When Theory Meets Practice: A Framework for Robust Profiling Side-channel Analysis

Stjepan Picek[1], Annelie Heuser[2], Lichao Wu[1], Cesare Alippi[3,4], and Francesco Regazzoni[3,5]

[1] Delft University of Technology, The Netherlands
[2] CNRS, IRISA, Rennes, France
[3] Università della Svizzera Italiana, Lugano, Switzerland
[4] Politecnico di Milano, Milano, Italy
[5] University of Amsterdam, The Netherlands

Abstract. Profiling side-channel attacks are considered the most potent form of side-channel attacks. They consist of two steps. First, the adversary builds a leakage model using a device similar to the target one. This leakage model is then exploited to extract the secret information from the victim's device. These attacks can be seen as a classification problem, where the adversary needs to decide to what class (and consequently, the secret key) the traces collected from the victim's device belong. The research community investigated profiling attacks in-depth, mostly by using an empirical approach. As such, it emerges that a theoretical framework to analyze profiling side-channel attacks comprehensively is still missing.

In this paper, we propose a theory-grounded framework capable of modeling and evaluating profiling side-channel analysis. The framework is based on the expectation estimation problem that has strong theoretical foundations. We quantify the effects of perturbations injected at different points in our framework through the robustness analysis, where the perturbations represent sources of uncertainty associated with measurements, non-optimal classifiers, and countermeasures. Finally, we use our framework to evaluate the performance of different classifiers using publicly available traces.

1 Introduction

Embedded and cyber-physical devices, connected to form the Internet of Things (IoT), are pervading every aspect of our lives. Even though they provide fundamental services, their use of sensitive data and access to critical infrastructure brings new security challenges. It follows that security becomes one of the most important extra-functional requirements that the designer should grant to the devices. Designing secure embedded devices is extremely challenging for two main reasons. First, the limited area and energy budget available in these devices are often insufficient to implement full flagged and robust cryptographic primitives. Second, these devices should be resistant to physical attacks. The

pervasive diffusion of these devices makes them physically available to adversaries willing to exploit the implementation's physical weaknesses to extract the stored secret information (typically, the secret key).

It is necessary to understand the adversary's capabilities to achieve resistance against physical attacks. Side-channel attacks (SCAs), in particular profiling ones, are by far the most studied (and the most powerful) physical attacks since they have been proved to be very effective both in the lab and in real-world applications [1]. In profiling attacks, the adversary first profiles the identical device (or, at least similar) to the one that will be attacked. In the second phase, using this profile and the victim device's traces, the adversary attempts to recover the secret key. Common examples of such profiling attacks are template attack [2, 3] as well as supervised machine learning-based attacks [4–9].

It is well-known that a template attack is the most powerful one from the information-theoretic perspective (given that certain assumptions hold) [3]. Simultaneously, numerous related works experimentally showed how machine learning techniques perform extremely well in many realistic scenarios, see, e.g., [5,9]. However, there is no guarantee that machine learning techniques would behave similarly in different scenarios, even if they share characteristics. Still, one would hope to be able to get answers to more specific questions, e.g.,:

1. What machine learning method is optimal for a given side-channel scenario?
2. What machine learning method should be preferred for multiple side-channel scenarios?
3. Which machine learning method is the most stable one?

Note that we also provide insights into connections among different settings or even countermeasures by answering those questions.

The current state-of-the-art in profiling side-channel analysis progressed tremendously in the last few years. There, deep learning results showed it is possible to break implementations protected even with countermeasures [5, 8]. More recently, the SCA community started investigating the explainability of machine learning-based attacks to produce more powerful attacks and novel countermeasures [10, 11]. However, to the best of our knowledge, no results offer strong theoretical insights into the performance of machine learning-based attacks. Consequently, a general framework, capable of modeling (and thus, fairly comparing) all the profiling attacks, and their characteristics, even when used in different conditions, is still missing.

To address this need, in this paper, we propose a framework based on the analysis of problems approached in probability, which models *all* the profiling attacks, allows to analyze their behavior and their performance and gives a rigorous insight into their robustness. We leverage on problems investigated in probability since, similarly to that class of problems, also in the case of profiling side-channel attacks, we are not searching *the* solution, but a valid answer according to a defined probabilistic figure of merit (which in our case can be the accuracy of the classifier or the guessing entropy). Furthermore, as in the case of problems approached in probability, we are interested in knowing to what ex-

tent our result has to be considered correct and what maximum error we could obtain.

Our framework is designed to analyze the behavior of profiling SCAs and give insights into the robustness of such methods. By robustness, we consider the system's ability to tolerate perturbations (random changes affecting the system). Thus, robustness (or a lack of it) can be considered as a consequence of adding perturbations to the system. Although we use the robustness analysis as a tool to obtain insights into the general behavior of profiling attacks, perturbations are also commonly occurring in SCA. Different sources of perturbations occur in profiling SCA due to the presence of:

- Environment noise.
- Countermeasures.
- The differences between the profiling device and the device under attack (portability).

While the first source of perturbation is immediate, the other two deserve further explanation. Consider a system with a countermeasure. The profiling attack does not "know" what the countermeasure is but can only "see" its consequence. This discrepancy can be successfully modeled as a random variable whose realizations are associated with perturbations affecting the system's functionality. Finally, while it is common to use the same device for both training and testing, in reality, there are two devices (the first one to train a model, and the second one to attack). This simplification is reasonable but will introduce errors where the attack's performance will be lower than the one measured in experiments with only one device [12]. Additionally, the difference in the measurements in practice can also arise, for instance, from different probes positions when using electromagnetic (EM) SCA.

By considering the robustness paradigm, we evaluate a setting that approximates the realistic one, where perturbations must occur, and uncertainty is present. That paradigm is also in the core of the well-known Provably Approximate Correct (PAC) learning [13], where PAC learning theory formalizes the way computation is carried out within an uncertainty affected environment.

The main contributions of this paper are:

- We propose a framework capable of modeling and evaluating profiling side-channel attacks to provide strong theoretical foundations for the framework.
- We consider the robustness of profiling attacks in 1) the presence of countermeasures and 2) environment settings like feature selection, dataset size, and hyperparameter tuning.
- We consider the robustness of profiling attacks for different figures of merit: accuracy, success rate, number of attack traces, and guessing entropy.

The rest of this paper is organized as follows. In Section 2, we discuss related works and directions commonly investigated in the profiling side-channel analysis. Section 3 first discusses the threat model, and then, it presents an intuitive description of our framework. Afterward, we formally define it through

the expectation estimation problem. Next, Section 4 presents details about our experimental setting: profiling methods, datasets, figures of merit, and framework parameters we investigate. In Section 5, we validate our framework by using a technique called stylized facts that compares the behavior of simulations and real-world data. Additionally, we present an experimental evaluation for publicly available datasets. Section 6 presents a discussion about general findings from the experiments and advantages/drawbacks of using our framework. Finally, Section 7 concludes the paper.

2 Related Work

In 1996, Kocher demonstrated the possibility to recover secret data by introducing a method for exploiting the leakages from the device under attack [14]. This is possible because implementations of cryptographic algorithms leak relevant information about the data processed through physical side-channels like timing [14], power consumption [15], EM emanation [16], or sound [17]. The side-channel attacks are usually divided into two groups: direct attack (see, e.g., Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [15]) and profiling attacks (see, e.g., Template Attack (TA) [3], stochastic models [18], or a number of machine learning-based techniques as discussed next).

Profiling attacks are carried out in two stages, profiling and attacking. In the profiling stage, the adversary has the full control of cryptographic hardware and can estimate the leaked information's probability distribution. In the second stage, the estimated distribution is used to extract the secret information from a victim device. Profiling attacks are the most powerful form of side-channel attacks. Because of this, they define the worst-case security assumptions. TA is the best (optimal) technique from an information-theoretic perspective if the attacker has an unbounded number of traces, and the noise follows the Gaussian distribution [19, 20]. After the template attack, the stochastic models technique emerged, which uses linear regression in the profiling phase [18]. In years to follow, researchers recognized certain shortcomings of template attacks and tried to modify them in order to deal better with the complexity and portability issues. One example of such an approach is the pooled template attack, where one pooled covariance matrix is used to cope with statistical difficulties [2].

Alongside such techniques, the SCA community realized that a similar approach to profiling is used in other domains in the form of supervised machine learning. Consequently, the researchers started experimenting with different machine learning methods and evaluating their effectiveness in the SCA context. A survey of available works reveals several papers that discuss different machine learning methods, targets, and experimental settings. When considering attacks on AES (as we do in this paper) and supervised machine learning methods, one can find a number of papers, see, e.g., [4, 6, 20–27]. More recently, deep learning techniques have come into the focus of the SCA community. The first results confirmed that expectations where most of the attention went to convolutional neural networks [5, 7–9, 28].

When trying to find a common denominator for those works, the most obvious one is that they report machine learning being able to reach high attack performance and often outperforming template attack. Additionally, deep learning can commonly break the implementations protected with countermeasures (at least those available in the publicly available datasets). Still, all of these works have "only" experimental results with sporadic attempts in explaining why such results are obtained. Besides works that consider how to improve the attack performance, more recently, there have been several works considering the topics of explainability and interpretability of deep learning in side-channel analysis [10, 11, 29].

Other aspects of the profiling side-channel attacks have been studied already in the past. The problem of comparing profiling side-channel attacks has been tackled by Standaert et al. [30] They proposed to use the information theory-based methodology to investigate the performance of the different phases of profiling side-channel attacks on exemplary simulated leakages [31]. Their results show that the quality of the profiling phase can be captured by an information-theoretic metric, while the key recovery phase is better measured with a security metric. The analysis of profiling side-channel attacks can also benefit from simplification and optimization. Among such works, a relevant one is from Bronchain et al., where the authors propose a quantitative tool for leakage certification [32]. Quantitative analysis is carried out by bounding the unknown mutual information metric on perceived information and on the hypothetical information. Durvaux et al. addressed the problem of bias in the security evaluation caused by estimation and assumption errors using sound statistical techniques to quantify the leakage of a device and guarantee that the amount of information extracted is close to the maximum achievable with a perfect model [33].

3 General Framework

In this section, we present a framework that can model all profiling side-channel attacks. First, we introduce the framework in an intuitive way, showing motivation and goals. Then, we introduce the background definitions and the numerical problems on which our framework is based. Finally, we formally present the framework for modeling profiling side-channel attacks.

3.1 Threat Model

We investigate a typical profiling side-channel setting. We consider this to be a de-facto standard as a number of certification laboratories are evaluating hundreds of security-critical products with this model daily.

In this setting, the adversary has access to a clone device running the target cryptographic algorithm. The clone device can be queried with a known key and plaintext, while the corresponding leakage trace is stored. Ideally, the adversary can perform an infinite number of queries and can construct the corresponding database of side-channel leakage traces to characterize a precise profiling model.

After the characterization is done, the adversary queries the victim device with known plaintext to obtain the secret key. The key recovery is carried out by comparing the side-channel leakage traces from the victim device with the profiling model previously characterized.

Since the focus of this paper is power and electromagnetic attacks, in the rest of the paper, when discussing profiling attacks, we consider profiling attacks that use power or electromagnetic radiation as side-channel. Furthermore, in this paper, we consider only those scenarios where the paradigm is supervised learning, and the task is classification, i.e., to learn a mapping f between a set of input variables X an output variable Y ($f : X \rightarrow Y$).

3.2 Intuitive Description of the Framework

The previous works on profiling side-channel attacks discussed in Section 2 provide a detailed overview of the attacks and methodology used. Still, they do not abstract the specificity of the attack to a higher-level framework. As a result, a complete analysis of the attack characteristics is empirical, and so is the direct comparison of different profiling techniques.

Figure 1 depicts the generic framework we propose to model profiling side-channel attacks. Recall, during such attacks, the adversary attempts to recover the secret (typically, the key) of a cryptographic device in two phases, commonly known as the profiling and attacking phase. The profiling of the device D (the physical realization of a cryptographic primitive), depicted on the right part of the figure, consists of collecting several leakage traces $x_i, 1 \leq i \leq n$ where n is the number of traces taken corresponding to the encryption of a certain number of plaintexts PT and a number of known keys K. For each trace x_i, we obtain T time samples (also known as features or attributes): $x_i = x_{i,1}, \ldots, x_{i,T}$. During the profiling phase, for each of the traces x_i, we additionally obtain the label of the trace y_i, which denotes the actual value the trace has (where the connection between the label and the key/plaintext includes the sensitive operation like the S-box and an appropriate leakage model). The leaked traces are typically altered by a random perturbation $\delta_1\theta$ that can be caused by measurement or algorithmic noise but also by the operation of a side-channel countermeasure. The leakage trace and the corresponding plaintext are used to derive an estimate of the secret key $\hat{K}$. These estimates of the secret key, along with the traces, are used to train a classifier C, depicted in the left part of Figure 1. The specific classifier can be arbitrarily selected from the corpus of all possible classifiers suitable for side-channel recovery. Each classifier is trained on the training set, in the sense that different training sets of the same size will generate different classifier (profiling) models. To model this phenomenon, we consider the given classifier's output influenced by a specific perturbation $\delta_2\theta$, which implies we are assuming that the intensity (variance) of two perturbations can be the same (i.e., different points are characterized by the same signal-to-noise ratio (SNR) or values affected by perturbations insist on the same bounded interval). Without loss of generality, we consider $\delta_1\theta$ and $\delta_2\theta$ to be equal to $\delta\theta$ since here, we do not differentiate between the perturbations coming from one or the other source.

The estimated values of the labels $\hat{y}_i$ are compared to the actual ones and used to estimate the classifier's robustness.

The robustness problem we consider here can be solved by considering the robustness of randomized algorithms [34]. In the particular randomized algorithm setting, we are interested in quantifying the robustness of an algorithm utilizing a suitable figure of merit. In our setting, we aim to quantify the robustness of a profiling side-channel attack (first seen as a supervised machine learning problem) to the perturbations caused by the measurement noise or countermeasures but also the intrinsic noise of a specific classifier. The framework we consider is general, which means it supports any profiled/supervised method and model, as well as any figure of merit. To validate our framework, in Section 5, we select to work with a number of common SCA classifiers and figures of merit.

Our framework proposes one theoretical interpretation for an analysis that is currently mostly done empirically. Consequently, we can achieve two goals:

1. The connection with well-understood problems (expectation estimation problem and robustness problem) allows us to model each profiling side-channel attack and thus compare, in an objective and rigorous way, the performance of different classifiers in a specific scenario.
2. A quantitative estimate of the confidence of our results. Ultimately, we will be able to answer the questions like "Which classifier family behaves better in some specific scenario?" and "How to compare different profiling side-channel attacks?".

3.3 Expectation Estimation Problem and Robustness

In this section, we aim to answer two questions which will be used in subsequent derivations and provide some common nomenclature. First, how to estimate the expected value of a function by identifying the minimum number of samples granting to build an approximation with an arbitrary level of accuracy ϵ and confidence δ (Section 3.3). Second, how to estimate the robustness of a system once affected by perturbations (see Section 3.3). To achieve this, we introduce the basic theory behind the expectation estimation problem and robustness analysis.

Lebesgue Measurability The first requirement that we have to fulfill is that the expected value we want to compute exists. In other words, we require that the integrals we need to compute are well defined, namely that the functions we need to compute are measurable. The most common type of integrals is the integrals of Lebesgue, that are well defined on functions that are Lebesgue measurable.

A generic function $u(\psi), \psi \in \Psi \subseteq \mathbb{R}^l$ is Lebesgue measurable with respect to Ψ when its generic step-function approximation S_N obtained by partitioning Ψ in N *arbitrary* domains grants that

$$\lim_{N \to \infty} S_N = u(\psi)$$

on set $\Psi - \Omega$, $\Omega \subseteq \mathbb{R}^l$ being a null measure set [35].

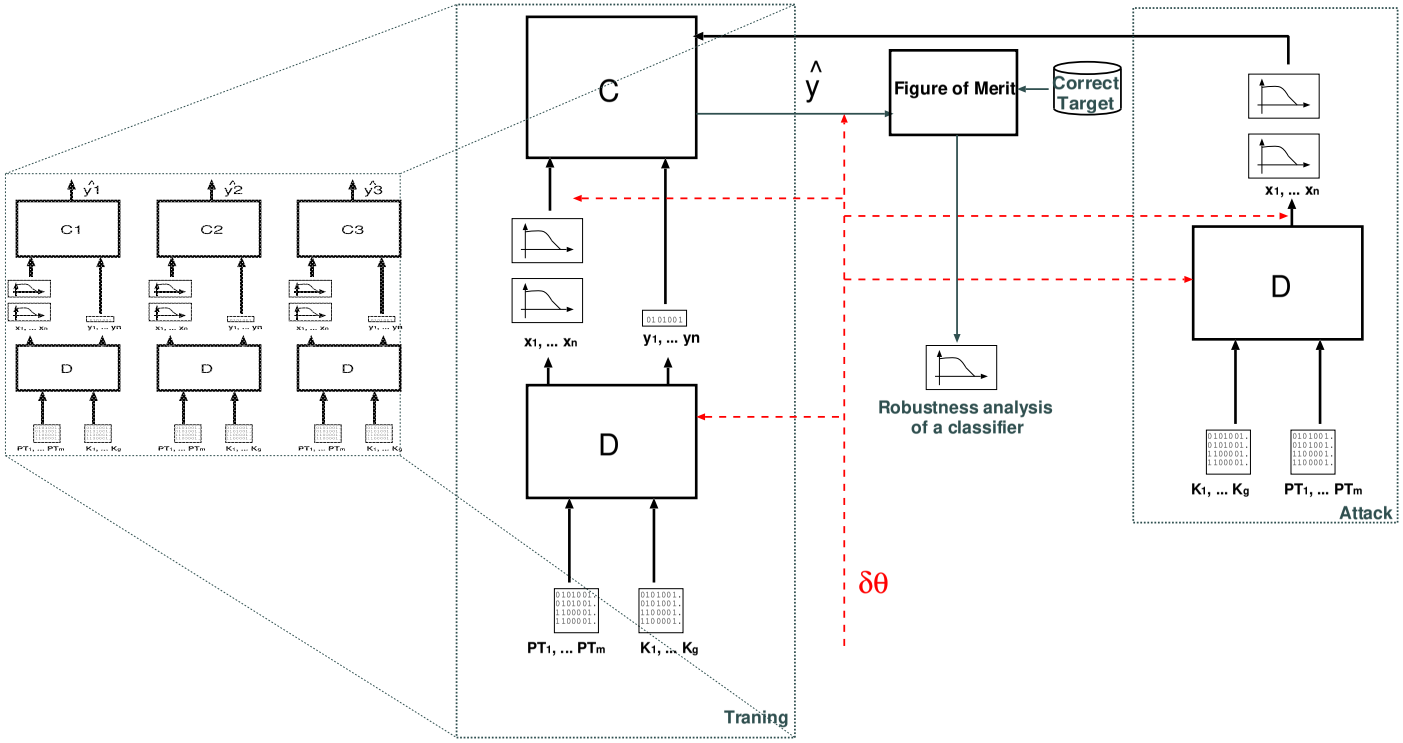


Fig. 1: The framework for the profiling side-channel analysis. The red line denotes the locations of possible perturbations. There are two phases of the attack. Profiling phase (left part of the figure in bold) and attack phase (right part of the figure in bold). Since the framework is generic, it supports any type of profiling attack, as denoted by several different classifiers on the far left side. This framework supports any: 1) profiling attack, 2) leakage model, 3) dataset (considering, e.g., side-channel information, level of noise, countermeasures used, dataset size).

The assumptions on Lebesgue measurably are extremely soft. Basically, no engineering-related mathematical computations are Lebesgue non-measurable: every time we integrate, we are implicitly assuming that the function we are integrating is Lebesgue measurably. The functions we are considering in our paper are also Lebesgue measurable, thus the integrals we compute in the rest of the paper are well-defined.

The Chernoff Bound The framework for expectation estimation we use is built starting from the Chernoff bound. In a nutshell, the Chernoff bound is a way to estimate the probability distribution of an unknown variable. The main advantage of the Chernoff bound is that it has extremely mild and quite generic assumptions that are fulfilled by an extremely large number of problems. Informally, the requirement of the Chernoff bound we use here are simply that the variable we are considering is bounded.

More precisely, the Chernoff bound allows determining the number of samples needed to estimate a probability with arbitrary accuracy in the estimate approximation and confidence in the made statement. The Chernoff bound can also be used to estimate the expected value of random variable according to estimation accuracy and confidence levels set by the designer [36]. We now define the Chernoff bound formally. The Chernoff bound for a generic probability density function and continuous variable ψ can be derived from the Hoeffding inequality for the empirical mean [37].

Let $x_1, \cdots x_n$ be a sequence of independent random variables so that each x_i is almost surely bounded by the interval $[a_i, b_i]$, i.e., $\Pr(x_i \in [a_i, b_i]) = 1$. Then, defining the empirical mean $\hat{E}_n = \frac{1}{n}\sum_{i=1}^{n} x_i$, we have that for any ϵ value the Hoeffding inequality for the empirical mean:

$$\Pr\left(|\hat{E}_n - E[\hat{E}_n]| \geq \epsilon\right) \leq 2e^{\frac{-2\epsilon^2 n^2}{\sum_{i=1}^{n}(b_i - a_i)^2}} \tag{1}$$

holds where E is the expectation operator.

Eq. (1) can be rewritten as:

$$\Pr\left(|\hat{E}_n - E[\hat{E}_n]| < \epsilon\right) > 1 - 2e^{\frac{-2\epsilon^2 n^2}{\sum_{i=1}^{n}(b_i - a_i)^2}}. \tag{2}$$

If $\hat{E}_n$ is the estimate $\hat{p}_n(\gamma)$ of a probability, e.g., $(p(\gamma) = \Pr(u(\psi) \leq \gamma)$ for a given positive scalar γ and a loss function $u(\psi))$, we have that for a generic random variable ψ_i the indicator function:

$$x_i = I\left(u(\psi_i) \leq \gamma\right) = \begin{cases} 1 \text{ if } u(\psi_i) \leq \gamma \\ 0 \text{ if } u(\psi_i) > \gamma \end{cases}$$

assumes values in $\{0, 1\}$. As a consequence, $a_i = 0, b_i = 1$ and Eq. (2) becomes

$$\Pr\left(|\hat{E}_n - E[\hat{E}_n]| < \epsilon\right) > 1 - 2e^{-2n\epsilon^2}.$$

Since, $\hat{p}_n(\gamma) = \hat{E}_n$ and $E[\hat{p}_n(\gamma)] = p(\gamma)$ we derive

$$\Pr\left(|\hat{p}_n(\gamma) - p(\gamma)| < \epsilon\right) > 1 - 2e^{-2n\epsilon^2}. \tag{3}$$

Finally, we derive the Chernoff bound by requesting $\delta \leq 2e^{-2n\epsilon^2}$:

$$n \geq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta}. \tag{4}$$

The Chernoff bound states that if we sample from the domain of random variable ψ according to its probability density function, then the Eq. (4) holds with confidence δ. In other words, we can build an approximation $\hat{p}_n$ of unknown probability $p(\gamma)$ with accuracy ϵ, and the statement will hold with confidence δ.

The Expectation Estimation Problem The expectation estimation problem we consider is based on the Chernoff bound. In its generic formulation, the expectation estimation problem consists of identifying the minimum number of samples needed to achieve an arbitrary level of accuracy and confidence in approximating the expected value of a given function $u(\psi)$.

Let $u(\psi) \in [0, 1]$ be a Lebesgue measurable function over $\Psi \subseteq \mathbb{R}^l$ and f_ψ be the probability density function of a random variable ψ defined over Ψ. The expectation estimation requires evaluation of the expected mean:

$$E[u(\psi)] = \int_\Psi u(\psi) f_\psi(\psi) d\psi. \tag{5}$$

Since the evaluation of the expected mean defined in Eq. (5) is generally computationally hard problem for a generic function, an approximation is built starting from n i.i.d. samples $\psi_1, \cdots, \psi_i, \cdots, \psi_n$ drawn from ψ according to f_ψ. We call

$$\hat{E}_n(u(\psi)) = \frac{1}{n} \sum_{i=1}^{n} u(\psi_i) \tag{6}$$

the empirical mean. It should be commented that $\hat{E}_n(u(\psi))$ is a random variable depending on the particular realization of the n samples. The Chernoff bound can be used to build an accurate approximation of Eq. (5) through Eq. (6) at accuracy level ϵ and confidence δ.

$\hat{E}_n(u(\psi))$ is the estimate of the figure of merit. By assuming that the condition $u(\psi) \in [0, 1]$ we immediately derive the bound on n thanks to the Hoeffding's inequality. In general, it is enough to require $u(\psi_i)$ to be bounded, e.g., to the same $a_i = a, b_i = b, i = 1, \ldots, n$. If that is the case, the bound on the number of samples becomes:

$$n \geq \frac{(b-a)^2}{2\epsilon^2} \ln \frac{2}{\delta}. \tag{7}$$

Note, there is a connection between the expectation estimation problem we consider here and the probability estimation problem commonly considered in SCA. In fact, the expectation estimation problem (including the Chernoff bounds) are computed starting from the expected values, ultimately relying on values obtained through the probability estimation problem.

Robustness Robustness of a system refers to the ability to tolerate perturbations that might affect its structural parameters and, in turn, its performance, measured utilizing a given figure of merit. In practice, what is desirable is that the system continues to work even in the presence of a perturbation, and what is to be estimated is the capability of correctly operating when perturbations occur. In our case, the system is the profiling attack, and we want to evaluate the robustness of such attacks for all possible effects that can affect the success of an attack (measurement noise, portability of the classifier, countermeasures). Ultimately, we want to understand under which conditions the attack is still possible.

More formally, a system/function $g(\theta, x)$ is robust with respect to perturbations $\delta\theta \in \Delta \in R^l$ at level $\gamma \in \mathbb{R}^+$ when, given a discrepancy function $u\left(g(\theta, x), g(\theta, \delta\theta, x)\right) \in \mathbb{U} \subset \mathbb{R}$ the system experiences a degradation in performance within γ:

$$u(\delta\theta) = u\left(g(\theta, x), g(\theta, \delta\theta, x)\right) \leq \gamma, \quad \forall \delta\theta \in \Delta, \forall x \in \tilde{X}. \tag{8}$$

In the rest of the paper, we assume that the level of perturbations can become arbitrarily large so that no small perturbation theories are viable. $u(\delta\theta)$ represents the perturbation impact on the behavior of the system, as observed through the figure of merit. Note that $u(\delta\theta)$ does not have an explicit function in the inputs in the sense that if inputs are in there, they are finite in number and fixed and belong to the set $\tilde{X}$, which is the discrete set containing a finite number of input instances.

In this setting, we need to determine the smallest γ satisfying the previous expression. Since this can be computationally intractable, we move to a probabilistic setting. There, a computation is robust at level γ with probability $1 - \eta$ for the perturbation space Δ when γ is the smallest value such that $\Pr(u(\delta\theta) \leq \gamma) \geq 1 - \eta,\ \forall \delta\theta \in \Delta$. Here, η is a small positive value in $[0, 1]$ and $1 - \eta$ is the confidence level.

Once defined $p(\gamma)$ to be the probability that $u(\delta\theta) \leq \gamma$ for an arbitrary but given γ value:

$$p(\gamma) = \Pr(u(\delta\theta) \leq \gamma)\ for\ each\ \delta\theta \in \Delta. \tag{9}$$

Eq. (9) can be estimated with Chernoff and the minimum γ identified through set $\Gamma = \{\gamma_1, \cdots, \gamma_k\}$).

3.4 The Profiling SCA Framework

We now express profiling side-channel attacks as the expectation estimation problem. The starting point is to map the steps of profiling analysis to the framework depicted in Figure 1. The first phase of profiling side-channel attacks is the training phase, which is represented by the bold part of Figure 1. The training phase begins with the collection of the traces corresponding with the encryption of several plaintext and keys.

Formally, during the encryption, the secret key k^* is processed with t plaintexts or ciphertexts of the cryptographic algorithm, while the attacker collects

a set of traces x. In the case of AES, typically k^* and t are processed in bytes, which reduces the attack complexity. The mapping y maps the plaintext or the ciphertext $t \in \mathcal{T}$ and the key $k^* \in \mathcal{K}$ to a value that is assumed to relate to the deterministic part of the measured leakage x. We denote the output of y as the label, which is coherent with the terminology used in the machine learning community. For profiling, there are two common models to define $y(t, k^*)$ so to calculate the labels of the measurement traces:

- *intermediate value (ID) model*: in this leakage model, the attacker considers an intermediate value of the cipher or the distance between two consecutive values processed. When considering the AES cipher, this leakage model results in 256 labels.
- *Hamming weight (HW) model*: in this leakage model, the attacker assumes the HW of the intermediate value model. When considering the AES cipher, this leakage model results in 9 labels.[6]

Considering the intermediate value, the profiling model may be more accurate but requires more resources. In particular, to gain stable estimations for each possible value, an attacker needs a sufficient number of measurements per value. Additionally, as the attacker needs to iterate through all values in the profiling phase as well as for each measurement in the attacking phase, the computational complexity may become high - especially when targeting ciphers operating on more than 8-bit.

The HW model's preference is related to the underlying device (e.g., for some devices, the power consumption is assumed to be roughly proportional to the number of bit transitions) and the lower complexity. In our analysis, we consider both leakage models for all datasets and profiling methods.

In the attack phase, the goal is to make predictions about the occurring labels

$$y(t_{a_1}, k_a^*), \ldots, y(t_{a_N}, k_a^*),$$

where k_a^* is the secret unknown key on the device under the attack.

4 Experimental Setting

This section discusses the datasets, machine learning classifiers, and the framework's experimental settings we consider.

4.1 Datasets

In our experiments, we consider several publicly available datasets representing a typical sample of commonly encountered scenarios and one simulated traces dataset (used for the framework validation).

[6] One more common leakage model would be the Hamming distance leakage model, which would also result in 9 labels.

DPAcontest v4 Dataset - DPAv4 This dataset provides traces (measurements) of a masked AES software implementation [38]. As the mask is known, one can easily turn it into an unprotected scenario. The most leaking operation is the processing of the S-box operation, where we attack the first round:

$$Y(k^*) = \texttt{Sbox}[P_1 \oplus k^*] \oplus \underbrace{M}_{\text{known mask}}, \tag{10}$$

where P_1 is the first plaintext byte. The SNR has a maximum value of 5.8577. The dataset has 100 000 traces where each trace has 3 000 features. This dataset is available at `http://www.dpacontest.org/v4/`.

AES_HD Dataset - AES_HD This dataset targets an unprotected implementation of AES-128 implemented on Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board. Side-channel traces were measured using a high sensitivity near-field EM probe, which was placed over a decoupling capacitor on the power line. We use the Hamming distance (HD) leakage model, and attack the last round of an unprotected hardware implementation (the register writing in the last round) [38]:

$$Y(k^*) = HW(\underbrace{\texttt{Sbox}^{-1}[C_{12} \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_8}_{\text{ciphertext byte}}), \tag{11}$$

where C_{12} and C_8 are two ciphertext bytes, and the relation between them is given through the inverse ShiftRows operation of AES. The traces have model-based SNR of a maximum value of 0.0096. In total, there are 50 000 traces corresponding to 50 000 randomly generated plaintexts, each trace with 1 250 features. This dataset is available at `https://github.com/AESHD/AES_HD_Dataset`.

Random Delay Dataset - AES_RD As our third use case, we use AES software implementation on an 8-bit Atmel AVR microcontroller with implemented random delay countermeasure as described by Coron and Kyzhvatov in [39]. We attack the first AES key byte and target the first S-box operation. For this dataset, the SNR has a maximum value of 0.0556. The dataset consists of 50 000 traces of 3 500 features each. This dataset is available at `https://github.com/ikizhvatov/randomdelays-traces`.

ASCAD Dataset The target platform is an 8-bit AVR microcontroller (ATmega8515) running a masked AES-128 implementation, and traces are made using electromagnetic emanation [40]. There are two versions of the ASCAD dataset: one with fixed key with 50 000 traces for profiling/training, and 10 000 for testing. The second version has random keys, and the dataset consists of 200 000 traces for profiling and 100 000 for testing. For both versions, we attack the key byte 3, which is the first masked byte. For the version with the fixed key, we use a pre-selected window of 700 features, while for the version

with random keys, the window size equals 1 400 features. Note that the leakage model does not leak information directly as it is first-order protected, and we, therefore, do not state a model-based SNR. This dataset is available at `https://github.com/ANSSI-FR/ASCAD`.

Simulated Dataset The circuit we simulated to obtain the simulated traces is a reduced portion of the AES algorithm, composed by a key addition followed by an S-box lookup. The obtained data are then stored in a register. The flow used to generate the simulated traces is implemented using state-of-the-art commercial electronic design automation commodities and is derived from the simulation flow presented by Regazzoni et al. [41]. The test circuit is designed using HDL language, synthesized with a synthesis tool (Synopsys design compiler), and placed and routed with an automated tool (Cadence Encounter). The final circuit, together with the parasitic extracted using the extractor build in the place and route tool, are simulated at SPICE level using Synopsys Nanosim, where the simulation resolution has been set to $1ps$, thus producing, for each input-output pair, a trace of 5 000 data points. The target technological library used in the process is the Nangate 45 nm library. At the end of the simulation process, this dataset contains 256 simulated traces of execution, one for each S-box output, and, being obtained from simulation, free from environmental or measurement noise.

4.2 Figures of Merit

We consider two standard metrics when conducting SCA: success rate and guessing entropy. Additionally, we consider one common machine learning metric: accuracy.

Most of the time, in side-channel analysis, an adversary is not only interested in predicting the labels $y(\cdot, k_a^*)$ in the attacking phase for which accuracy is a good metric, but aims at revealing the secret key k_a^*. For this, common measures are the success rate (SR) and the guessing entropy (GE) of a side-channel attack [31]. In particular, let us assume, given Q amount of traces in the attacking phase, an attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability with $|\mathcal{K}|$ being the size of the keyspace. So, g_1 is the most likely and $g_{|\mathcal{K}|}$ the least likely key candidate.

Success rate. It is defined as the average empirical probability that g_1 is equal to the secret key k_a^*.

Guessing entropy. It is the average position of k_a^* in $\mathbf{g}$.

Accuracy. It is defined as the ratio of the correctly classified examples and the total number of examples.

We report two types of results for GE. First, we report the number of traces needed to reach guessing entropy $GE < 10$. This metric is interesting as it allows us to consider the required number of traces. What is more, we still obtain

meaningful results for scenarios where we cannot break the target. Second, we report the guessing entropy value. We calculate the number of traces needed to reach $GE = 0$ and use that number of traces in all subsequent experiments (for all intensity levels). Note that $SR = 0.9$ means that 9 out of 10 results (for a certain number of attack traces) shows that the best key guess is the correct key.

4.3 Profiling methods – Classifiers

It is not a trivial task to select the best classifier for the given problem. Some classifiers can still be regarded as a usual choice when a highly accurate classification is sought [42]. To provide relevant experiments, we select several classifiers that are a common choice in SCA, as discussed in Section 2.

Template Attack The template attack (TA) relies on the Bayes theorem and considers the features as dependent [3]. In the state-of-the-art, template attack relies mostly on a normal distribution. Accordingly, a template attack assumes that each $P(\boldsymbol{X} = \boldsymbol{x}|Y = y)$ follows a (multivariate) Gaussian distribution parameterized by its mean and covariance matrix for each class Y. The authors of [2] propose to use only one pooled covariance matrix averaged over all classes Y to cope with statistical difficulties and thus lower efficiency. In our experiments, we use the pooled template attack.

Naive Bayes The Naive Bayes (NB) classifier is based on the Bayesian rule but is labeled "Naive" as it works under a simplifying assumption that the predictor features are mutually independent among the features, given the class value [43]. The existence of highly-correlated features in a dataset can influence the learning process and reduce the number of successful predictions. NB assumes a normal distribution for predictor features and outputs posterior probabilities as a result of the classification procedure [43].

Radial Kernel Support Vector Machines Radial Kernel Support Vector Machines (denoted SVM in this paper) is a kernel-based machine learning family of methods that are used to classify both linearly separable and linearly inseparable data accurately. The idea for linearly inseparable data is to transform them into a higher dimensional space using a kernel function, wherein the data can usually be classified with higher accuracy. The scikit-learn implementation we use considers libsvm's C-SVC classifier that implements SMO-type algorithm [44]. The multi-class support is handled according to a one-vs-one scheme.

Random Forest Random Forest (RF) is a well-known ensemble decision tree learner [45]. Decision trees choose their splitting attributes from a random subset of k attributes at each internal node. The best split is taken among these randomly chosen attributes.

Multilayer Perceptron The multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP consists of multiple layers (at least three) of nodes in a directed graph, where each layer is fully connected to the next one, and training of the network is done with the backpropagation algorithm [46].

Convolutional Neural Networks Convolutional neural networks (CNNs) commonly consist of three types of layers: convolutional layers, pooling layers, and fully-connected layers. The convolution layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. Pooling decrease the number of extracted features by performing a down-sampling operation along the spatial dimensions. The fully-connected layer (the same as in MLP) computes either the hidden activations or the class scores.

4.4 Points of Interest

For NB, SVM, RF, MLP, and TA, we use the 50 most important features, as commonly done in related works. To select those features, we use the Pearson correlation coefficient [47]:

$$Pearson(x, y) = \frac{\sum_{i=1}^{N}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}. \tag{12}$$

We also use MLP and CNN, where we do not conduct any feature selection but instead use the full set of available features (raw traces or window of traces).

Hyperparameter Tuning We conduct a tuning phase to select hyperparameters for which classifiers perform well over considered datasets. We emphasize that the tuned parameters represent a reasonable choice that exhibits good behavior but should not be considered the best possible ones. A more detailed tuning could result in a somewhat improved performance if concentrating on any specific scenario. Nevertheless, since we consider scenarios where we introduce perturbations in the testing phase, there is no guarantee that good initial hyperparameters would still be suitable for the added noise traces. For TA and NB, there are no hyperparameters to tune. For SVM, there are two significant tuning parameters: the cost of the margin C and the kernel parameter γ. We conduct a grid search for $C = [0.001, 0.01, 0.1, 1]$ and $\gamma = [0.001, 0.01, 0.1, 1]$. In the end, we select $C = 1$ and $\gamma = 1$. For RF, we experiment with a different number of trees $I = [10, 50, 100, 200, 500, 1\,000]$ and select 200 traces for our architecture. For MLP with feature selection, we investigate *tanh* and *ReLU* activation functions, *sgd* and *adam* optimizers, and architectures of shape $(50, 25, 10, 50), (50, 30, 50), (50, 30, 20, 50)$. After the tuning phase, we decide to

use the *tanh* activation function, *adam* optimizer, and $(50, 30, 20, 50)$ configuration of layers/nodes. For RF, MLP, and SVM, we use 5-fold cross-validation.

For MLP without feature selection, we use an architecture from the related works, denoted as ASCAD_MLP [40]. It consists of six hidden layers with 200 neurons in each layer and the *ReLU* activation function.

For CNN, we consider the architecture from [40], and we denote it ASCAD_CNN. Note that while this architecture was originally designed for the ASCAD dataset, we adjust the input layer size and use it in all our experiments. This architecture consists of four convolutional layers and two fully-connected layers. The filter numbers of each convolutional layers are $(64, 128, 256, 512)$. The filter size and pooling stride are set to 11 and 2, respectively. For the fully-connected layers, each layer contains 4 096 neurons.

Additionally, we use different architectures based on [9]. We denote them as "methodology", and for each dataset, we use the architecture designed for that dataset. For DPAv4 and AES_HD, we use one convolutional block and a fully-connected layer with two neurons. The convolutional block consists of the convolution layer (two filters) and the average pooling layer (pooling size and strides 2). The learning rate is 0.001. For ASCAD, we use one convolutional block and two fully-connected layers with ten neurons each. The convolutional block consists of the convolution layer (four filters) and the average pooling layer (pooling size and strides 2). The learning rate is 0.005. Finally, for AES_RD, we use three convolutional blocks and two fully-connected layers with ten neurons each. The first convolutional block consists of the convolution layer (eight filters) and the average pooling layer (pooling size and strides 2). The second block has 16 filters in convolution, and the average pooling layer with size and strides 50. Finally, the third block has convolution with 32 filters and average pooling with stride and size 7. The learning rate is 0.00001. All CNN architectures use the $SELU$ activation function. For MLP and CNN, we run experiments for 100 epochs, and we use batch normalization (to normalize the input layer by adjusting and scaling the activations) equal to 100. We use a validation dataset of 5 000 traces.

4.5 Framework Setting

We conduct our experiments for a scenario where $\epsilon = 0.1$ and $\delta = 0.1$. The Chernoff bound (Eq. (4)) gives $n \geq 149.7$ to achieve the desired level of accuracy and confidence. Consequently, we set $n = 150$ in our experiments, which means that every experiment is repeated 150 times to obtain statistically significant results. Next, we select to work with noise α equal to 0.005 that goes in the range $[-\alpha \cdot if, \alpha \cdot if]$, where if denotes the intensity factor in the range $[1, 50]$. By doing so, we will evaluate noise levels up to 25% of the signal, which would capture many realistic settings (more noise is, of course, possible). More precisely, we can consider our scenario as working with 50 different noise intensities (perturbations). Recall, the noise we inject could arise from various sources in practical applications, for example:

– variations between the profiling and attacking device [12],
– environmental noise between acquisition campaigns,
– (minor) changes in the experimental setup (probes, devices, hyperparameters, points of interest).
– effect of countermeasure randomness.

5 Framework Validation and Application

In this section, we first use our framework with the simulated traces to validate its correctness. Next, we use it to evaluate the profiling attacks' performance on the publicly available datasets. In total, we conduct more than 500 experiments to evaluate our framework objectively.

Note that our framework considers the setting where we add noise to the attack traces only. This simulates the behavior one would encounter in, e.g., 1) portability due to differences between profiling device and the device under attack, 2) real-world applications due to the changes in the environment setup (e.g., changes in the probe location for EM acquisition [12], environment noise, etc.), 3) targets with countermeasures as they would cause differences between profiling and testing device, and 4) changes in the hyperparameter setting as suboptimal hyperparameter choices result in less powerful profiling model that cannot fit the leakage well. We do not add perturbation to both training and attack traces, as this would simulate different devices (when, e.g., testing the influence of a certain countermeasure, one needs to consider it both for the profiling and attack phases). Finally, we do not add perturbation to the training phase only as it would serve as a regularization factor that helps the classification procedure [8]. Additionally, we emphasize that this evaluation aims not to find the best performing methods but to confirm our framework's validity and obtain information about the robustness of profiling methods.

In Algorithm 1, we present the pseudocode of the procedure we follow to assess the robustness of a certain classifier against perturbations in the attack phase. Since the number of samples equals 150 ($n = 150$) and the number of intensities equals 50 ($f = 50$), it means that for each scenario, we need to run the attack 7 500 times. Note that the fact we repeat the experiments multiple times (here, $n = 150$) is connected with the procedure of how one obtains guessing entropy from the key rank. As guessing entropy is the average key rank calculated on randomly selected traces, there is no difference from simply calculating key rank n times with different Gaussian noise. As such, we speed up the process and do not require key rank to be averaged 100 times (as commonly done in the literature), but instead, we repeat the procedure 150 times as per Chernoff bound. [7]

[7] We additionally run the experiments where we averaged 100 key rank experiments for each intensity f and value n, but did not notice any statistically significant difference.

Algorithm 1 Algorithm to solve the probabilistic robustness evaluation problem [34].

Identify the perturbation space Δ and the random variable $\delta\theta$ with pdf $f_{\delta\theta}$ over Δ;
Select the accuracy ϵ and confidence δ
Identify the interested performance level set $\Gamma = \{\gamma_1, \cdots, \gamma_k\}$
$\hat{p}_{n,\Gamma}(\gamma)$= verification-problem$(\Delta, f_{\delta\theta}, u(\delta\theta), \Gamma, \epsilon, \delta)$
use $\hat{p}_{n,\Gamma}(\gamma)$

function verification-problem $(\Delta, f_{\delta\theta}, u(\delta\theta), \Gamma, \epsilon, \delta)$:
Draw $n \geq \frac{1}{2\epsilon^2} \ln \frac{2}{\delta}$ samples $\delta\theta_1, \cdots, \delta\theta_n$ from $\delta\theta$ according to $f_{\delta\theta}$
For each $\gamma \in \Gamma$ estimate

$$\hat{p}_n(\gamma) = \frac{1}{n} \sum_{i=1}^{n} I\left(u(\delta\theta_i) \leq \gamma\right), \; I\left(u(\delta\theta_i) \leq \gamma\right) = \begin{cases} 1 \text{ if } u(\delta\theta_i) \leq \gamma \\ 0 \text{ if } u(\delta\theta_i) > \gamma \end{cases}$$

Return $\hat{p}_{n,\Gamma}$

5.1 Framework Validation on Simulated Traces

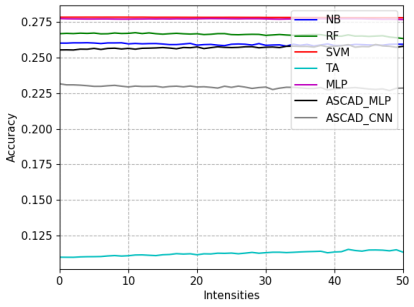
In general, it is not an easy problem to validate a theoretical framework for machine learning as one needs to consider all possible scenarios. As this is impossible in practice, we first use the concept of stylized facts, which is a generalization that summarizes data. More precisely, we examine the framework's behavior with the simulation data and compare it with real data. If the simulation data behaves in the same manner as the real data, we can assume that the framework can indeed be used with real data. Naturally, this approach must handle certain inaccuracies as simulation data is far from a perfect representation of the real data. Here, we use the simulated traces where we add Gaussian noise, as discussed in the next section. This dataset can then be compared with DPAv4, as they are the best match considering the noise levels.

Simulated Traces - Data Augmentation and Noise Addition Recall, our simulated dataset contains 256 traces, one for each possible S-box output value and 5 000 features. We select 50 features that correspond to the highest correlation between the traces and the S-box output to match our other datasets. Next, we need to extend this dataset with additional traces to build a reliable profiling model. We do this by adding Gaussian noise to the simulation traces:

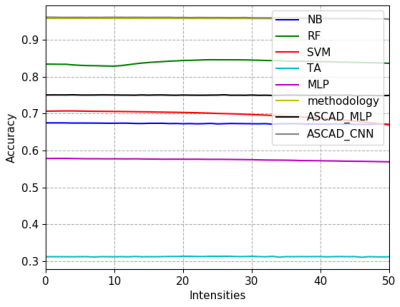
$$X' = X + N. \tag{13}$$

We make the simplified but still commonly used assumption that the noise is univariate Gaussian distributed with zero-mean, i.e., $N \sim \mathcal{N}(0, \sigma^2)$ where σ is the standard deviation. We repeat this procedure 15 000 times to create 15 000 noisy traces with $\sigma = (0, 0.5]$. Note that our final dataset contains approximately the same number of traces for each S-box output value. We divide such simulated and data augmented dataset into 10 000 traces for training and 5 000 for testing. We denote this dataset as AES_SIM.

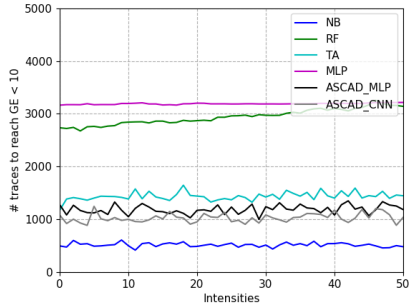
In Figure 2, we depict the behaviors for the simulated dataset (AES_SIM) and real-world dataset (DPAv4) when adding perturbations. First, in Figures 2a and 2b, we depict the results for accuracy. Note that while the accuracy levels are significantly different (as AES_SIM has much more noise than DPAv4), all classifiers' general behavior follows the same trends. Similar observations hold for Figures 2c and 2d, where we observe a slight decrease in the performance, coupled with more instability of results due to added noise. Still, note that the classifiers behave in the same way, see, e.g., the stable behavior for MLP in both cases.



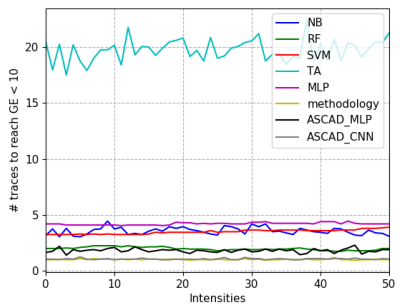
(a) AES_SIM dataset, HW leakage model, accuracy.



(b) DPAv4 dataset, HW leakage model, accuracy.



(c) AES_SIM dataset, HW leakage model, GE.

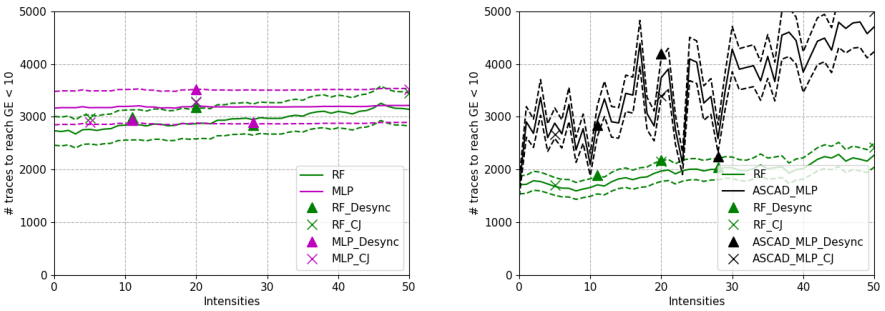


(d) DPAv4 dataset, HW leakage model, GE.

Fig. 2: Comparisons between AES_SIM and DPAv4 datasets.

In the second experiment with the simulated dataset, we aim to confirm that perturbed traces represent a good approximation of various countermeasures. Consequently, we use the same 15 000 traces as before (256 simulated traces data augmented to 15 000 with Gaussian noise). On this dataset, we again consider 50 perturbation intensities (Gaussian noise from 0% to 25% of the signal). Addition-

ally, we introduce the countermeasures to the dataset, where we consider clock jitter and desynchronization countermeasures (the pseudocode is given in Supplementary Material A.1). Then, under the assumption that our framework can model the behavior of various countermeasures, we expect that the attack results for datasets with clock jitter or desynchronization countermeasures are aligned with perturbed traces (for instance, we can estimate signal-to-noise (SNR) ratio for both datasets, and expect the results to be similar for similar SNR values). As a result, we use traces with 1/2/3 point(s) of desynchronization or 1/2/3 of the clock jitters level. One can expect that with an increased countermeasure level, the SNR of the traces is reduced. We show the results in Figure 3. Acronym $\_Desync$ denotes desynchronization added to the traces, and $\_CJ$ clock jitter added. Full lines depict the perturbation experiments, and dotted lines denote the $\pm 10\%$ ranges. Crosses and triangles denote the result obtained for the same profiling model if instead of perturbations, one considers the dataset with countermeasures. The results clearly show that traces with countermeasures are aligned with the perturbed traces. This indicates that our framework can model various countermeasures, and consequently, it can be used to estimate the influence of adding countermeasures to a target. Indeed, by considering the results for various intensities, we can extrapolate whether a certain attack method is robust and the influence of the attack performance if adding a specific countermeasure to the target.



(a) HW leakage model.

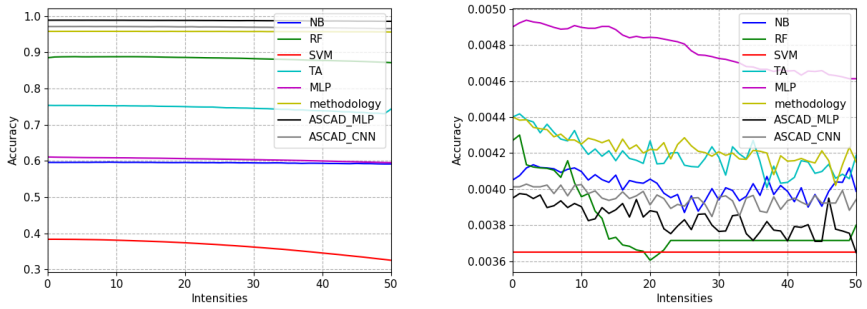
(b) ID leakage model.

Fig. 3: Results for perturbed traces and traces with added countermeasures.

5.2 Publicly Available Datasets and Framework Evaluation

Next, we present and briefly discuss several characteristic scenarios for publicly available datasets. Note that we select only a few characteristic examples of solutions, and we do not give all the results for all considered scenarios. Note that the figures show results for both classifiers with and without feature selection.

In Figure 4, we depict results when considering accuracy. First, in Figure 4a, we consider the DPAv4 dataset in the Hamming leakage model. We observe that classifiers behave 1) "stable" in the sense that their accuracy with the increase in the perturbation level slightly decreases, or 2) "resilient", as we observe no influence of perturbation to the attack performance. This behavior is as expected as the DPAv4 dataset is easy to attack, and even by adding Gaussian noise, all considered datasets reach good performance. Somewhat surprising is that SVM behaves the worst and is the most sensitive to perturbation, but our experiments do indicate SVM to be very sensitive to hyperparameter tuning, which explains its relatively poor performance. Note that all techniques from the deep learning domain considering all features (ASCAD_MLP, ASCAD_CNN, methodology) perform much better than simpler attacks and deteriorate marginally even for the highest intensity level. Already considering the identity leakage model in Figure 4b shows significantly different behavior. We depict only four attacks as the rest of the methods have accuracy equal to zero regardless of the intensity level. Here, we observe that all attacks deteriorate with the increase in the perturbation intensity, where the best performing method deteriorates the most. Since we deal with the scenario with 256 labels, these results are not surprising as the classification task is much more difficult.



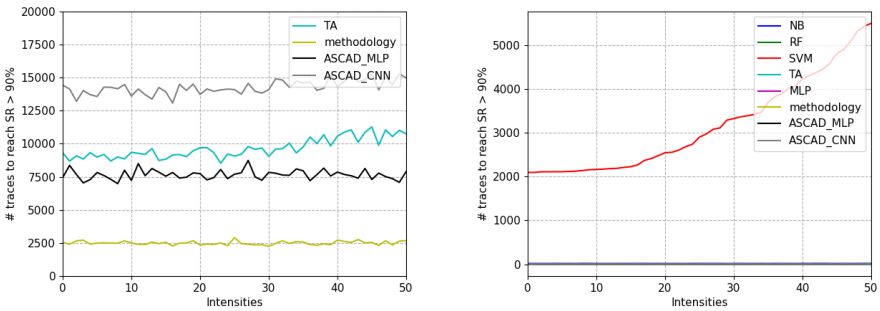
(a) DPAv4, HW leakage model.

(b) AES_HD, ID leakage model.

Fig. 4: Results for accuracy.

In Figure 5, we consider the number of attack traces to reach a success rate of 0.9. Note that here the lower the value, the better is attack. The two best attacks for the HW leakage model do not suffer due to the addition of noise, while TA and ASCAD_CNN indeed deteriorate when considering higher noise levels. For the identity model, we show an interesting case in Figure 5b, where SVM consistently decreases in the performance with the addition of noise. We can consider this perfect example of an attack method that performs stable under perturbation, where stable means performance is gradually decreasing. Deep learning architectures do not have any performance degradation, which

indicates they are more robust. Note that in Figure 5a, several classifiers lines overlap around zero, which means they are exhibit resilient behavior.

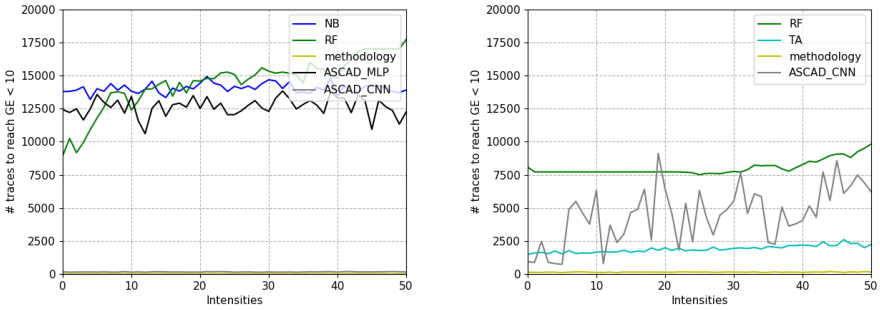


(a) ASCAD fixed key, HW leakage model. (b) DPAv4, ID leakage model.

Fig. 5: Results for the number of traces to reach $SR = 0.9$.

Figure 6 depicts results for the number of traces needed to reach GE less than 10. Note that this scenario is relatively resilient as we do not differentiate between the number of traces while $GE < 10$. As such, we can expect slower degradation of attack performance as while GE is lower than 10, we require the same number of attack traces, regardless of the exact GE value. Note that CNN techniques perform well for the Hamming weight leakage model, and we do not see any influence due to noise addition. This represents the resilient (to perturbation) behavior. At the same time, MLP deteriorates with the increase in noise, which is expected for profiling models that do not have enough capacity to account for the noise in the attack phase. Stated differently, those techniques overfit as they cannot generalize to different setup (a correct term would be that they overspecialize [10] as we change the attack dataset). For the identity leakage model (Figure 6b, we observe a new type of behavior for ASCAD_CNN: adding noise results in an unpredictable behavior. Then, it is impossible to estimate the behavior for any specific noise level, but at the same time, we do see a tendency to reduced attack performance. This behavior can be called "unstable" due to the inability to estimate the behavior for any specific noise level. Other attack methods behave stable as we observe a slight decrease in attack performance as the noise level increases.

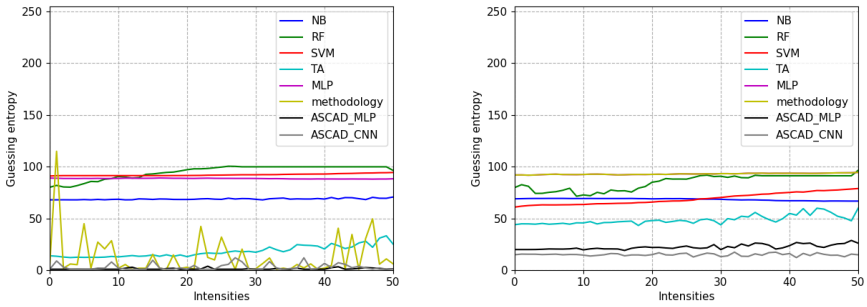
Finally, in Figure 7, we depict the results for the GE metric. We calculate it in the following way: we find the minimal number of attack traces required to reach the best possible performance without considering perturbations. Then, we always take that number of attack traces and add perturbations. When considering the HW leakage model (Figure 7a), we notice either a very gradual decrease in the attack performance or erratic behavior (the methodology CNN architecture). This indicates that most of the classifiers are indeed robust, and



(a) AES_RD, HW leakage model. (b) ASCAD fixed key, ID leakage model.

Fig. 6: Results for number of traces to reach $GE < 10$.

only highly specialized architectures could face some issues. The extension of this behavior can be seen in Figure 7b, where again most of the classifiers are relatively stable, and methodology becomes stable but only because it performs poorly. From that perspective, we can call this type of behavior "no learning" as the architecture is not sensitive to noise since it did not learn the profiling model well. As a conclusion (a relatively natural one), we see that highly specialized architectures may not be the best choice if considering real-world settings like portability or any difference in the environment setup between the training and testing phases.



(a) DPAv4, HW leakage model. (b) DPAv4, ID leakage model.

Fig. 7: Results for GE.

Finally, as we run the experiments with different dataset sizes (for framework validation and framework evaluation on the publicly available dataset), and with different hyperparameters (different MLP or CNN architectures), we observe

that the influence of those framework parameters is not so strong when dealing with larger perturbations. This indicates less need for very precise hyperparameter tuning if one expects differences between training and testing datasets.

6 General Observations and Remarks

There are four main types of behavior one can expect when modeling the system with perturbations. We give the list in the order from the most preferred to the least preferred setting (where preference is aligned with the attacker perspective):

1. "Resilient" behavior. In this setting, the classifier is resilient to perturbations, which means that its performance remains practically unchanged even in the presence of noise. Naturally, after some point, the performance starts to deteriorate, and then, the behavior resembles one of the following types.
2. "Stable" behavior. In this setting, the classifier is affected by perturbation, and the performance is gradually decreased.
3. "Unstable" behavior. Here, a small perturbation results in a significant performance drop or very erratic behavior, i.e., jumps between good and poor performance.
4. "No learning" behavior. With this behavior, the perturbation does not influence the performance of the profiling method. This happens because the classifier, even before perturbation, was not working, and naturally, making the problem more difficult cannot improve the behavior but also cannot deteriorate it. While this behavior has certain similarities with the first one, the underlying idea is different. Here, the classifier never works while in the "Resilient" setting, it is stable up to a certain level of perturbation (i.e., no difference between that level of perturbation and no perturbation).

Besides these four types, several subtypes can be recognized by combining the basic behaviors' traits.

The main advantages of using the proposed framework are:

- The framework allows an objective and reliable way to estimate the performance of the profiling side-channel analysis.
- The framework supports considering robustness analysis in different ways. Indeed, here we opted to analyze the classifiers' robustness to the addition of noise in the attack phase. This approach fits different scenarios, from portability, changes in the dataset size, countermeasures, hyperparameter changes, classifiers change, etc. Still, we could also evaluate the robustness concerning some other perturbation like the addition or removal of features from traces or pre-processing approaches [48].
- With our framework, it is possible to "map" different setting: for instance, 1) to see what level of perturbation causes the classifiers to behave in the same way as, e.g., having a masking countermeasure, or, 2) at what point the suboptimal hyperparameter results in the same behavior as an added perturbation.

Ultimately, our framework allows us to directly apply to the profiling side-channel analysis problem the whole body of knowledge developed in years of studying the robustness problem. Based on these advantages, we immediately see one very practical application of our framework: to help designers/manufacturers to make more secure systems. With it, we can observe the added benefit of a certain countermeasure and if it is sufficient to prevent a certain type of attack. Thus, by using this framework, we can design systems that are more secure against various types of attacks.

Our framework is based on robustness analysis. We inherit the advantages of such an approach, but also the limitations. We list two of them, discussing their direct implication to the profiling side-channel attacks problem:

- One needs to know what to evaluate, i.e., what perturbation to take into account.
- Observing the robustness does not necessarily tell us how to make the classifier more robust.

Note that those limitations are not connected with the SCA domain, but rather the robustness analysis. Indeed, one needs to know what to evaluate, and why would that make sense. Finally, to make the classifier more robust, one would need to conduct perturbations in the sense of the hyperparameter changes.

In this paper, we modeled the perturbations as Gaussian noise added to the attack traces only. This allowed us to reach the following general conclusions based on our experimental results:

- There are four main types of behaviors we can observe with classifiers.
- Easier datasets or leakage models with fewer labels are, in general, more resilient to perturbations.
- Deep learning is, in general, more resilient to perturbations.
- Different types of countermeasures can have the same effect on the classifier, and different types of countermeasures can be approximated with Gaussian noise.

7 Conclusions

In this paper, we concentrate on profiling side-channel attacks and uncertainties stemming from the experimental results. To that end, we propose a general framework that can be used to analyze the behavior of any profiling side-channel attack (i.e., a method performing the classification task). Our framework supports datasets with any characteristics as well as different leakage models. To offer such a general behavior, we model it as the expectation estimation problem to achieve any desired accuracy and confidence level. After we model the classifiers, we use the robustness analysis to estimate their performance in the presence of perturbations. Such an analysis allows us to answer which classifier is the most robust for a specific scenario. We give robustness analysis for several figures of merit: accuracy, success rate, guessing entropy, and the number of traces to reach a specific attack performance.

We believe our framework will be a powerful tool that will allow researchers to compare various classifiers' behavior more fairly than it is done up to now. Since profiling SCA in realistic settings should consider different profiling and attacking devices, we expect that our framework will allow more than just connecting SCA with problems that have reliable theoretical results. More precisely, our framework allows modeling the realistic behavior of profiling SCA, where uncertainty must occur due to several different noise sources.

We note that our framework can be demanding: to conduct a proper analysis, the Chernoff bound requires a large number of experiments, which potentially can be a prohibiting factor for specific computationally intensive classifiers or very large datasets. One easy way how to circumvent this is to parallelize the process for different perturbation levels. Another interesting direction would be to consider different sources of perturbations, e.g., not only adding removing noise, but also adding/removing features from a dataset.

References

1. Mangard, S., Oswald, E., Popp, T.: Power Analysis Attacks: Revealing the Secrets of Smart Cards. Springer (December 2006) ISBN 0-387-30857-1, `http://www.dpabook.org/`.
2. Choudary, O., Kuhn, M.G.: Efficient template attacks. In Francillon, A., Rohatgi, P., eds.: Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. Volume 8419 of LNCS., Springer (2013) 253–270
3. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: CHES. Volume 2523 of LNCS., Springer (August 2002) 13–28 San Francisco Bay (Redwood City), USA.
4. Heuser, A., Zohner, M.: Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Schindler, W., Huss, S.A., eds.: COSADE. Volume 7275 of LNCS., Springer (2012) 249–264
5. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Preprocessing. In: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. (2017) 45–68
6. Picek, S., Heuser, A., Jovic, A., Ludwig, S.A., Guilley, S., Jakobovic, D., Mentens, N.: Side-channel analysis and machine learning: A practical perspective. In: 2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017. (2017) 4095–4102
7. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1) (Nov. 2018) 209–237
8. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(3) (2019) 148–179
9. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1) (Nov. 2019) 1–36

10. van der Valk, D., Picek, S., Bhasin, S.: Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/1477 (2019) https://eprint.iacr.org/2019/1477.
11. Masure, L., Dumas, C., Prouff, E.: Gradient visualization for general characterization in profiling attacks. In Polian, I., Stöttinger, M., eds.: Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings. Volume 11421 of Lecture Notes in Computer Science., Springer (2019) 145–167
12. Bhasin, S., Chattopadhyay, A., Heuser, A., Jap, D., Picek, S., Shrivastwa, R.R.: Mind the portability: A warriors guide through realistic profiled side-channel analysis. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020, The Internet Society (2020)
13. Valiant, L.: Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World. Basic Books, Inc., New York, NY, USA (2013)
14. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of CRYPTO'96. Volume 1109 of LNCS., Springer-Verlag (1996) 104–113
15. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. CRYPTO '99, London, UK, UK, Springer-Verlag (1999) 388–397
16. Quisquater, J.J., Samyde, D.: Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Attali, I., Jensen, T., eds.: Smart Card Programming and Security, Berlin, Heidelberg, Springer Berlin Heidelberg (2001) 200–210
17. Genkin, D., Shamir, A., Tromer, E.: Acoustic cryptanalysis. Journal of Cryptology **30**(2) (Apr 2017) 392–443
18. Schindler, W., Lemke, K., Paar, C.: A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, ed.: CHES. Volume 3659 of LNCS., Springer (Sept 2005) 30–46 Edinburgh, Scotland, UK.
19. Heuser, A., Rioul, O., Guilley, S.: Good is Not Good Enough — Deriving Optimal Distinguishers from Communication Theory. In Batina, L., Robshaw, M., eds.: CHES. Volume 8731 of Lecture Notes in Computer Science., Springer (2014)
20. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In Mangard, S., Poschmann, A.Y., eds.: Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers. Volume 9064 of Lecture Notes in Computer Science., Springer (2015) 20–33
21. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. Journal of Cryptographic Engineering **1** (2011) 293–302 10.1007/s13389-011-0023-x.
22. Lerman, L., Bontempi, G., Markowitch, O.: Power analysis attack: An approach based on machine learning. Int. J. Appl. Cryptol. **3**(2) (June 2014) 97–115
23. Lerman, L., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES - Reaching the limit of side-channel attacks with a learning model. J. Cryptographic Engineering **5**(2) (2015) 123–139
24. Lerman, L., Medeiros, S.F., Bontempi, G., Markowitch, O.: A Machine Learning Approach Against a Masked AES. In: CARDIS. Lecture Notes in Computer Science, Springer (November 2013) Berlin, Germany.

25. Picek, S., Heuser, A., Guilley, S.: Template attack versus Bayes classifier. Journal of Cryptographic Engineering **7**(4) (Nov 2017) 343–351
26. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). (May 2015) 106–111
27. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers **PP**(99) (2017) 1–1
28. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. (2016) 3–26
29. van der Valk, D., Picek, S.: Bias-variance decomposition in machine learning-based side-channel analysis. Cryptology ePrint Archive, Report 2019/570 (2019) `https://eprint.iacr.org/2019/570`.
30. Standaert, F.X., Koeune, F., Schindler, W.: How to compare profiled side-channel attacks? In: International Conference on Applied Cryptography and Network Security, Springer (2009) 485–498
31. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In Joux, A., ed.: Advances in Cryptology - EUROCRYPT 2009, Berlin, Heidelberg, Springer Berlin Heidelberg (2009) 443–461
32. Bronchain, O., Hendrickx, J.M., Massart, C., Olshevsky, A., Standaert, F.X.: Leakage certification revisited: Bounding model errors in side-channel security evaluations. In: Annual International Cryptology Conference, Springer (2019) 713–737
33. Durvaux, F., Standaert, F.X., Veyrat-Charvillon, N.: How to certify the leakage of a chip? In: Annual International Conference on the Theory and Applications of Cryptographic Techniques, Springer (2014) 459–476
34. Alippi, C.: Intelligence for Embedded Systems: A Methodological Approach. Springer Publishing Company, Incorporated (2014)
35. Rudin, W.: Real and Complex Analysis, 3rd Ed. McGraw-Hill, Inc., New York, NY, USA (1987)
36. Chernoff, H.: A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. Ann. Math. Statist. **23**(4) (12 1952) 493–507
37. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58**(301) (1963) 13–30
38. TELECOM ParisTech SEN research group: DPA Contest (4^{th} edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
39. Coron, J., Kizhvatov, I.: An Efficient Method for Random Delay Generation in Embedded Software. In: Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings. (2009) 156–170
40. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptogr. Eng. **10**(2) (2020) 163–188
41. Regazzoni, F., Cevrero, A., Standaert, F.X., Badel, S., Kluter, T., Brisk, P., Leblebici, Y., Ienne, P.: A design flow and evaluation framework for DPA-resistant instruction set extensions. In Clavier, C., Gaj, K., eds.: CHES09. Volume 5747 of LNCS. Springer (September 2009) 205–19
42. Fernández-Delgado, M., Cernadas, E., Barro, S., Amorim, D.: Do we Need Hundreds of Classifiers to Solve Real World Classification Problems? Journal of Machine Learning Research **15** (2014) 3133–3181

43. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **29**(2) (1997) 131–163
44. Fan, R.E., Chen, P.H., Lin, C.J.: Working Set Selection Using Second Order Information for Training Support Vector Machines. J. Mach. Learn. Res. **6** (December 2005) 1889–1918
45. Breiman, L.: Random Forests. Machine Learning **45**(1) (2001) 5–32
46. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016) `http://www.deeplearningbook.org`.
47. James, G., Witten, D., Hastie, T., Tibsihrani, R.: An Introduction to Statistical Learning. Springer Texts in Statistics. Springer New York Heidelbert Dordrecht London (2001)
48. Wu, L., Picek, S.: Remove some noise: On pre-processing of side-channel measurements with autoencoders. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(4) (Aug. 2020) 389–415

A Supplementary Material

A.1 Adding the Countermeasures

For both countermeasures, we use the approach given by Wu and Picek [48]. The desynchronization of the traces adds randomness to the time domain. The pseudocode for constructing traces with desynchronization is shown in Algorithm 2.

Algorithm 2 Add Desynchronization.

function ADD_DESYNC($trace, desync\_level$)
  $new\_trace \leftarrow []$ ▷ container for new trace
  $level \leftarrow$randomNumber$(0, desync_level)$
  $i \leftarrow 0$
  while $i + level < len(trace)$ **do**
    $new\_trace[i] \leftarrow traces[i + level]$ ▷ add desynchronization to the trace
    $i \leftarrow i + 1$
  return new_trace

Clock jitters is a typical hardware countermeasure against SCA, realized by introducing the instability in the clock [5]. The clock jitters increase the randomness for each point in the time domain, where the accumulation of the deforming effect increases the misalignment of the traces and decreases the intermediate data correlation. We simulate the clock jitters by randomly adding or removing points with a pre-defined range. Similar approaches are used in [5]. More precisely, we generate a random number r, when scanning each point in the trace, r points will be added to the trace if r is larger than zero. Otherwise, the following r points in the trace are deleted. The pseudocode for constructing traces with clock jitters is shown in Algorithm 3. The manipulated traces are then padded with zero to keep the traces the same length.

Algorithm 3 Add Clock Jitters.

```
1: function ADD_CLOCK_JITTERS(trace, clock_jitters_level)
2:     new_trace ← []                                   ▷ container for new trace
3:     i ← 0
4:     while i < len(trace) do
5:         new_trace[i] ← new_trace[i].append(trace[i])
6:         r ←randomNumber(0, clock_jitters_level)      ▷ level of clock jitters
7:         if r < 0 then
8:             i ← i + r                                ▷ skip points
9:         else
10:            j ← 0
11:            average_amplitude ← (trace[i] + trace[i + 1])/2
12:            while j < r do
13:                new_trace ← new_trace.append(average_amplitude)   ▷ add points
14:                j ← j + 1
15:        i ← i + 1
16:    return new_trace
```