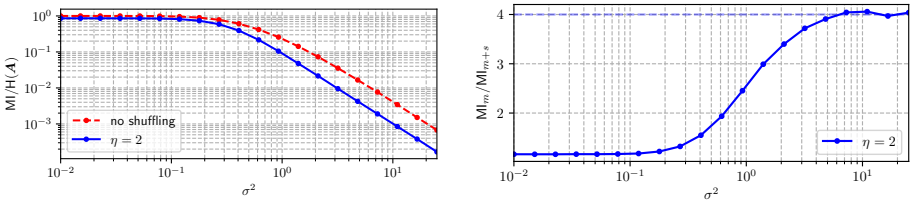
Input: inputs $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\eta-1}\}$ and $\{\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\eta-1}\}$, shuffling configuration $(x^\alpha, x^\beta, x^\gamma)$ with $\alpha, \beta, \gamma \in \{s, i, j\}$ and randomness $r_s^{i,j}$ defined as: $\forall s, \forall i, \forall j$, such that $i < j$, $r_s^{i,j} \leftarrow \{0,1\}$ and $r_s^{j,i} = r_s^{i,j}$ and $r_s^{i,i} = 0$.

Output: outputs $\{\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\eta-1}\}$ such that $\forall s \in \{0, 1, \ldots, \eta-1\}, c_s = a_s \otimes b_s$.

for α in $\theta^\alpha \xleftarrow{x^\alpha} \boldsymbol{\Theta}^\alpha$ **do** ▷ one permutation over A
  for β in $\theta^\beta \xleftarrow{x^\beta} \boldsymbol{\Theta}^\beta$ **do** ▷ $|A|$ permutations over B
    for γ in $\theta^\gamma \xleftarrow{x^\gamma} \boldsymbol{\Theta}^\gamma$ **do** ▷ $|A| \cdot |B|$ permutations over Γ
      $c_s^i \leftarrow c_s^i \oplus r_s^{i,j} \oplus (a_s^i \otimes b_s^j)$

Pruning the configuration space. Based on the previous configurations, there are 6 possible ways to order the loops and, for each of these orderings, there are 2^3 possibilities of shuffling. This leads to a total of 48 cases. In order to reduce the number of configurations to investigate in detail, we first notice that in Algorithm 8, shuffling the indexes i or j never leads to a security improvement. Indeed, shuffling on i (resp., j) only means shuffling the accesses to the shares a_s^i (resp., b_s^j). Because in an encoding $\boldsymbol{a}_s$ of a_s, the position of the shares has no effect on security, an adversary can obtain information on a_s by observing leakages on all the shuffled shares a_s^i without being impacted by the shuffling of i (resp., j). We further note that in the high-noise regime (that we assume for now), this observation also holds when we shuffle i and j. In this case, the cross-products are shuffled and information on each of the a_s^i involved in these cross-products is obtained from the η observations of the shuffled $a_{\boldsymbol{\theta}^s}^i$. But information on a_s can still be recovered from the input/output tuples of the multiplication, without being impacted by the shuffling of i. So this configuration makes the information of the cross-products harder to exploit, while it is known that the information provided by these cross-products is dominated by the information of the multiplications' input/output tuples when the noise is large [CS19]. Therefore, we next limit our investigations to configurations with 0^i and 0^j.

This reduces the set of combinations to 3 possibilities. We next list them and discuss their security. The first one is $(1^s, 0^i, 0^j)$ and corresponds to the aforementioned *shuffling tuples* (Algorithm 7) for which the security impact is only linear in the size of the permutation. The second configuration is $(0^i, 1^s, 0^j)$ (resp., $(0^j, 1^s, 0^i)$). For this



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ **(b)** $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Figure 8.7: Shuffling ISW shares (Algorithm 8, config. $(0^i, 0^j, 1^s)$): IT analysis.

configuration the security of the inner loop variable b^j (resp., a^i) differs from the one on the outer loop variable a^i (resp., b^j) and the security of the inner loop variable is similar to the *shuffling tuples* option, which is not desirable. Finally, the third configuration is $(0^i, 0^j, 1^s)$. It is similar to the *shuffling shares* of linear layers, where the permutation is applied to $\pi = \eta$ independent elements. In this case, every loading of the input shares a_s^i and b_s^j as well as the updates of c_s^i are shuffled among the η independent ISW multiplications. As a result, the information on each of these operations is reduced by a factor η that is later amplified by masking. Therefore, it provides an exponential gain factor η^d as in Equation 8.7. This gain is confirmed by the IT analysis depicted in Figure 8.7 for $d = 2$ and $\eta = 2$, where the ratio between $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})$ and $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ for high enough noise is of $2^2 = 4$.

8.4.2.3 Shuffling everything.

A last natural question is to know whether it is possible to obtain a better security, similar to the one obtained for the *shuffling everything* of linear layers. For these operations, the improvement was obtained by shuffling jointly all the pairs (i, j) instead of independently. For the ISW multiplications, this can be done by merging some of (or all) the loops in Algorithm 8. To list the possible combinations when two loops are merged, we use Algorithm 9 along with notations of the form $(x^{\alpha,\beta}, x^{\gamma})$. For example, $(x^{i,j}, x^s)$ means that the outer loop is a merge of the loops on i and j and hence over d^2 elements. More precisely, in Algorithm 8 the operator $\times$ denotes the Cartesian product and $\boldsymbol{\theta}^{\alpha,\beta}$ is a permutation over the set $A \times B$. Alternatively, all loops can be merged together into a single one operating on $d^2 \cdot \eta$ elements. We denote this configuration as $(x^{s,i,j})$ and detail it in Appendix D.2, Algorithm 10.

Based on these configurations, a first observation is that loops on i and j must be merged together in order to avoid the same asymmetry

Algorithm 9 Generic shuffled masked ISW multiplications with 2 loops merged.

Input: inputs $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\eta-1}\}$ and $\{\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\eta-1}\}$, shuffling configuration $(x^{\alpha,\beta}, x^{\gamma})$ with $\alpha, \beta, \gamma \in \{s, i, j\}$ and randomness $r_s^{i,j}$ defined as: $\forall s, \forall i, \forall j$, such that $i < j$, $r_s^{i,j} \leftarrow \{0,1\}$ and $r_s^{j,i} = r_s^{i,j}$ and $r_s^{i,i} = 0$.

Output: outputs $\{\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\eta-1}\}$ such that $\forall s \in \{0, 1, \ldots, \eta-1\}, c_s = a_s \otimes b_s$.

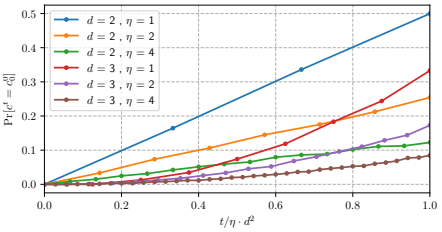
for α, β in $\theta^{\alpha,\beta} \xleftarrow{x^{\alpha,\beta}} \boldsymbol{\Theta}^{\alpha,\beta}$ **do**
  for γ in $\theta^{\gamma} \xleftarrow{x^{\gamma}} \boldsymbol{\Theta}^{\gamma}$ **do** ▷ $|A|$ permutations over B
    $c_s^i \leftarrow c_s^i \oplus r_s^{i,j} \oplus (a_s^i \otimes b_s^j)$

issues as in the previous *shuffling shares* approach. This reduces the possibilities of merging loops to three combinations that are $(1^s, 1^{i,j})$, $(1^{i,j}, 1^s)$ and $(1^{s,i,j})$. However, all these options induce a non-uniform permutation of the outputs shares. Figure 8.8 illustrates this effect for various parameters d and η. There, the y-axis is the probability that a given valid output share (in this case c_0^0) is generated during the t-th iteration of the algorithm. The x-axis is the iteration t that is normalized by the total number of iterations $\eta \cdot d^2$. To explain this non-uniformity, we stress that one output share c_0^0 is valid once it has been updated d times in Algorithm 9. Therefore, during the first few iterations it is unlikely that the current operation is the d-th update of c_0^0 as it becomes for the last few iterations. As a result, in all the plots of Figure 8.8, the probability to generate c_0^0 globally increases with t, with some differences depending on the shuffling configuration.

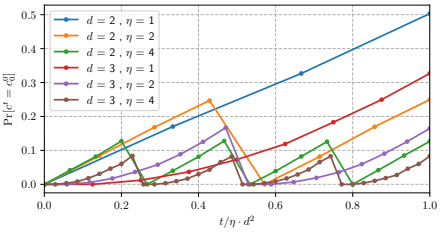
We note that such non-uniform permutations of the output shares may offer some (even good) level of security. Yet, they suffer from the significant drawback that their analysis would require analyzing complex permutation biases that are specific to the configurations and the parameters d and η. We therefore rule out these options in our following investigations for usability reasons.

8.4.3 Summary of our investigations

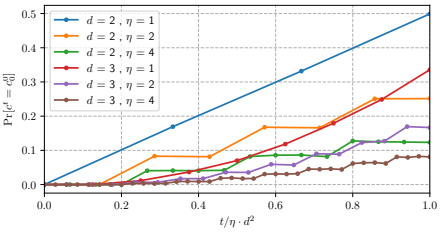
Table 8.1 and Table 8.2 contain a summary of the different combinations of masking and shuffling we considered, when the secret vectors are of size η. The table reports the security gain factor compared to a masked-only implementation (i.e., $\mathsf{MI}_m/\mathsf{MI}_{m+s}$), the size of the permutation(s) used (i.e., π) and the number of fresh permutation(s) needed (i.e.,



(a) Configuration $(1^{s,i,j})$**.**



(b) Configuration $(1^{s}, 1^{i,j})$**.**



(c) Configuration $(1^{i,j}, 1^{s})$**.**

Figure 8.8: Proba. to generate output share c_0^0 at cycle t when merging ISW loops.

	$\mathsf{MI}_m/\mathsf{MI}_{m+s}$	π	# perm.
shuffling tuples	η	η	1
shuffling shares	η^d	η	d
shuffling everything	$\binom{d\cdot\eta}{d}$	$d \cdot \eta$	1

Table 8.1: Summary of the shuffling + masking combinations in linear layers.

perm.). As mentioned above, the *shuffling everything* approach cannot be straightforwardly applied to non-linear layers. Hence, we next focus on the *shuffling shares* solution which allows the same exponential security gain for both linear and non-linear layers, enabling balanced designs (and evaluate the *shuffling tuples* option for comparison purposes).

8.5 Time versus security for shuffled ISW

In the previous section, we have discussed how to amplify the impact of shuffling by combining it with masking, for both linear and non-linear operations. In this section, we focus on the practical instantiation of the *shuffling tuples* and *shuffling shares* methods, with a focus on 32-bit software platforms, and we compare them to a masked-only implementation. That is, we study Algorithm 8 with configurations

	$\mathsf{MI}_m/\mathsf{MI}_{m+s}$	π	# perm.
shuffling tuples	η	η	1
shuffling shares	η^d	η	d^2
shuffling everything	?	?	?

Table 8.2: Summary of the shuffling + masking combinations in non-linear layers.

$(0^i, 0^j, 1^s)$, $(1^s, 0^i, 0^j)$ and $(0^s, 0^i, 0^j)$ in the context of bitslice software implementations. We first detail the randomness that is required by such implementations. Then, we describe the parameters that influence their execution time and their security level. Finally, we propose concrete performance evaluations and leverage them in order to discuss general guidelines for combinations of masking and shuffling that best trade performance and security.

As a preliminary remark, we mention that when protecting a cryptographic implementation with shuffling, a preliminary (cipher-specific) challenge is to find independent operations that can be executed in parallel. In order to keep the following discussions independent of the primitive to protect, we therefore consider a general use case where we aim to implement #AND ISW multiplications in parallel. As will be clear next, it allows us to draw general conclusions about how masking and shuffling should be combined (i.e., which of the countermeasures should use the available parallelism in priority), which can then be directly translated into concrete guidelines for implementing actual ciphers (or possibly modes of operations, in case they offer additional levels of parallelism).

8.5.1 Randomness requirements

The randomness required to combine masking and shuffling is composed of two terms. The first one is due to masking and remains independent of the shuffling parameter η. For the masked (only) ISW multiplication from Algorithm 2, $d \cdot (d-1)/2$ random bits per bitwise multiplication are needed. This means that #AND times more random bits are needed in our context. The second term is due to shuffling and corresponds to the randomness needed to generate the permutation(s) $\boldsymbol{\theta}$. Precisely, a permutation on η elements can be generated with $\eta \cdot \log_2 \eta$ random bits [VMKS12]. When combined with masking by *shuffling tuples*, a single of these permutations must be generated, as reported in Table 8.1 and Table 8.2. When combined with masking by *shuffling shares*, a total of d^2 of these permutations must be generated. As a result, the *shuffling*

Parameter	Description
#AND	Number of masked bitwise AND to operate
bs	Effective size of the registers
$\pi = \eta$	Size of the permutations such that $\eta = \#\text{AND}/bs$
d	The number of shares
r	Number of cycles to generate 32 random bits
$\mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})$	Mutual information on a single unshuffled shared bit

Table 8.3: Summary of parameters for bitslice masking and shuffling.

shares strategy requires a total amount of random bits given by:

$$\underbrace{\left(\#\text{AND} \cdot \frac{d \cdot (d-1)}{2}\right)}_{\text{masking}} + \underbrace{\left(d^2 \cdot \eta \cdot \log_2 \eta\right)}_{\text{shuffling}}. \tag{8.9}$$

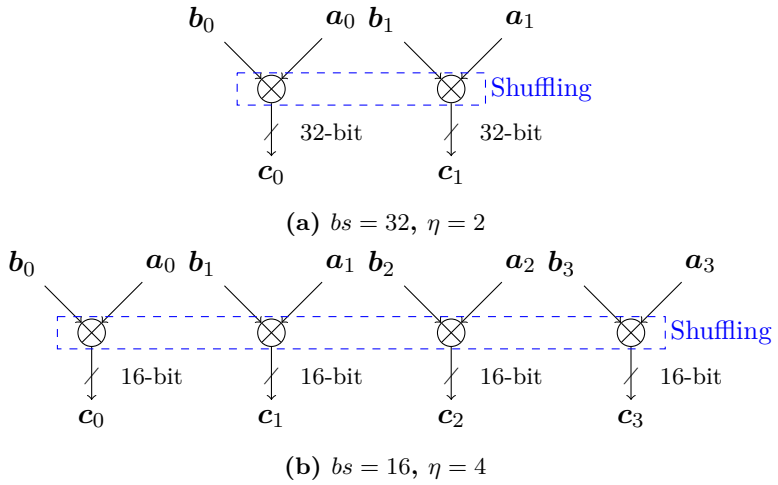
8.5.2 The bitslice masking + shuffling design space

We now introduce the different parameters that influence the execution time as well as the security level of masked and shuffled bitslice implementations. These parameters are summarized in Table 8.3, where the first block contains parameters that are under control of the implementers and the second one contains parameters that depend (partially) on the software platform used.

We next detail these parameters and their interactions.

Next, we observe that in contrast with bitslice implementations that favor parallelism, shuffling rather applies to serialized independent operations: the more such operations, the larger the size of the permutation and therefore the impact of shuffling. As a result, the main question when combining bitslice masking and shuffling is whether one should favor serialization or parallelism.

In order to answer this question, we will analyze the impact of the "effective size" of the register, denoted as bs, which is the parameter reflecting the tradeoff between serialization and parallelism. It is defined as the number of useful bits in a single register, and therefore corresponds to the number of Boolean operations that are parallelized. The maximum value of bs is given by the physical register size. Therefore, on a 32-bit software platform, we have $bs \leq 32$. Yet, the designer could also reduce bs to increase the permutation size η that is equal to $\#\text{AND}/bs$. For example, Figure 8.9 illustrates two options in the masking + shuffling design space for #AND=64. The first option (Figure 8.9a) is to maximize the parallelism with $bs = 32$ and so $\eta = 2$.



(a) $bs = 32$, $\eta = 2$

(b) $bs = 16$, $\eta = 4$

Figure 8.9: Different options in the design space for #AND=64.

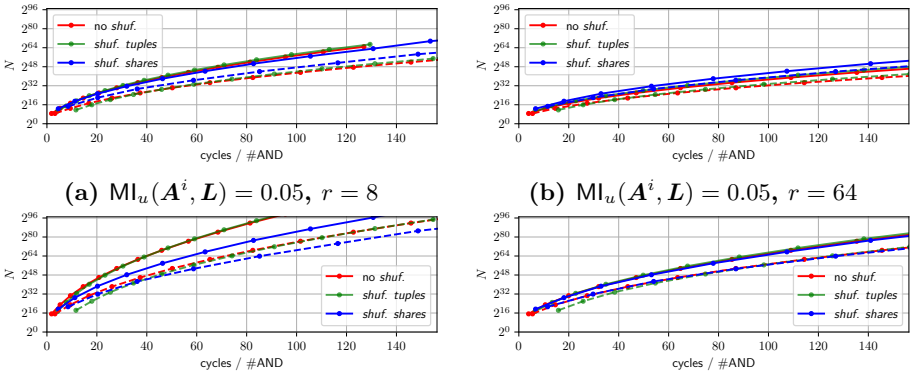
A second option (Figure 8.9b) is to decrease the parallelism with $bs = 16$ and to increase the permutation size up to $\eta = 4$.

The other parameters that we need to evaluate the performance vs. security tradeoff of masked and shuffled implementations are the randomness cost and the noise level. Precisely, and as described in Subsection 8.5.1, both masking and shuffling require randomness. Hence, the throughput at which random bits are generated has an impact on the cycle count. We introduce the parameter r that is the latency in cycles needed to generate 32 random bits once requested. As for the noise level, we quantify it with the amount of information that is available on a single share of an unprotected implementation $\mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})$.

8.5.3 Performance evaluation and discussion

In order to evaluate the security vs. performance tradeoff of the *shuffling tuples* and *shuffling shares* options, we complement the previous security evaluations by measuring the total cycle count of protected implementations running on a 32-bit ARM Cortex-M4 with the design parameters from Table 8.3.[8] Precisely, when only masking we use a bitslice instantiation of Algorithm 2 that is repeated η times to operate #AND bitwise secure multiplications. The security level is then given by Equation 2.7. For the *shuffling tuples* strategy, a similar approach is used but the η ISW multiplications are performed out of order thanks to the generation of a single permutation. The security level is then given

[8]Implementations are written in C and compiled with the -O3 optimization flag.



(a) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.05$, $r = 8$

(b) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.05$, $r = 64$

(c) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.005$, $r = 8$

(d) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.005$, $r = 64$

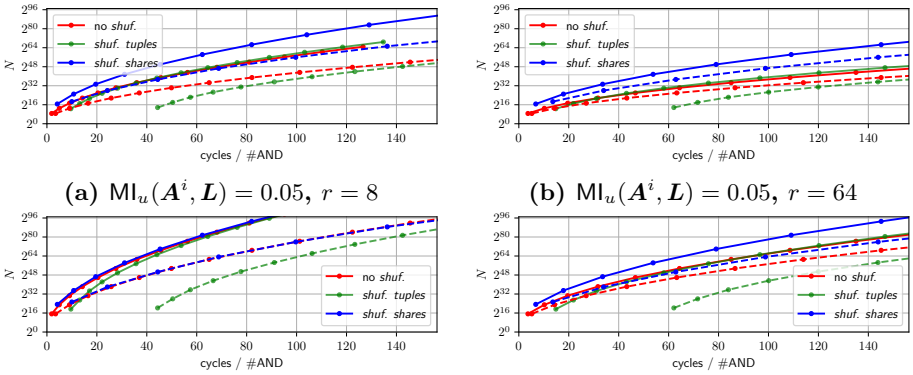
Figure 8.10: Cycles vs. security for 128 bitwise ANDs for various noise levels and randomness cost. Continuous are for $bs = 32$ and dashed ones for $bs = 16$.

by Equation 8.6. For the *shuffling shares* strategy, we use an efficient implementation (Appendix D.2, Algorithm 11) that does not require to pre-compute randomness but rather generates it on-the-fly. Its security level is given by Equation 8.7.[9]

The resulting execution time versus security curves are reported in Figure 8.10 for #AND=128 and in Figure 8.11 for #AND=512. The x-axis is the number of cycles used to perform the secure multiplications normalized by #AND. The y-axis reports the data complexity of the worst-case attack. Each data point is for a different masking order d, with the leftmost being $d = 2$ and increasing with steps of one when moving to the right. The red curves are for masked-only implementations, green ones for *shuffling tuples* and blue ones for *shuffling shares*. Continuous lines are for $bs = 32$, hence using the full physical register. The dashed curves are for $bs = 16$, hence doubling the serialization.

We draw two general conclusions regarding the combination of masking and shuffling from these plots. First, the choice of taking $bs = 32$ is always better than taking $bs = 16$. That is for a fixed execution time (x-axis), the resulting attack data complexity is always larger for $bs = 32$ than for $bs = 16$, and this trend holds for lower bs

[9] We stress that the comparative value of our information theoretic analyses is due to the fact that the constants they imply are independent of the masking and shuffling parameters (i.e., they only depend on the size of the target intermediate variable and success rate). We also recall that they are only valid under a sufficient noise. The concrete investigation of a low-noise case study is given in the next section.



(a) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.05$, $r = 8$

(b) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.05$, $r = 64$

(c) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.005$, $r = 8$

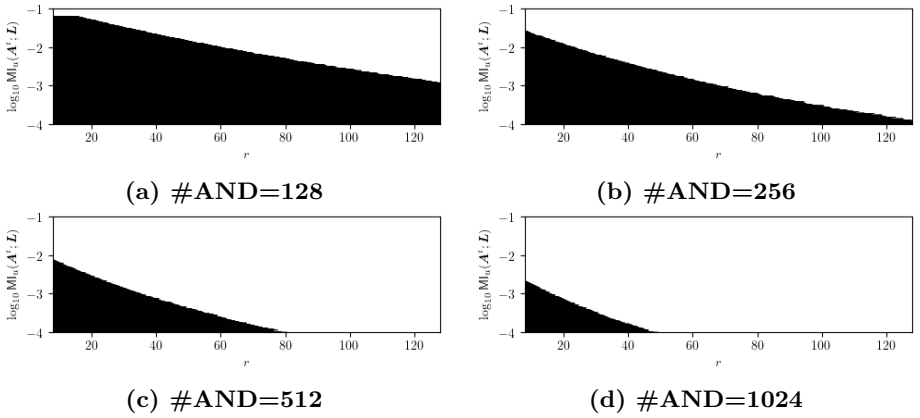
(d) $\mathsf{MI}_u(\boldsymbol{A}^i, \boldsymbol{L}) = 0.005$, $r = 64$

Figure 8.11: Cycles vs. security for 512 bitwise ANDs for various noise levels and randomness cost. Continuous are for $bs = 32$ and dashed ones for $bs = 16$.

values. It implies that a designer should first favor the parallelization of the masked computations he has to perform, and shuffle what is left to serialize.[10] Second, the combination of masking and shuffling benefits from a larger #AND. For example, if 80 cycles are spend per bitwise multiplication, the resulting security is about 2^{40} for #AND=128 for *shuffling shares* according to Figure 8.10b. For the same number of clock cycles per bitwise multiplication, having #AND=512 provides a security around 2^{50} according to Figure 8.11b.

Masking only vs. shuffling tuples. By comparing the masked-only implementation with the *shuffling tuples* strategy, we observe that *shuffling tuples* does not bring a significant gain. That is, for a fixed performance level (i.e., value of the x-axis), *shuffling tuples* at best brings a marginal improvement. This can be explained by the fact that *shuffling tuples* is always slower than masking only due to the permutation generation, while only bringing a gain of η in attack complexity. Increasing this security gain would require to reduce bs, and so to favor serialization. But as shown above, this degrades the time vs. security tradeoff which rather pushes for masking with maximum parallelism.

[10] One corollary of this observation is that (for high enough noise levels as we consider), the use of dummy operations in order to increase the amount of operations to shuffle does not pay off: increasing the number of shares is a better strategy.



(a) #AND=128 **(b) #AND=256**

(c) #AND=512 **(d) #AND=1024**

Figure 8.12: Masking + *shuffling shares* design space for a 64-bit security target and $bs = 32$. Masking only is the most efficient solution in black areas.

Masking only vs. shuffling shares. Interestingly, the conclusion is more shaded when considering the *shuffling shares* approach. In this case, the interest of combining countermeasures essentially depends on the randomness cost r, noise level $\mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})$ and amount of independent operations available #AND. That is, for expensive randomness, relatively low noise (still sufficient for the countermeasures to be effective) and high #AND, *shuffling shares* is the best solution. Whenever decreasing the randomness cost or noise level, or decreasing #AND, this advantage vanishes and masking-only can become the best option. (e.g., in Figure 8.10c). We summarize this design space with Figure 8.12, where the x-axis is the randomness cost and the y-axis is the noise level $\mathsf{MI}_u(\boldsymbol{A}^i; \boldsymbol{L})$. Black areas represent contexts where masking alone is more efficient than *shuffling shares* in order to reach a given security target (here 64-bit). White areas represent contexts where *shuffling shares* is more efficient. Black areas are for both cheap randomness and high noise levels (and so lie in the bottom left). By increasing the number of independent multiplications #AND (and accordingly η), this area is reduced and the masking + shuffling approach becomes beneficial.

8.6 Real-world (low-noise) case study

Based on the previous evaluations, we can conclude that the new (*shuffling shares*) combination of masking and shuffling that we propose in this chapter can indeed be an interesting asset for the design of

side-channel secure implementations. It could in particular be quite effective to protect bitslice designs based on large permutations where a lot of parallel operations can be easily identified. Yet, these evaluations have been performed under the assumption of a sufficient noise level which may not be observed in practice. In this section, we therefore complement these analyses with a real-world evaluation of the masking + shuffling countermeasure implemented in a commercial MCU.

The motivation for this investigation comes from the intuition that shuffling could be an option to protect devices where the physical noise is not sufficient for masking to be directly effective. In order to clarify whether this expectation is founded, we propose a worst-case evaluation where the adversary has full knowledge and control of her target implementation during a profiling phase. In particular, she has knowledge of the randomness used to generate the permutations enabling profiled attacks against the shuffling as in Section 8.3. We conclude this section by briefly discussing how our observations evolve when relaxing the adversarial capabilities, in the spirit of a backwards evaluation [ABB+20].

8.6.1 Target and measurement setup

We investigate the implementation of ISW multiplications enhanced with the *shuffling shares* approach, executed on the 32-bit ARM Cortex-M3 of the STM32 VLDISCOVERY commercial board. In order to obtain clean measurements, we modified the board and removed all the decoupling inductances and capacitors. We used the available slot to add an 8[MHz] external crystal and derived the maximum 24[MHz] internal clock from it. In order to measure the side-channel leakage $\boldsymbol{L}$, we placed a current probe (i.e., the CT1 from Tektronix 1[GHz]) on the dedicated jumper between the n-board power regulator and the MCU power pins. Eventually, we sampled this signal with a 12-bit resolution at a sampling rate of 500[MSamples/s] thanks to a PicoScope 5244D. As shown next, the resulting setup allows collecting measurements with low noise.

8.6.2 Worst-case adversary

We evaluated this implementation with the adversary of Section 8.3 and more precisely with the model $\tilde{\mathsf{m}}_{\mathsf{New}}(\boldsymbol{y}|\boldsymbol{l})$ from Equation 8.4. For this purpose, we extract information on the secret variables using a standard template attack in a principal subspace. Precisely, we estimate the leakage PDFs with Gaussian templates after dimensionality reduction using Linear Discriminant Analysis (LDA) [APSQ06].

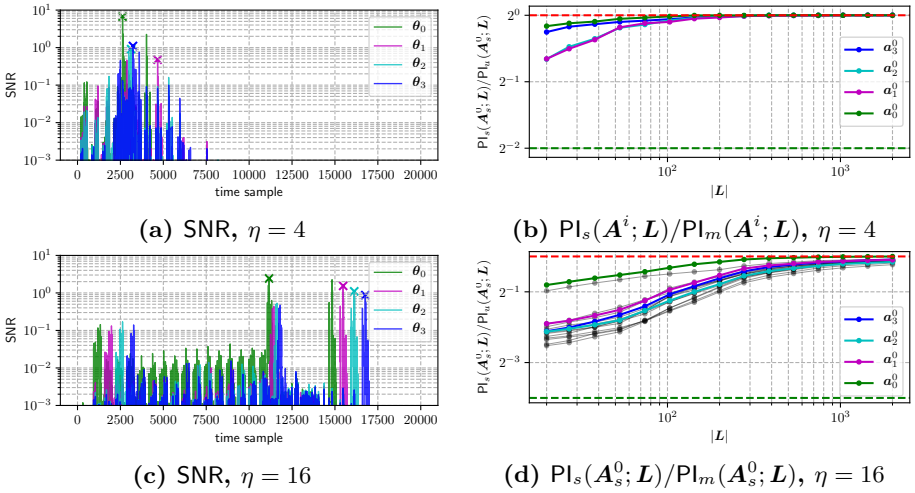
The resulting leakage analysis is reported in Figure 8.13 for implementations with $\eta = 4$ and $\eta = 16$. On the left figures, the x-axis is the time and the y-axis is the SNR (in log-scale) of the indexes of the permutation $\boldsymbol{\theta}_c$. We observe that the maximum value for each of them is around 1 and many other dimensions lead to an SNR two orders of magnitude lower (especially for $\eta = 16$). On the right, we report the number of dimensions of the leakage vector exploited by the adversary $|\boldsymbol{L}|$ on the x-axis and the y-axis reports (in log-scale) the information reduction per share. Namely it is the ratio between the PI of elements in the first share vector $\boldsymbol{a}^0$ for a shuffled implementation (i.e., $\hat{\mathsf{PI}}_s(\boldsymbol{A}_c^0; \boldsymbol{L})$) and the one of an unshuffled implementation (i.e., $\hat{\mathsf{PI}}_u(\boldsymbol{A}_c^0; \boldsymbol{L})$). The green dashed curves represents the expected impact of the suffling in case permutation indexes do not leak (as assumed in Section 8.4), namely $1/\eta$. The red dashed curve equals 1 and corresponds to a level of leakage such that shuffling is ineffective.[11]

As the number of dimensions exploited by the adversary increases, this ratio gets close to one. For $\eta = 4$, it is stuck at 1, meaning that the adversary directly gains full knowledge of the permutation and is therefore not impacted by the shuffling. For $\eta = 16$, a similar behavior is observed but a larger number of dimensions must be exploited. For $|\boldsymbol{L}| = 2,000$, the average ratio is equal to 0.92, meaning that the information per share is only reduced by 8%. For example, it implies that the gain factor of this combination of countermeasures with 8 shares is around 2 (while it would be 16^8 with a sufficient noise). We conclude that this implementation lies in the low noise region of Figure D.1 in Appendix D.1, where a limited gain is obtained by combining masking & shuffling.

Eventually, we insist that these experiments do not show that masking and shuffling cannot be implemented securely (the previous sections showed the opposite). What they show is that when the noise level provided by a leaking device is too low for masking to be effective, it is in general too low for shuffling to be effective as well. It may even be more challenging to implement shuffling securely on low-cost embedded devices since the memory access it relies on can be more leaky than (for example) bitslice computations in such devices.

So overall, it remains an important challenge to ensure a sufficient level of noise on embedded MCUs, so that masking, shuffling and their combination becomes effective. Reaching this goal with existing low-cost devices (similar to the ARM Cortex-M3 we analyzed) is an interesting research direction. Yet, the recurrent difficulties caused by low physical

[11] We report also the PI on the permutation indexes in Appendix D.3, Figure D.2.



(a) SNR, $\eta = 4$

(b) $\mathsf{PI}_s(\boldsymbol{A}^i;\boldsymbol{L})/\mathsf{PI}_m(\boldsymbol{A}^i;\boldsymbol{L})$, $\eta = 4$

(c) SNR, $\eta = 16$

(d) $\mathsf{PI}_s(\boldsymbol{A}_s^0;\boldsymbol{L})/\mathsf{PI}_m(\boldsymbol{A}_s^0;\boldsymbol{L})$, $\eta = 16$

Figure 8.13: IT analysis of real permutation leakages. Left: SNR in function of the time. Right: information loss in function of the # of dimensions exploited.

noise for the implementation of side-channel countermeasures also suggest that solving the issue at the technological level by guaranteeing a minimum level of intrinsic noise on security MCUs could be highly beneficial in terms of security vs. performance tradeoff.

Conclusion 9

In this thesis, I illustrated that by granting the evaluator full control of the target, he can quantitatively approach the worst-case side-channel security of implementations. More precisely in the case of software implementations, assuming that the leakage on each of the shares is independent allows one to express the leakage distribution in such a way that it can be efficiently approximated.

9.1 Design for low-cost MCUs

My analyses suggest that on low-cost MCUs, reaching high side-channel security is highly non trivial. Indeed, countermeasures against differential power analysis, such as masking and shuffling, require physical noise which is inherently lacking in these devices. Concretely, I did not demonstrate that masking is ineffective but that a large number of shares might be needed (see Chapter 7) which implies (possibly) prohibitive overheads for a single primitive execution. This conclusion therefore motivate approaches at the mode level such as [BBC+20a] which allow amortizing the cost of the countermeasures.

Even if my thesis primarily focuses on the evaluation of countermeasures, the results raise questions about designs. While the design of an implementation and its evaluation are usually assumed to be two independent successive tasks, I gain the conviction that a join effort between design and evaluation may be beneficial. As an example, the designer has to select the number of shares for an implementation, which depends on the information that the evaluator is able to extract. To a larger extent, Chapter 8 illustrates that the design space is large and that the optimization of the cost vs security tradeoffs depends on physical parameters. With the open-source evaluation tools coming with this thesis, my hope is to enable faster iterations between design and evaluation teams, resulting in more secure and more efficient implementations.

9.2 Open challenges

Bounding information. Providing an upper bound for the MI between the leakage and the cryptography enables to bound formally the complexity of worst-case attacks against a given implementation. In Chapter 4, we provided such a bound for discrete leakage based on the modeling of the leakage with histograms. However, the convergence of the bound badly scales with the number of dimensions considered in the leakage making its tightness a disadvantage for its practical usage. Providing bounds of such a form with a faster converge is so an open important challenge.

Quantify independence assumption. For many chapters, I assumed that the leakage on each of the share is independent hence the corresponding methods are order-preserving. For the investigated implementations, it appeared that the lack of noise was the main weakness, meaning that even if no lower order flaw is present, implementations were still weak. Yet, the importance of independence assumption on the worst-case security depends on the noise level [Sta18]. To be fully effective even when lower-order flaws are of importance, (close to) worst-case tools should not be order preserving. This could be achieved by leveraging deep-learning which escapes these limitation or by integrating possible low-order leaks in factor graphs of SASCA.

Hardware implementations. In this thesis, I mainly focus on the evaluation of software implementation. The resulting tools exploited the (reasonable) assumption that the leakage on each of the shares is independent since these are processed serially. However in hardware implementations, the shares are generally processed in parallel. Another future line of research could leverage the knowledge of the source code and randomness during profiling but without using independence assumption making the tools also suitable for hardware implementations.

Simple power analysis security. In this thesis, I focus on the resistance of protected implementations against differential power analysis where the adversary can observe leakage while she controls the plaintexts. Yet, simple power analysis, where the adversary can only observe (averaged) leakage of a few different inputs, can also be used to reach side-channel security. While evaluating DPA essentially consists in evaluating the ratio between signal and noise, evaluating of SPA requires to know the noise free leakage function. This makes the evaluation dependent on the measurement setup and computational resources of the

evaluator [BMPS21] and bounding such attacks in a consistent manner appears as another challenge that should be addressed.

Personal Related Papers

[ABG+21] Melissa Azouaoui, Olivier Bronchain, Vincent Grosso, Kostas Papagiannopoulos, and François-Xavier Standaert. Bitslice masking and improved shuffling: How and when to mix them in software? *IACR Cryptol. ePrint Arch.*, page 951, 2021.

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ASCAD with a single side-channel trace. *IACR Cryptol. ePrint Arch.*, page 817, 2021.

[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.

[BMDS21] Olivier Bronchain, Loïc Masure, François Durvaux, and François-Xavier Standaert. Efficient profiled attacks on masking scheme, extended. 2021.

[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.

[BS21] Olivier Bronchain and François-Xavier Standaert. Breaking masked implementations with many shares on 32-bit software platforms or when the security order does not matter. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):202–234, 2021.

[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.

[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.

Other Personal Papers

[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+20] 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.

[BDFS18] Olivier Bronchain, Louis Dassy, Sebastian Faust, and François-Xavier Standaert. Implementing trojan-resilient hardware from (mostly) untrusted components designed by colluding manufacturers. In *ASHES@CCS*, pages 1–10. ACM, 2018.

[BFL+21] Olivier Bronchain, Sebastian Faust, Virginie Lallemand, Gregor Leander, Léo Perrin, and François-Xavier Standaert. MOE: multiplication operated encryption with trojan resilience. *IACR Trans. Symmetric Cryptol.*, 2021(1):78–129, 2021.

[BMPS21] Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. Improved leakage-resistant authenticated encryption based on hardware AES coprocessors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):641–676, 2021.

[BSS21] Olivier Bronchain, Tobias Schneider, and François-Xavier Standaert. Reducing risks through simplicity: high side-channel security for lazy engineers. *J. Cryptogr. Eng.*, 11(1):39–55, 2021.

[KBBS21] Dina Kamel, Davide Bellizia, Olivier Bronchain, and François-Xavier Standaert. Side-channel analysis of a learning parity with physical noise processor. *J. Cryptogr. Eng.*, 11(2):171–179, 2021.

[MBS21] Charles Momin, Olivier Bronchain, and François-Xavier Standaert. A stealthy hardware trojan based on a statistical fault attack. *Cryptogr. Commun.*, 13(4):587–600, 2021.

[OBCS21] Charles-Henry Bertrand Van Ouytsel, Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. How to fool a black box machine learning based side-channel security evaluation. *Cryptogr. Commun.*, 13(4):573–585, 2021.

[UBS21] Balazs Udvarhelyi, Olivier Bronchain, and François-Xavier Standaert. Security analysis of deterministic re-keying with masking & shuffling: Application to isap. In *COSADE*, Lecture Notes in Computer Science. Springer, 2021.

[UvWBS20] Balazs Udvarhelyi, Antoine van Wassenhove, Olivier Bronchain, and François-Xavier Standaert. On the security of off-the-shelf microcontrollers: Hardware is not enough. In *CARDIS*, volume 12609 of *Lecture Notes in Computer Science*, pages 103–118. Springer, 2020.

Bibliography

[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.

[ABG+21] Melissa Azouaoui, Olivier Bronchain, Vincent Grosso, Kostas Papagiannopoulos, and François-Xavier Standaert. Bitslice masking and improved shuffling: How and when to mix them in software? *IACR Cryptol. ePrint Arch.*, page 951, 2021.

[ADP+20] Melissa Azouaoui, François Durvaux, Romain Poussier, François-Xavier Standaert, Kostas Papagiannopoulos, and Vincent Verneuil. On the worst-case side-channel security of ECC point randomization in embedded devices. In *INDOCRYPT*, volume 12578 of *Lecture Notes in Computer Science*, pages 205–227. 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.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *CHES*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[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.

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ascad with a single side-channel trace. *IACR Cryptology ePrint Archive*, 2021:817, 2021.

[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.

[BGHR14] Nicolas Bruneau, Sylvain Guilley, Annelie Heuser, and Olivier Rioul. Masks will fall off - higher-order optimal distinguishers. In *ASIACRYPT (2)*, volume 8874 of *Lecture Notes in Computer Science*, pages 344–365. 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.

[Bih97] Eli Biham. A fast new DES implementation in software. In *FSE*, volume 1267 of *Lecture Notes in Computer Science*, pages 260–272. Springer, 1997.

[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.

[BKPT19] Ryad Benadjila, Louiza Khati, Emmanuel Prouff, and Adrian Thillard. `https://github.com/ANSSI-FR/SecAESSTM32`. 2019.

[BMPS21] Olivier Bronchain, Charles Momin, Thomas Peters, and François-Xavier Standaert. Improved leakage-resistant authenticated encryption based on hardware AES coprocessors. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):641–676, 2021.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptogr. Eng.*, 10(2):163–188, 2020.

[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.

[BSS21] Olivier Bronchain, Tobias Schneider, and François-Xavier Standaert. Reducing risks through simplicity: high side-channel security for lazy engineers. *J. Cryptogr. Eng.*, 11(1):39–55, 2021.

[CDP16] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Kernel discriminant analysis for information extraction in the presence of masking. In *CARDIS*, volume 10146 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2016.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In *CHES*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.

[CEM18] Thomas De Cnudde, Maik Ender, and Amir Moradi. Hardware masking, revisited. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(2):123–148, 2018.

[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.

[CFOS21] Gaëtan Cassiers, Sebastian Faust, Maximilian Orlt, and François-Xavier Standaert. Towards tight random probing security. In *CRYPTO (3)*, volume 12827 of *Lecture Notes in Computer Science*, pages 185–214. Springer, 2021.

[CGLS20] Gaëtan Cassiers, Benjamin Gregoire, Itamar Levi, and François-Xavier Standaert. Hardware private circuits: From trivial composition to full verification. *IEEE Transactions on Computers*, pages 1–1, 2020.

[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. In *International Cryptographic Module Conference (ICMC 2013)*, page 13.

[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] Gaëtan 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.

[CS21] Gaëtan Cassiers and François-Xavier Standaert. Provably secure hardware masking in the transition- and glitch-robust probing model: Better safe than sorry. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(2):136–158, 2021.

[DBL20] Real-world snapshots vs. theory: Questioning the t-probing security model. *CoRR*, abs/2009.04263, 2020.

[dCGRP19] Eloi de Chérisey, Sylvain Guilley, Olivier Rioul, and Pablo Piantanida. Best information is most successful mutual information and success rate in side-channel analysis.

IACR Trans. Cryptogr. Hardw. Embed. Syst., 2019(2):49–79, 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.

[DMMS21] Sébastien Duval, Pierrick Méaux, Charles Momin, and François-Xavier Standaert. Exploring crypto-physical dark matter and learning with physical rounding towards secure and efficient fresh re-keying. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):373–401, 2021.

[DS16] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In *EUROCRYPT (1)*, volume 9665 of *Lecture Notes in Computer Science*, pages 240–262. Springer, 2016.

[DSP17] François Durvaux, François-Xavier Standaert, and Santos Merino Del Pozo. Towards easy leakage certification: extended version. *J. Cryptogr. Eng.*, 7(2):129–147, 2017.

[DSV14] François Durvaux, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. How to certify the leakage of a chip? In *EUROCRYPT*, volume 8441 of *Lecture Notes in Computer Science*, pages 459–476. Springer, 2014.

[DSV+15] François Durvaux, François-Xavier Standaert, Nicolas Veyrat-Charvillon, Jean-Baptiste Mairy, and Yves Deville. Efficient selection of time samples for higher-order DPA with projection pursuits. In *COSADE*, volume 9064 of *Lecture Notes in Computer Science*, pages 34–50. Springer, 2015.

[DZD+17] A. Adam Ding, Liwei Zhang, François Durvaux, François-Xavier Standaert, and Yunsi Fei. Towards sound and optimal leakage detection procedure. In *CARDIS*, volume

10728 of *Lecture Notes in Computer Science*, pages 105–122. Springer, 2017.

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

[FGP+18] Sebastian Faust, Vincent Grosso, Santos Merino Del Pozo, Clara Paglialonga, and François-Xavier Standaert. Composable masking schemes in the presence of physical defaults & the robust probing model. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(3):89–120, 2018.

[FMPR10] Guillaume Fumaroli, Ange Martinelli, Emmanuel Prouff, and Matthieu Rivain. Affine masking against higher-order side channel analysis. In *Selected Areas in Cryptography*, volume 6544 of *Lecture Notes in Computer Science*, pages 262–280. Springer, 2010.

[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.

[GHRS15] Sylvain Guilley, Annelie Heuser, Olivier Rioul, and François-Xavier Standaert. Template attacks, optimal distinguishers and the perceived information metric. *Cryptarchi*, 2015. `https://perso.uclouvain.be/fstandae/PUBLIS/162.pdf`.

[GJJ+11] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, Pankaj Rohatgi, et al. A testing methodology for side-channel resistance validation. In *NIST non-invasive attack testing workshop*, volume 7, pages 115–136, 2011.

[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.

[GSP13] Vincent Grosso, François-Xavier Standaert, and Emmanuel Prouff. Low entropy masking schemes, revisited. In *CARDIS*, volume 8419 of *Lecture Notes in Computer Science*, pages 33–43. Springer, 2013.

[HHO20] Anh-Tuan Hoang, Neil Hanley, and Máire O'Neill. Plaintext: A missing feature for enhancing the power of deep learning in side-channel analysis? breaking multiple layers of side-channel countermeasures. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):49–85, 2020.

[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.

[Hot31] Harold Hotelling. The generalization of student's ratio. *The Annals of Mathematical Statistics*, 2(3):360–378, 1931.

[HRG14] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. Good is not good enough - deriving optimal distinguishers from communication theory. In *CHES*, volume 8731 of *Lecture Notes in Computer Science*, pages 55–74. Springer, 2014.

[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.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KL14] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography, Second Edition.* CRC Press, 2014.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *CRYPTO*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[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.

[LBS19] Itamar Levi, Davide Bellizia, and François-Xavier Standaert. Reducing a masked implementation's effective security order with setup manipulations and an explanation based on externally-amplified couplings. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):293–317, 2019.

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

[LM19] Liran Lerman and Olivier Markowitch. Efficient profiled attacks on masking schemes. *IEEE Trans. Information Forensics and Security*, 14(6):1445–1454, 2019.

[LP07a] Kerstin Lemke-Rust and Christof Paar. Analyzing side channel leakage of masked implementations with stochastic methods. In *ESORICS*, volume 4734 of *Lecture Notes in Computer Science*, pages 454–468. Springer, 2007.

[LP07b] Kerstin Lemke-Rust and Christof Paar. Gaussian mixture models for higher-order side channel analysis. In *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 14–27. Springer, 2007.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *COSADE*, volume 9064 of *Lecture Notes in Computer Science*, pages 20–33. Springer, 2015.

[LZC+21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):235–274, 2021.

[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.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.

[MOBW13] Luke Mather, Elisabeth Oswald, Joe Bandenburg, and Marcin Wójcik. Does my device leak information? an a priori statistical power analysis of leakage detection tests. In *ASIACRYPT (1)*, volume 8269 of *Lecture Notes in Computer Science*, pages 486–505. Springer, 2013.

[Moo96] Todd K. Moon. The expectation-maximization algorithm. *IEEE Signal Process. Mag.*, 13(6):47–60, 1996.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards.* Springer, 2007.

[MOS11] Stefan Mangard, Elisabeth Oswald, and François-Xavier Standaert. One for all - all for one: unifying standard differential power analysis attacks. *IET Inf. Secur.*, 5(2):100–110, 2011.

[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.

[MRSS18] Amir Moradi, Bastian Richter, Tobias Schneider, and François-Xavier Standaert. Leakage detection with the x2-test. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2018(1):209–237, 2018.

[MS16] Amir Moradi and François-Xavier Standaert. Moments-correlating DPA. In *TIS@CCS*, pages 5–15. ACM, 2016.

[MS21] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected AES implementation on ARM. *IACR Cryptol. ePrint Arch.*, page 592, 2021.

[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.

[MWM21] Thorben Moos, Felix Wegener, and Amir Moradi. DL-LA: deep learning leakage assessment A modern roadmap for SCA evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):552–598, 2021.

[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.

[Pan03] Liam Paninski. Estimation of entropy and mutual information. *Neural Computation*, 15(6):1191–1253, 2003.

[PCP20] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):337–364, 2020.

[PGMP19] Thomas Prest, Dahmun Goudarzi, Ange Martinelli, and Alain Passelègue. Unifying leakage models on a rényi day. In *CRYPTO (1)*, volume 11692 of *Lecture Notes in Computer Science*, pages 683–712. Springer, 2019.

[Pin99] Allan Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[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.

[PRB09] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical analysis of second order differential power analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.

[PSDQ05] Eric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved higher-order side-channel attacks with FPGA experiments. In *CHES*, volume 3659 of *Lecture Notes in Computer Science*, pages 309–323. Springer, 2005.

[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.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (EMA): measures and counter-measures for smart cards. In *E-smart*, volume 2140 of *Lecture Notes in Computer Science*, pages 200–210. Springer, 2001.

[Ren98] Alvin C Rencher. *Multivariate statistical inference and applications*, volume 338. Wiley-Interscience, 1998.

[RPD09] Matthieu Rivain, Emmanuel Prouff, and Julien Doget. Higher-order masking and shuffling for software implementations of block ciphers. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 171–188. Springer, 2009.

[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.

[RSV09] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the AES: why time also matters in DPA. In *CHES*, volume 5747 of *Lecture Notes in Computer Science*, pages 97–111. Springer, 2009.

[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.

[SKS09] François-Xavier Standaert, François Koeune, and Werner Schindler. How to compare profiled side-channel attacks? In *ACNS*, volume 5536 of *Lecture Notes in Computer Science*, pages 485–498, 2009.

[SM16] Tobias Schneider and Amir Moradi. Leakage assessment methodology - extended version. *J. Cryptogr. Eng.*, 6(2):85–99, 2016.

[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.

[Sta19] François-Xavier Standaert. Towards and open approach to secure cryptographic implementations (invited talk), EUROCRYPT 2019. pages xv, `https://www.youtube.com/watch?v=KdhrsuJT1sE`, 2019.

[SVO+10] François-Xavier Standaert, Nicolas Veyrat-Charvillon, Elisabeth Oswald, Benedikt Gierlichs, Marcel Medwed, Markus Kasper, and Stefan Mangard. The world is not enough: Another look on second-order DPA. In *ASIACRYPT*, volume 6477 of *Lecture Notes in Computer Science*, pages 112–129. Springer, 2010.

[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.

[Wag12] Mathias Wagner. 700+ attacks published on smart cards: The need for a systematic counter strategy. In *COSADE*,

volume 7275 of *Lecture Notes in Computer Science*, pages 33–38. Springer, 2012.

[WAGP20] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):147–168, 2020.

[Wel47] B. L. Welch. The generalization of 'student's' problem when several different population variances are involved. *Biometrika*, 34(1-2):28–35, 1947.

[WO19a] Carolyn Whitnall and Elisabeth Oswald. A cautionary note regarding the usage of leakage detection tests in security evaluation. *IACR Cryptol. ePrint Arch.*, 2019:703, 2019.

[WO19b] Carolyn Whitnall and Elisabeth Oswald. A critical analysis of ISO 17825 ('testing methods for the mitigation of non-invasive attack classes against cryptographic modules'). In *ASIACRYPT (3)*, volume 11923 of *Lecture Notes in Computer Science*, pages 256–284. Springer, 2019.

[WOS14] Carolyn Whitnall, Elisabeth Oswald, and François-Xavier Standaert. The myth of generic dpa...and the magic of learning. In *CT-RSA*, volume 8366 of *Lecture Notes in Computer Science*, pages 183–205. Springer, 2014.

[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):389–415, 2020.

[Wy92] LIN Wen-ying. An overview of the performance of four alternatives to hotelling's t square. *Educational Research Journal*, 7:110–114, 1992.

[ZBHV20] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):1–36, 2020.

[ZDF20] Ziyue Zhang, A. Adam Ding, and Yunsi Fei. A fast and accurate guessing entropy estimation algorithm for full-key recovery. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(2):26–48, 2020.

[ZZN+20] Jiajia Zhang, Mengce Zheng, Jiehui Nan, Honggang Hu, and Nenghai Yu. A novel evaluation metric for deep learning-based side channel analysis and its extended application to imbalanced data. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):73–96, 2020.

Part IV

Appendices

Multivariate leakage detection

A

A.1 Investigated covariance matrix

The exact values of the investigated covariance (cf. Figure 3.7) denoted as $\sigma^2_{i,j}$ are defined according to a dependency coefficient Δ and the SNR. The diagonal elements are given by the SNR such that $\sigma^2_{i,i} = 2/\mathsf{SNR}$, and the non-diagonal elements are defined as:

$$\sigma^2_{i,j} = \max(\sigma^2_{i,i} \cdot (1 - \Delta \cdot |i - j|), 0),$$

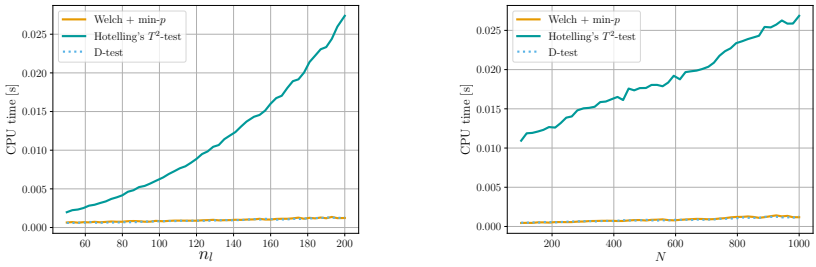
where Δ measures the independence of one random variable $\boldsymbol{X_i}$ (resp., $\boldsymbol{Y_i}$) with the adjacent ones. Since $\boldsymbol{\Sigma}_1$ is diagonal, the coefficient Δ is equal to infinity. For $\boldsymbol{\Sigma}_2$ and $\boldsymbol{\Sigma}_3$, the coefficients are respectively $\Delta = 0.1$ and $\Delta = 0.02$. For example, this results in the covariances $\sigma^2_{0,1} = 0.9$ and $\sigma^2_{0,2} = 0.8$ for $\boldsymbol{\Sigma}_2$ with $\mathsf{SNR} = 1$.

A.2 Computational complexity evaluations

Once the traces have been collected, the obtained measurements need to be processed. Since these methods require different knowledge about the data, the time spend in processing is not the same. In the following, we show the CPU time required for the three leakage detection methods. The code is written in `Python3` and is based on `Numpy1.14` library. The time is recorded for a single core of an `Intel(R) Core(TM) i7-5500U CPU @ 2.40GHz`.

First, we observe that the computational complexity of Welch's t-test with min-p grows linearly with the number of points in the trace n_l, since they are processed independently (cf. Figure A.1a). However, computing and inverting the pooled covariance matrix is roughly quadratic in n_l, as depicted in Figure A.1a. As a result, it rapidly becomes out of reach for large traces (e.g., pre-processed for higher-order detection).

Second, the complexity grows linearly with the number of collected measurements $N_{\boldsymbol{X}}$ and $N_{\boldsymbol{Y}}$ in all the cases. However, the constant is much larger for the Hotelling's T^2-test due to the matrix computation (cf. Figure A.1b).



(a) $N = 1000$ **(b)** $n_l = 200$

Figure A.1: Required CPU time of the leakage detection algorithms.

Finally, we mention that even if the processing complexity of the Hotelling's T^2-test is much larger, it significantly reduces the time spent in measurement (in the case of dense traces), which is generally the longest part of a side-channel security evaluation. So in the case of short and dense traces (i.e., open-source design), the Hotelling's T^2-test remains the methodology of choice.

Machine learning based evaluations

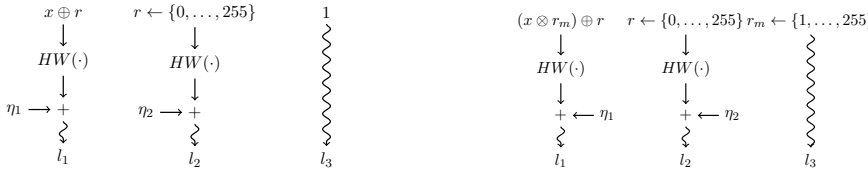
B

As mentioned in the introduction, machine learning / deep learning algorithms are increasingly popular tools to analyze side-channel resistance. In this section, we question whether such tools can improve the attacks in the previous section. Since one of the claimed advantages of machine learning / deep learning algorithms is their potential for automation, we will consider two experimental settings. First, a close to black box setting where the neural network is fed with the same POIs as the attacks in the previous section. Next, a dissection attack where the neural network is used to extract information about specific targets. We analyze the performance of a MLP as a first step and leave the investigation of other machine learning / deep learning tools as a scope for further investigations.

B.1 Methodology

In contrast with gTs that only require one pass on the data to estimate means and covariances, meta-parameters must be tuned for a MLP, which is typically done with (k-fold) cross-validation. Parameters such as the number of layers, the learning rate and the activation function were selected accordingly (with $k = 5$). We allowed a maximum number of 3 hidden layers with up to 500 nodes, a learning rate between 0.01 and 0.2, and *sigmoid*, *relu* and *tanh* activation layers. We selected the best set of parameters with a random grid search performed independently for each of the investigated problems. The number of epochs was fixed to 250. Once the meta-parameters are found, the MLP is trained again with all the training samples. Practically, the open source library `Keras` was used as a front end to `TensorFlow`.[1] The training was performed on 48 available cores. We refer to the recent work [ZBHV20] for more details on such parameters' selection.

[1] `https://keras.io`



(a) Boolean masking. **(b) Affine masking.**

Figure B.1: Simulation settings.

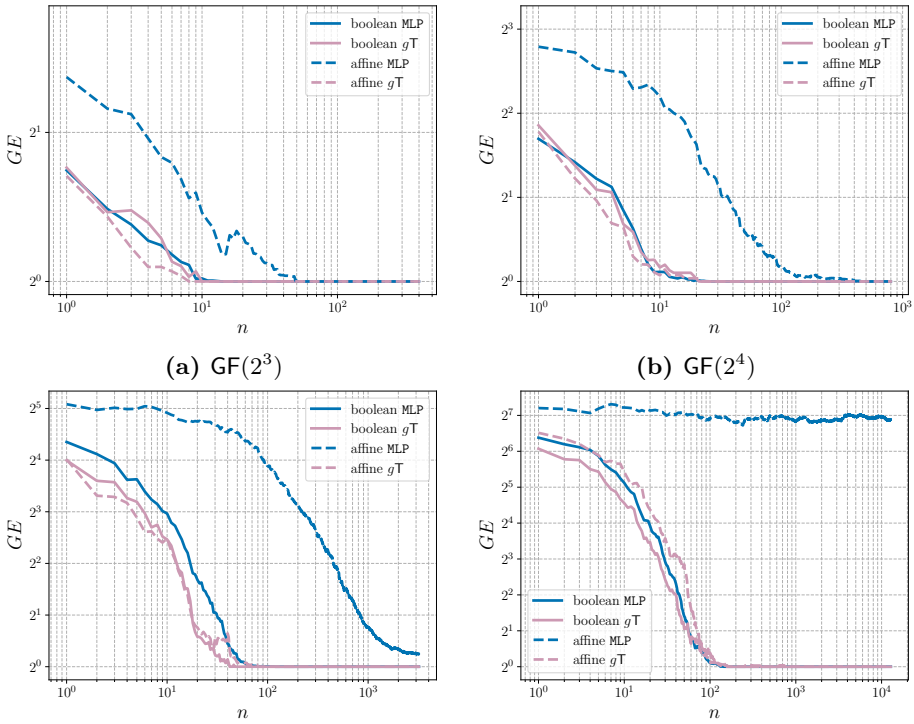
B.2 Close to black-box evaluations

Using the previous methodology, we tried to perform a close to black box analysis of the ANSSI implementation. However, despite feeding an MLP with the same POIs as for the previous attacks, none of the attacks succeeded. To gain intuition about this failure, we therefore ran some simulated experiments. Namely, we compared a gT adversary similar to the previous section to an MLP attack in two scenarios:

4. The first scenario corresponds to a Boolean masking (as illustrated in Figure B.1a). There, the secret value x is XORed with a single random byte r. In order to perform the attack, the gT adversary needs knowledge of x and r associated to the leakage samples $l_\star$. He can then estimate the distributions of l_1 and l_2 individually. The attack is performed by summing over all the possible r's as shown in Equation 6.5. The training of the MLP is performed without the knowledge of the masks and therefore called black box. It only needs the label x corresponding to the $l_\star$'s.

5. The second scenario corresponds to an affine masking with *known* multiplicative mask r_m (Figure B.1b). It is therefore similar to the previous practical experiments where r_m was recovered with a SPA. Despite trivariate, the gT adversary in this setting is essentially the same as the one targeting the previous Boolean masking scheme: she just needs to compute $x \otimes r_m$ in the attack phase. Indeed, if r_m is known the only remaining protection is a Boolean masking with two shares. By contrast, the MLP adversary does not change, raising the question whether she will be able to automatically learn the field multiplication.

Note that in case (1), an additional useless dimension is added so that the MLP can take exactly the same inputs in both scenarios (for the gT, this has no impact given an accurate profiling [LPB+15]). Furthermore, and in order to help interpreting the results, we investigated attacks



(a) GF(2^3) (b) GF(2^4)

(c) GF(2^6) (d) GF(2^8)

Figure B.2: GEs of MLP and gT in front of Boolean masking and affine masking with known multiplicative mask for SNR =10 with n traces (estimated over 100 experiments).

against encodings in Galois fields of sizes ranging from three to eight bits (hoping to observe a gradual degradation of the MLP attack in the second scenario as the field size increases). The number of traces used for the profiling is fixed to $2,000$ per intermediate values, which is a similar profiling effort as in the previous practical experiments. We use the Hamming weight leakage model and the SNR is set to 10 (i.e., a high value since we are interested in the hardness of learning the field multiplication).

The GEs obtained for each of the scenarios and adversaries are shown in Figure B.2. The first observation is that for all the experiments, the gT attacks against the Boolean masking and the ones against the affine masking with known multiplicative masks show similar performances. So as expected, a Boolean masking scheme and an affine masking scheme with known multiplicative mask are equivalent for such an adversary (who can easily perform the multiplication by r_m with perfect knowledge

of r_m).

The second observation is that the efficiency of the MLP attacks compared to the one of gT differs in function of the scenario. In the context of a Boolean encoding, the MLP adversary performs as good as the gT without requiring knowledge of the randomness during the profiling. This makes the MLP an appealing solution when the randomness cannot be controlled by the evaluator. It implies that the MLP is able to efficiently estimate the Gaussian mixture model with a low noise (i.e., a SNR of 10).

By contrast, and contrary to what is observed for a gT adversary, the MLP requires more traces to recover a word of the key for the affine masking scheme than for the Boolean masking, despite the known multiplicative mask. Furthermore, this trend clearly increases with the size of the field in which the multiplications are performed. For example, if the operations are performed in $\mathsf{GF}(2^4)$, the secret can be recovered with 5 times more traces than with a Boolean masking; in $\mathsf{GF}(2^8)$, hundred times more samples do not allow to recover the secret byte. We conclude that the problem of learning a field multiplication based on side-channel leakages is not trivial (despite it is trivial to perform manually).

Overall, our results suggest the attack of the ANSSI implementation with less than 2,000 traces (and similar time complexities and profiling efforts as ours) as an interesting challenge for black box machine learning based evaluations. While such experiments are admittedly no proofs, we believe the general intuition they illustrate is correct and important: even if the field multiplications of an affine masking scheme can eventually be learned in a black box manner, it is likely that it will imply significant profiling overheads while manually exploiting the knowledge of an additive mask is actually trivial.

So these simulated experiments allow an interesting illustration of the pros and cons of machine learning in the context of side-channel security evaluations. On the positive side, they are handy to automate some parts of the attacks such as the learning of a Gaussian mixture with low noise (which is interesting since performed without mask knowledge). On the negative side, they remain limited in a fully black box setting. In this respect, the ANSSI implementation provides a realistic example where the gap between a black box security evaluation and a countermeasures' dissection is actually wide.

Bitslice masking attacks C

C.1 AES Sbox circuit

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

Table C.1: Top 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 C.2: Middle non-linear layer

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

Table C.3: Bottom linear layer

C.2 Clyde Sbox circuit

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

Table C.4: Clyde Sbox

C.3 Moment-based leakage detection

We report in Figure C.1 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.

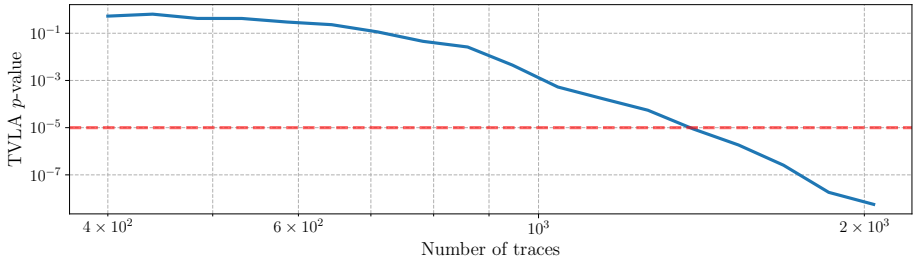
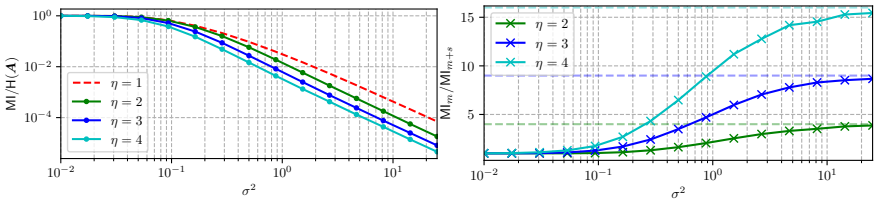


Figure C.1: Multivariate second-order TVLA evaluation for the Cortex-M0 with $d = 3$ shares. The p-value is adapted in function of the number of sample within the traces.

Combination of masking and shuffling

D

D.1 Additional simulations with permutation leakage



(a) $d = 2$, $\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$ **(b)** $d = 2$, $\mathsf{MI}_m(\boldsymbol{A}; \boldsymbol{L})/\mathsf{MI}_{m+s}(\boldsymbol{A}; \boldsymbol{L})$

Figure D.1: Shuffling shares (Algorithm 5) with permutation leakage IT analysis.

D.2 Additional algorithms

Algorithm 10 Generic shuffled masked ISW multiplications with all loops merged.

Input: inputs $\{\boldsymbol{a}_0, \boldsymbol{a}_1, \ldots, \boldsymbol{a}_{\pi-1}\}$ and $\{\boldsymbol{b}_0, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_{\pi-1}\}$, shuffling configuration $(x^{\alpha,\beta,\gamma})$ with $\alpha, \beta, \gamma \in \{s, i, j\}$ and randomness $r_s^{i,j}$ defined as: $\forall s, \forall i, \forall j$, such that $i < j$, $r_s^{i,j} \leftarrow \{0,1\}$ and $r_s^{j,i} = r_s^{i,j}$ and $r_s^{i,i} = 0$.

Output: outputs $\{\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{\pi-1}\}$ such that $\forall s \in \{0, 1, \ldots, \pi - 1\}, c_s = a_s \otimes b_s$.

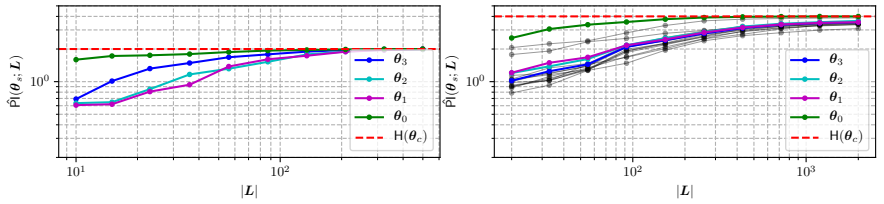
for α, β, γ in $\theta^{\alpha,\beta,\gamma} \xleftarrow{x^{\alpha,\beta,\gamma}} \boldsymbol{\Theta}^{\alpha,\beta,\gamma}$ **do**
    $c_s^i \leftarrow c_s^i \oplus r_s^{i,j} \oplus (a_s^i \otimes b_s^j)$

Algorithm 11 Efficient Masked and Shuffled ISW.

Input: $\boldsymbol{a}$ with $a_s = \sum_{i=0}^{d-1} a_s^i$ and $\boldsymbol{b}$ with $b_s = \sum_{i=0}^{d-1} b_s^i$ and $\pi = |\boldsymbol{a}| = |\boldsymbol{b}|$.
Output: $\boldsymbol{c}$ with $c_s = \sum_{i=0}^{d-1} c_s^i$ and $c_s = a_s \otimes b_s$.

for i in $[0, \ldots, d-1]$ **do**
  for s in $\boldsymbol{\theta}^s \leftarrow \boldsymbol{\Theta}^s$ **do**
    $c_s^i = a_s^i \otimes b_s^i$
for i in $[0, \ldots, d-1]$ **do**
  for j in $[i+1, \ldots, d-1]$ **do**
    for s in $\boldsymbol{\theta}^s \leftarrow \boldsymbol{\Theta}^s$ **do**
      $r_s \leftarrow \{0, 1\}$
      $c_s^i \leftarrow c_s^i \oplus (r_s \oplus (a_s^i \otimes b_s^j))$
    for s in $\boldsymbol{\theta}^s \leftarrow \boldsymbol{\Theta}^s$ **do**
      $c_s^j \leftarrow c_s^j \oplus (r_s \oplus (a_s^j \otimes b_s^i))$

D.3 Perceived information on permutation indexes



(a) $\mathsf{PI}(\boldsymbol{\theta}_c; \boldsymbol{L})$, $\eta = 4$ **(b)** $\mathsf{PI}(\boldsymbol{\theta}_c; \boldsymbol{L})$, $\eta = 16$

Figure D.2: Information on θ_i in function of the # of dimensions exploited.