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

Stjepan Picek[1], Annelie Heuser[2], 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. Profiled 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 to. The research community investigated profiled attacks in-depth, mostly by using an empirical approach. As such, it emerges that a theoretical framework to comprehensively analyze profiled side-channel attacks is still missing.

In this paper, we propose a theory grounded framework capable of modeling and evaluating profiled 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 poses 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 not sufficient 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 physical weaknesses of the implementation to extract the stored secret information (typically, the secret key).

It is necessary to have a complete understanding of the adversary's capabilities to achieve resistance against physical attacks. Side-channel attacks (SCAs), in particular profiled 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 profiled attacks, the adversary first profiles the device that is identical (or, at least similar) to the one that will be attacked. In the second phase, using this profile and the traces measured from the victim device, the adversary attempts to recover the secret key. Well-known examples of such profiled 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 point of view (given that certain assumptions hold) [3]. At the same time, a large literature experimentally shows how machine learning techniques perform extremely well in many realistic scenarios, see, e.g., [5, 9]. However, there is no guarantee about how machine learning techniques would behave in different scenarios, even if they are similar. Still, one would expect 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?

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

The current state-of-the-art in profiled side-channel analysis progressed tremendously in the last few years. There, results with deep learning show 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 in efforts to produce more powerful attacks, but also novel countermeasures. At the same time, to the best of our knowledge, no results are offering theoretical insights into the performance of machine learning-based attacks or providing frameworks in which attacks can be evaluated better.

In this paper, we aim at filling in this gap and offer results considering the general performance of machine learning attacks (and, in fact, all profiled attacks). We propose a general framework intended to analyze the behavior of profiled SCAs and give insights into the robustness of such methods. We define by term robustness the ability of a system to tolerate perturbations (random changes affecting the system). Although we use the robustness analysis as a tool to obtain insights into the general behavior of profiled attacks, perturbations are commonly occurring in SCA. Several sources of perturbations occur in profiled SCA due to 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 [10]. 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 [11], 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 profiled side-channel attacks where we provide strong theoretical foundations for the framework.
– We consider the robustness of profiled attacks in 1) the presence of countermeasures and 2) environment settings like feature selection, dataset size, and hyperparameter tuning.
– We consider the robustness of profiled attacks for commonly used figures of merit: accuracy, success rate, 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 profiled 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: profiled methods, datasets, figures of merit, and framework parameters we use. 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 results for four publicly available datasets. Section 6 presents a discussion about general findings from the experiments and advantages of using our framework. Finally, Section 7 concludes the paper and Appendices A and B give experimental results when changing the profiling models and dataset sizes, respectively.

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 [12]. In other words, implementations of cryptographic algorithms leak relevant information about the data processed through physical side-channels like timing [12], power consumption [13], EM emanation [14], or sound [15]. Today, when considering SCA and symmetric-key cryptography, there are two main directions one could follow:

1. direct attack, see, e.g., Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [13].
2. profiled attacks, see, e.g., Template Attack (TA) [3], stochastic models [16], or a number of machine learning-based techniques.

Profiled attacks require a profiling stage, i.e., a step during which the cryptographic hardware is under full control of the adversary to estimate the leaked information's probability distribution. Such attacks have received much attention in recent years because they define the worst-case security assumptions. There, a template attack is a de-facto standard in the SCA community. TA is the best (optimal) technique from an information-theoretic point of view if the attacker has an unbounded number of traces, and the noise follows the Gaussian distribution [17, 18]. After the template attack, the stochastic attack emerged using linear regression in the profiling phase [16]. 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, some 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, 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., [4,6,18–25]. More recently, deep learning started to capture the attention of the SCA community. The first results confirmed that expectations where most of the attention went to convolutional neural networks [5, 7–9, 26].

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 performance and often outperform template attack. Additionally, deep learning is commonly able to 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 some attempts in explaining why such results are obtained.

Besides works that consider how to improve attack performance, more recently, there have been several works considering the topics of explainability

and interpretability of deep learning in side-channel analysis [27–29]. Finally, to the best of our knowledge, there is one work considering a framework for profiled side-channel analysis, but there, the authors concentrate on the profiling set size [30]. We note that our framework encompasses this perspective as we model changes in setup or scenario by perturbing the attack set.

3 General Framework

In this section, we present a framework that can model all profiled 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 profiled side-channel attacks and show how this is tightly connected with the PAC learning concept.

3.1 Threat Model

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

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 corresponding leakage measurement trace is stored. Ideally, the adversary can have infinite queries and corresponding database of side-channel leakage measurements to characterize a precise model. Next, the adversary queries the attack device with known plaintext to obtain the unknown key. The corresponding side-channel leakage measurement is compared to the characterized model to recover the key.

In the rest of this paper, when discussing profiled side-channel attacks, we consider those attacks that use power or electromagnetic radiation as side-channel. Finally, we consider only those scenarios where the task is classification, i.e., to learn how to assign a class label to examples.

3.2 Intuitive Description of the Framework

The previous works on profiled 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 in order to model profiled 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 measurements taken corresponding to the encryption of a certain number of plaintexts PT and a number of known keys K. For each measurement 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 measurements x_i, we additionally obtain the label of the measurement y_i, which denotes the actual value the measurement has. The leaked traces are typically altered by a random perturbation δ_1 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 models. To model this phenomenon, we consider the given classifier's output influenced by a specific perturbation δ_2, which implies we are assuming the intensity (variance) of the two perturbations can be the same (i.e., different points are characterized by the same SNR or values affected by perturbations insist on the same bounded interval). Without loss of generality, we consider δ_1 and δ_2 to be equal to δ 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 [31]. 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 profiled 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, to date, was 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 profiled side-channel attack and thus compare, in a sound 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 in a sound way the questions like "Which classifier family behaves better in some specific scenario?" and "How to compare different profiled side-channel attacks?".

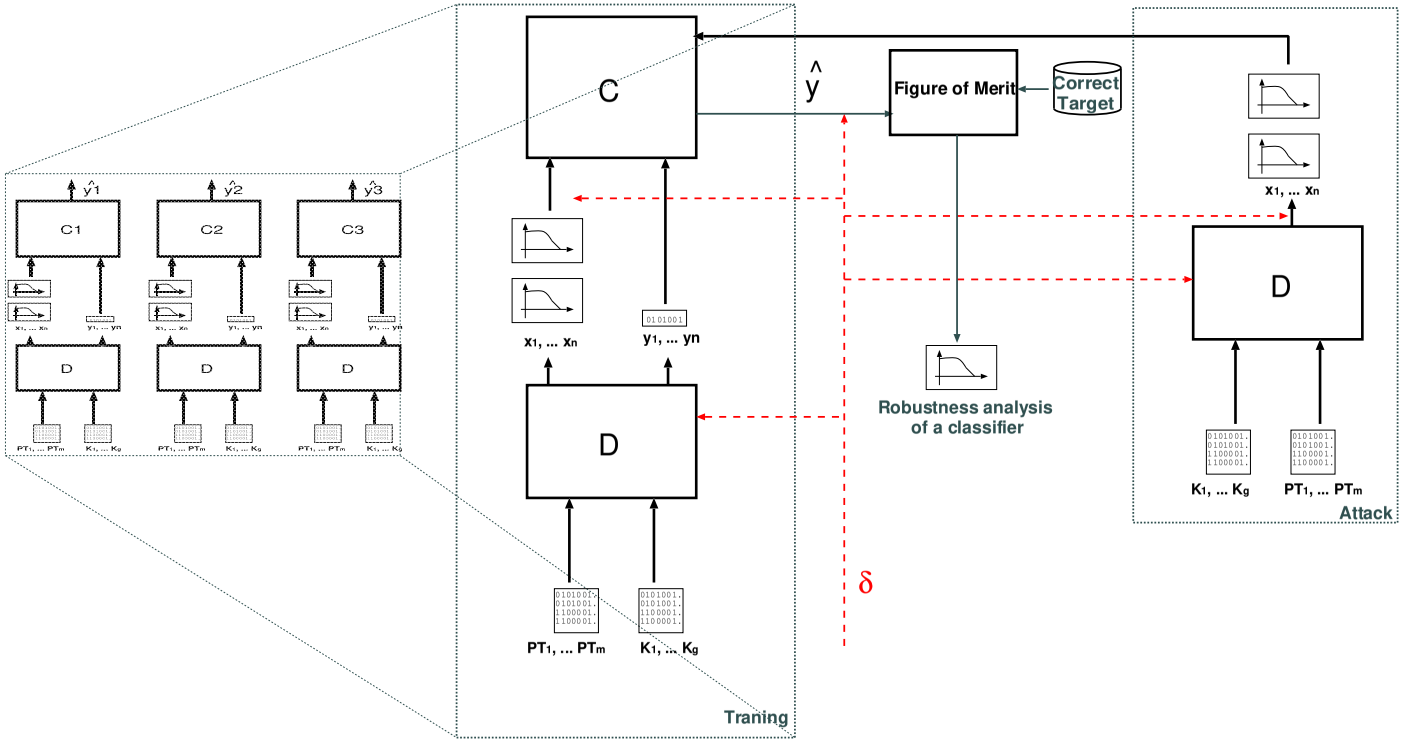


Fig. 1: The framework for the profiled side-channel analysis. The red line denotes the locations of possible perturbations. There are two phases of the attack. Profiling phase (left part of the figure in bold) and attack phase (right part of the figure in bold). Since the framework is generic, it supports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

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).

Lebesgue Measurability 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 [32]. Basically, no engineering-related mathematical computations are Lebesgue non-measurable.

The Chernoff Bound 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 [33]. The Chernoff bound for a generic probability density function and continuous variable ψ can be derived from the Hoeffding inequality for the empirical mean [34].

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; the statement holds with confidence δ.

The Expectation Estimation Problem The expectation estimation problem consists in 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:

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

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.

More formally, 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 Profiled SCA Framework

We now express profiled side-channel attacks as the expectation estimation problem. The starting point is to map the steps of profiled analysis attacks to the framework depicted in Figure 1. The first phase of profiled 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 measurement 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 profiled analysis, there are two main models to define $y(t, k^*)$ so to calculate the labels of the measurement traces:

– *intermediate value model*: in this model, the attacker considers an intermediate value of the cipher or the distance between two consecutive values processed.

– *Hamming weight (HW) model*: in this model, the attacker assumes the HW of the intermediate value model.

Considering the intermediate value, the model may be more accurate but requires more resources. In particular, to gain stable estimations for each possible value, an attacker needs a sufficient amount of measurement traces 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 profiled methods.

The adversary first profiles the clone device with the known keys and uses the obtained profiles for the attack. In particular, the attack operates in two phases:

– *profiling phase:* N traces $\boldsymbol{x}_{p_1}, \ldots, \boldsymbol{x}_{p_N}$, plaintext/ciphertext $t_{p_1}, \ldots, t_{p_N}$ and the secret key k_p^*, such that the attacker can calculate the labels $y(t_{p_1}, k_p^*)$, …, $y(t_{p_N}, k_p^*)$.
– *attacking phase:* Q traces $\boldsymbol{x}_{a_1}, \ldots, \boldsymbol{x}_{a_Q}$ (independent from the profiling traces), plaintext/ciphertext $t_{a_1}, \ldots, t_{a_Q}$.

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

In this section, we discuss the datasets we use, machine learning classifiers, and the framework's experimental settings we consider later in the paper.

4.1 Datasets

In our experiments, we consider four publicly available datasets that are a representative sample of commonly encountered scenarios and one simulated traces dataset.

DPAcontest v4 Dataset The 4th version provides measurements of a masked AES software implementation [35] (denoted as DPAv4 in this paper). As the mask is known, one can easily turn it into an unprotected scenario. This is a software implementation, and the most leaking operation is the processing of

the S-box operation, where we attack the first round. Accordingly, the leakage model changes to:

$$Y(k^*) = \texttt{Sbox}[P_{b_1} \oplus k^*] \oplus \underbrace{M}_{\text{known mask}}, \tag{10}$$

where P_{b_1} is a plaintext byte and we choose $b_1 = 1$. The SNR has a maximum value of 5.8577. SNR (signal-to-noise ratio) is defined as $\frac{var(signal)}{var(noise)} = \frac{var(y(t,k^*))}{var(x-y(t,k^*))}$. This dataset is available at `http://www.dpacontest.org/v4/`.

AES_HD Dataset This dataset is chosen to target an unprotected implementation of AES-128. The core of AES-128 was written in VHDL in a round-based architecture, taking 11 clock cycles for each encryption. A UART module is wrapped around the core to enable external communication. The module is designed to allow accelerated measurements in order to avoid any DC shift due to environmental variation over prolonged measurements. The total area footprint of the design contains 1 850 LUT, and 742 flip-flops. Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board was used to implement the design. Side-channel traces were measured using a high sensitivity near-field EM probe, which was placed over a decoupling capacitor on the power line. Measurements were sampled on the Teledyne LeCroy Waverunner 610zi oscilloscope. A commonly used HD leakage model, when attacking the last round of an unprotected hardware implementation, is the register writing in the last round [35], i.e.,

$$Y(k^*) = HW(\underbrace{\texttt{Sbox}^{-1}[C_{b_1} \oplus k^*]}_{\text{previous register value}} \oplus \underbrace{C_{b_2}}_{\text{ciphertext byte}}), \tag{11}$$

where C_{b_1} and C_{b_2} are two ciphertext bytes, and the relation between b_1 and b_2 is given through the inverse ShiftRows operation of AES. $b_1 = 12$ was chosen, which resulted in $b_2 = 8$, as it is one of the easiest bytes to attack. The obtained measurements that form the dataset are relatively noisy and the resulting model-based SNR has a maximum value of 0.0096. In total, there are 500 000 traces corresponding to 500 000 randomly generated plaintexts, each trace with 1 250 features. Since this implementation leaks in the HD model, we denote it as AES_HD. This dataset is available at `https://github.com/AESHD/AES_HD_Dataset`.

Random Delay Dataset As our third use case, we use an actual protected implementation (we denote it as AES_RD). Our target is a software implementation of AES on an 8-bit Atmel AVR microcontroller with implemented random delay countermeasure as described by Coron and Kyzhvatov in [36]. We mounted our attacks against the first AES key byte by targeting the first S-box operation. The dataset consists of 50 000 traces of 3 500 features each. For this dataset, the SNR has a maximum value of 0.0556. This dataset is available at `https://github.com/ikizhvatov/randomdelays-traces`.

ASCAD Dataset The last publicly available dataset we used to test our framework is the ASCAD database [37]. The target platform is an 8-bit AVR microcontroller (ATmega8515) running a masked AES-128 implementation, and measurements are made using electromagnetic emanation. The dataset provides 60 000 traces, where originally 50 000 traces were used for profiling/training and 10 000 for testing. We use the raw traces and use the pre-selected window of 700 relevant samples per trace corresponding to masked S-box for $i = 3$. Interested readers can find more information about this dataset in [37]. This dataset is available at `https://github.com/ANSSI-FR/ASCAD`. 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. The SNR for the ASCAD dataset is ≈ 0.8 under the assumption we know the mask while it is almost 0 with the unknown mask.

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. [38]. 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 (that will be added as described in Section 5.1).

4.2 Figures of Merit

We consider three standard metrics when conducting SCA: accuracy, success rate, and guessing entropy. The accuracy is defined as:

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}. \tag{12}$$

TP refers to true positive (correctly classified positive), TN to true negative (correctly classified negative), FP to false positive (falsely classified positive), and FN to false negative (falsely classified negative) instances. TP, TN, FP, and FN are well-defined for hypothesis testing and binary classification problems. In the multi-class classification, they are defined in one class–vs–all other classes manner and are calculated from the confusion matrix.

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 [39]. In particular, let us assume, given Q amount of samples 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.

The success rate is defined as the average empirical probability that g_1 is equal to the secret key k_a^*. The guessing entropy is the average position of k_a^* in $\mathbf{g}$. As SCA metrics, we report the number of traces needed to reach a success rate SR of 0.9 or guessing entropy $GE < 10$.

4.3 Profiled methods – Classifiers

It is not a trivial task to select the best classifier for the given problem. Still, some classifiers can be regarded as a usual choice when a highly accurate classification is sought [40]. In order to provide relevant experiments, we select several classifiers that are a common choice in SCA, as discussed in Section 2.

Naive Bayes The Naive Bayes (NB) classifier [41] is based on the Bayesian rule but is labeled "Naive" as it works under a simplifying assumption that the predictor features (measurements) are mutually independent among the features, given the class value. 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. NB classifier outputs posterior probabilities as a result of the classification procedure [41]. The Bayes' formula is used to compute the posterior probability of each class value y given the vector of N observed feature values x.

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. Radial kernel-based SVM used here has two significant tuning parameters: the cost of the margin C and the kernel parameter γ. The scikit-learn implementation we use considers libsvm's C-SVC classifier that implements SMO-type algorithm [42]. 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 [43]. 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, and the trees are built without pruning, RF is a parametric method concerning the number of trees in the forest. RF is a stochastic method because of its two sources of randomness: bootstrap sampling and attribute selection at node splitting. Commonly, the most important parameter to tune is the number of trees I (note, we do not limit the tree size.)

Template Attack The template attack (TA) relies on the Bayes theorem and considers the features as dependent. 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 that is 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 that version of the attack.

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 [44]. We investigate the behavior of MLP with various activation functions, solvers, number of layers, and the number of nodes.

Convolutional Neural Networks Convolutional neural networks (CNNs) commonly consist of three types of layers: convolutional layers, pooling layers, and fully-connected layers. 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 downsampling 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 50 most important features, as commonly done in related works. To select those features, we use the Pearson correlation coefficient [45]:

$$Pearson(x, y) = \frac{\sum_{i=1}^{N}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}. \tag{13}$$

For convolutional neural networks, we do not conduct any feature selection. Additionally, we run experiments with MLP without any feature selection (denoted DLMLP) as related works showed it to be a powerful attack option, especially when considering masked implementations.

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 measurements with the added noise.

For TA and NB, there are no parameters to tune. For SVM, we conduct a grid search for $C = [0.001, 0.01, 0.1, 1]$ and $\gamma = [0.001, 0.01, 0.1, 1]$. For RF, we experiment with a different number of trees $I = [10, 50, 100, 200, 500, 1\,000]$. For MLP when using feature selection, we investigate activation functions *tanh* and $ReLU$, solvers $lbfgs$ and *adam*, and number of layers/nodes $[(50, 10, 50)$,$(50, 30, 20, 50)$, $(50, 25, 10, 25, 50)]$. For RF, MLP, and SVM, we use 5-fold cross-validation.

Selected Hyperparameters For SVM, we select $C = 1$ and $\gamma = 1$. For RF, we use 200 trees, and for MLP with feature selection *tanh* activation function, *adam* solver, and $(50, 30, 20, 50)$ configuration of layers/nodes.

For DLMLP (MLP without feature selection), we use six hidden layers with 200 neurons per layer, $ReLU$ activation function, $RMSprop$ optimizer, and the learning rate equal to 0.00001. This architecture is based on [37].

For CNN, we use different architectures based on [9]. 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 DLMLP 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. Note, while these hyperparameters reached the best attack performance, we do not claim they are the optimal ones.

Profiling Set Size We consider a setting with 20 000 measurements for the profiling phase, and 20 000 measurements for the attack phase. While the profil-

ing set is usually larger than the attack set, we opted to increase the size of the attack set to evaluate the influence of perturbation in the attack phase.

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. Next, we select to work with the noise α equal to 0.005 that goes in the range $[-\alpha \cdot f, \alpha \cdot f]$, where f denotes the factor going in the range $[1, 50]$. By doing so, we will evaluate noise levels going up to 25% of the signal, which would capture most of the realistic settings (more noise is, of course, possible, but then we reach setting where the trained model is very far from the test data). More precisely, we can consider our scenario as working with 50 different noise intensities. Recall, the noise we inject could arise from various sources in practical applications, for example:

- variations between the profiling and attacking device [10],
- 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 simulated measurements to validate its correctness. Next, we use the framework to evaluate four publicly available datasets. We consider three standard figures of merit, six classifiers (two versions of MLP), two leakage models, and settings with and without feature selection in our experiments. Additionally, we explore the influence of the profiling set size and hyperparameter tuning. In total, we conduct more than 500 experiments to evaluate our framework objectively. Note, we give results where noise is added to the attack phase measurements. This simulates the behavior one would encounter in portability but also from the influence of the countermeasure or environment (as adding measurements to the training phase would cause profiling methods to model such data, which would make the model less reliable but the difference between the train and test data would remain "same"). Additionally, the goal of this evaluation is not to find the best performing method but to confirm the validity of our framework and obtain information about the robustness of profiled methods.

In Algorithm 1, we give the pseudocode of the procedure we follow in order to assess the robustness of a certain classifier against perturbations in the attacking environment. Since $n = 150$ and the number of intensities f equals 50, this means that for each scenario, we need to run the attack phase 7 500 times, where for each of those times, we run guessing entropy 50 times to obtain statistically relevant data.

Algorithm 1 Algorithm to solve the probabilistic robustness evaluation problem.

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 Measurements

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 use the concept of stylized facts, which is a generalization that summarizes data. More precisely, we examine the behavior of the framework with the simulation data and compare it with the real data. If the simulation data behaves in the same manner as the real data, then we can assume that the framework can indeed be used with real data. Naturally, this approach must be able to handle certain inaccuracies as simulation data is far from a perfect representation of the real data. Here, we use the simulated measurements where we add Gaussian noise. This dataset can then be compared with DPAv4, as both datasets do not have countermeasures and have (relatively) small amount of noise. Additionally, we compare the simulated measurements with the added random delay countermeasure and the AES_RD dataset.

Simulated Measurements Data Augmentation and Noise Addition Recall, our dataset with simulated measurements contains 256 traces, one for each possible S-box output value and 5 000 features. To match our other datasets, we select 50 features that correspond to the highest correlation between the measurements and the S-box output. Next, we need to extend this dataset to allow for profiled attacks to build reliable models and to add noise. For this, we use two noise settings: Gaussian noise and Gaussian noise plus random delay.

To mimic real (noisy) data, we add noise to the simulated measurements in the following way:

$$X' = X + N. \tag{14}$$

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 for 15 000 times to create 15 000 noisy traces with $\sigma = (0, 0.5]$. Note, our final dataset contains approximately the same number of measurements for each S-box output value.

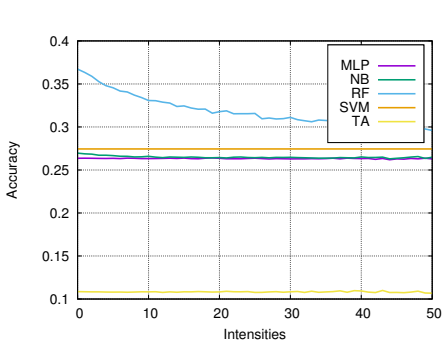
Additionally, we repeat the procedure of adding noise, but now, we add a random delay. For this we randomly select five indices within the features to add an artificially created measurement point. In particular, we randomly select $i \in [1, 49]$ and create

$$x_{new} = \frac{x_i + x_{i+1}}{2}, \tag{15}$$

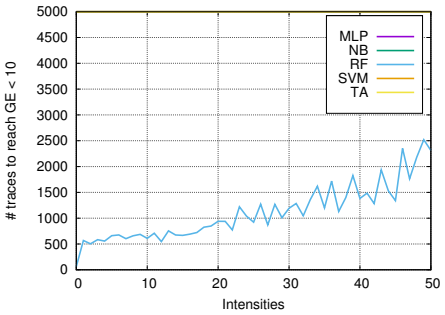
where x_{new} will be placed between x_i and x_{i+1}. The remaining setting stays the same, and we create 15 000 traces where we keep the number of measurements per class the same in the final dataset. We divide the simulated dataset into 10 000 for training and 5 000 for testing.

For these experiments, we do not consider deep learning as we conducted the feature selection step. In Figure 2, we display results for simulations with Gaussian noise and DPAv4 dataset. Figure 2a shows the influence of perturbations when using accuracy as the figure of merit. We can observe that most of the methods do not suffer from the added perturbation, but the accuracy is around 26%, which indicate that the classifier did not actually learn to classify but simply assigns all measurements to the Hamming weight 4 (see [7] for more explanation about the imbalancedness problem). The only classifier that works is RF, and the effect of perturbation is clear as accuracy drops from 37% to 29%. We observe that even when considering the setting with the largest intensity, RF still does more than simply classifying all measurements into HW 4 (as HW 4 is the most occurring class). This indicates that from all considered classifiers, RF is the only one that works in noisy settings and even adapts to more noise in the attack phase. Similar observations can be seen when considering guessing entropy. RF starts by easily breaking the target when there is no added perturbation, and then its behavior gradually decreases where we see that we need around 25 times more traces when the intensity equals 50 to reach a similar performance level.

In Figure 3, we investigate the behavior of profiled methods when dealing with the random delay countermeasure added to the simulated dataset. We depict here the intermediate value leakage model results, so there are no obvious results indicating imbalancedness, but we observe very low values for accuracy. More precisely, in Figure 3a, we observe three characteristic behaviors. First, SVM does not change the behavior at all. While the accuracy is low, the classifier is stable in the presence of noise, and as such, there is no influence of perturbation (which, of course, does not mean that the classifier makes good predictions). Second, RF starts as the best performing method but suffers from the influence of perturbation and ends with an accuracy below the one from SVM. Third, all other classifiers (TA is somewhat worse than MLP and NB) start with marginally better behavior than random guessing, and with added noise, their performance slightly reduces to random guessing. In Figure 3b, we depict results for guessing entropy. Interestingly, we observe that only NB and



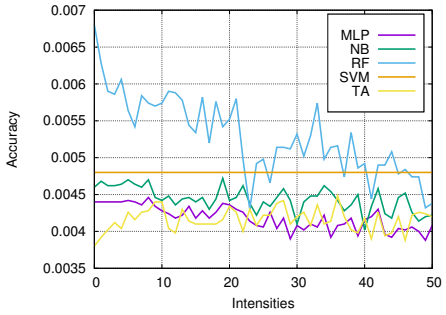
(a) Simulated measurements with Gaussian noise, HW leakage model, accuracy.



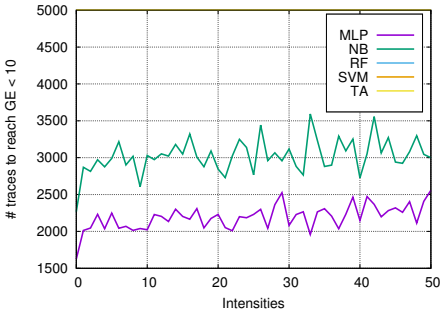
(b) Simulated measurements with Gaussian noise, HW leakage model, guessing entropy.

Fig. 2: Simulated measurements with added Gaussian noise.

MLP manage to break the target with up to 5 000 attack traces. Their performance slightly decreases with the increase in the intensity level, which is in line with their behavior for accuracy. Note how RF cannot break target even with 5 000 attack traces, which is once more confirmation how accuracy can be misleading metric, even in the intermediate value leakage model (this is well established for the HW leakage model).



(a) Simulated measurements with random delay, intermediate value leakage model, accuracy.



(b) Simulated measurements with random delay, intermediate value leakage model, guessing entropy.

Fig. 3: Simulated measurements with added random delays.

5.2 Publicly Available Datasets and Framework Evaluation

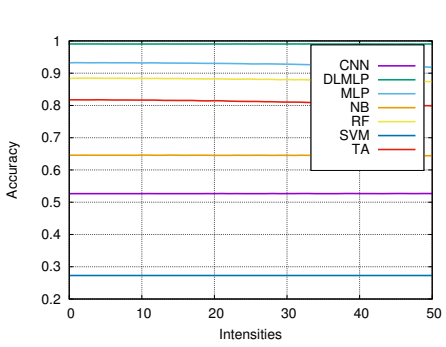
Next, we present and briefly discuss several characteristic scenarios for publicly available datasets. In Figure 4, we depict results for DPAv4 and AES_RD datasets. First, in Figure 4a, we see how the accuracy is high, and for all classifiers, except SVM, this dataset should be easy to classify. Such observations are confirmed from guessing entropy results (Figure 4b), as, for most of the classifiers, an increase in the perturbation intensity does not bring any decrease in the attack performance. The exception is SVM, where we observe how the performance decreases and already for intensity equal to 25, 20 000 attack traces is not enough to break the target. Finally, TA cannot break the target regardless of the perturbation level.

For AES_RD, we observe that most of the classifiers perform on the level of random guessing (Figure 4c) except CNN, which has significantly higher accuracy. As we know that CNNs are well-equipped to handle the random delay countermeasure due to their spatial invariance [5, 8], this result can serve as a strong indication that CNN should be able to break the target rather easily. This is confirmed in Figure 4d, where we observe CNN breaking the target easily, and no amount of added perturbation makes the attack harder. An interesting result appears for NB, as it also breaks the target (granted, with orders of magnitude more traces than CNN). We postulate this happens due to its generative nature and independence of features assumption.

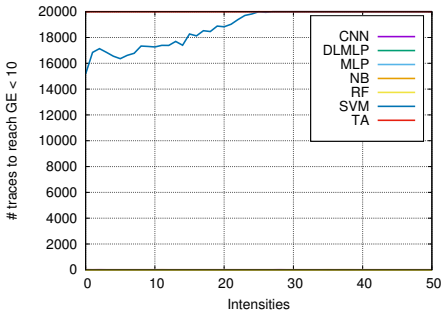
In Figure 5, we depict results for AES_HD and ASCAD. First, for AES_HD and accuracy, observe how the accuracy has erratic behavior when adding perturbation. This happens because AES_HD is already very noisy dataset to start with, and then, any change in the noise level, brings unstable behavior as many predictions are equally likely. The behavior for guessing entropy shows that while accuracy can be affected (as each measurement is treated separately), the effect of cumulative probabilities stabilizes the results for guessing entropy. Still, several methods cannot break the target even without added noise, while NB is most stable (as a consequence of feature independence assumption, the noise has less effect).

Finally, we show results for the ASCAD dataset. From the accuracy perspective, several methods can be considered as performing well, but still under the influence of added noise (RF, CNN, DLMLP). On the other hand, SVM conducts random guessing regardless of the added noise. Finally, NB, MLP, and TA classifiers show more oscillations in their behavior due to added noise. On the guessing entropy side, we observe that classifiers do not suffer much when adding more perturbation. CNN and DLMLP perform very well, while TA cannot break the implementation even with 20 000 traces and regardless of the perturbation level. Other classifiers perform reasonably well (similar as in related works), where RF has most problems with handling high perturbation levels.

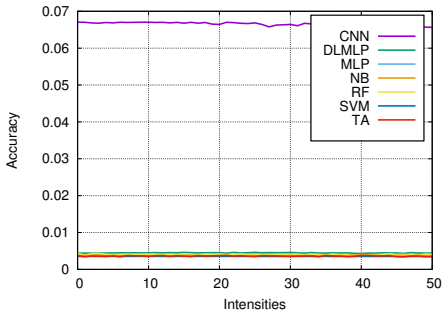
In Appendix A and B, we depict additional results where we show the influence of the suboptimal hyperparameter choice and smaller dataset size, respectively. We note that these experiments are in line with already presented results



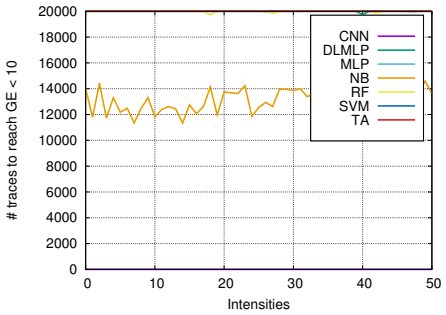
(a) DPAv4, HW leakage model, accuracy.



(b) DPAv4, HW leakage model, guessing entropy.



(c) AES_RD, intermediate value leakage model, accuracy.



(d) AES_RD, intermediate value leakage model, guessing entropy.

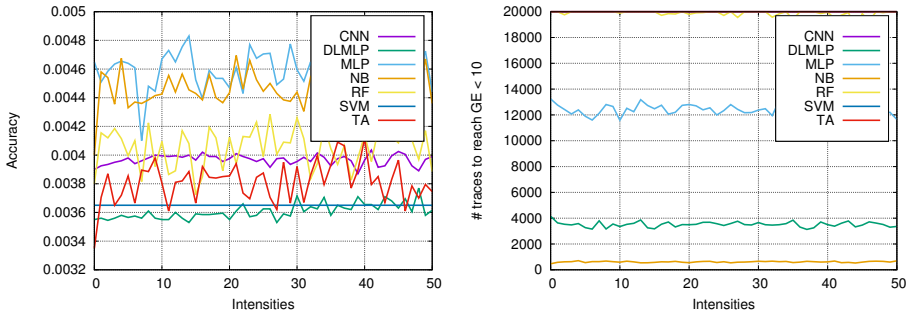
Fig. 4: Results for the DPAv4 and AES_HD datasets.

and indicate that we can model various effects with noise, but also, that our framework can be used in various settings.

6 General Observations and Remarks

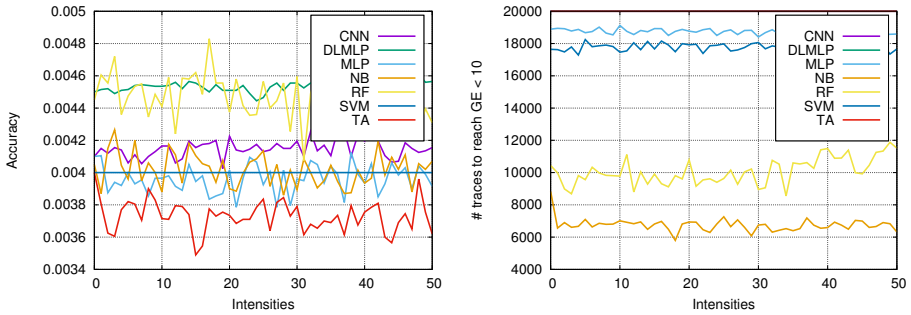
There are four main types of behavior one can expect when modeling the system with perturbations. We give the list in the order from the most preferred to the least preferred setting:

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, already a small level of perturbation results in a significant performance drop or very erratic behavior, i.e., jumps between good and poor performance.



(a) AES_HD, intermediate value leakage model, accuracy.

(b) AES_HD, HW leakage model, guessing entropy.



(c) ASCAD, intermediate value leakage model, accuracy.

(d) ASCAD, HW leakage model, guessing entropy.

Fig. 5: Results for the AES_HD and ASCAD datasets.

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, there are several subtypes one can recognize by combining the traits of the basic behaviors. Next, we give several remarks that clarify the potential of our framework. Note that we give this in a classifier-agnostic way.

1. Our framework indicates what settings/classifiers are most resilient to perturbation, which can then be used to select the preferred attack. Note how this changes the research question from the best-performing method (which

can be a very volatile concept that depends on many factors) into the most stable method.

2. If the dataset is easy to attack, adding perturbations does not significantly influence the performance. Indeed, our results indicate that the classifiers then behave as "Resilient" or "Stable".
3. If the dataset is difficult to attack due to noise, then adding perturbation commonly results in "Unstable" behavior.
4. Accuracy is the least resilient metric for perturbations as we consider each measurement separately. More resilience to perturbation happens with guessing entropy and success rate due to the cumulative probabilities approach. Additionally, we observe an interesting trade-off. From the performance side without added perturbation, guessing entropy is more powerful as it considers all key guesses and not only the best one. When adding perturbation, the success rate is more stable as it considers only the best guess, so it avoids any instability stemming from multiple key guesses that are (approximately) equally likely.
5. When considering the random delay and masking countermeasures, we see that the classifiers can work easier with random delay and perturbation than with masking and perturbation. This indicates that masking benefits more from added noise than random delay, which is especially interesting for portability settings where there are always differences between the profiling and attack device.
6. 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.

Besides these observations, we make two more remarks:

- The generative nature of TA and NB gives those methods more resilience to perturbations, which is especially pronounced for NB, as there, we also work under feature independence assumption.
- Our results indicate that accuracy is not a reliable measure for the intermediate value leakage model for difficult datasets. This is well-known for the Hamming weight model [7].

When considering the hyperparameter tuning and perturbations, we see that tuning is beneficial if perturbations are not too large. If we expect to work in scenarios with large perturbations, it could be better just to conduct a coarse-grained tuning since the performance will not suffer due to the influence of noise. A similar effect can be observed from decreasing the dataset size where for smaller levels of noise, there is not much influence, but larger perturbations affect smaller datasets more (as the learned models are less reliable).

7 Conclusions

In this paper, we concentrate on the problem of profiled 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 profiled 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 where we can 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 the question of which classifier is the best for some scenarios. We give robustness analysis for all standard figures of merit in SCA: accuracy, success rate, and guessing entropy.

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

We note that our framework is demanding: to conduct a proper analysis, the Chernoff bound requires a large number of samples, 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 intensities. Another option, which we plan to explore in the future work is to use the Chernoff-Okamoto bound that is tighter than the Chernoff bound. Since it applies when $p \leq 0.5$ only, it remains to be seen how useful this bound can be in the SCA domain.

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. 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. Cryptology ePrint Archive, Report 2019/661 (2019) `https://eprint.iacr.org/2019/661`.
11. Valiant, L.: Probably Approximately Correct: Nature's Algorithms for Learning and Prospering in a Complex World. Basic Books, Inc., New York, NY, USA (2013)
12. 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
13. 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
14. 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
15. Genkin, D., Shamir, A., Tromer, E.: Acoustic cryptanalysis. Journal of Cryptology **30**(2) (Apr 2017) 392–443
16. 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.
17. 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)
18. 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
19. 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.

20. 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
21. 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
22. 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.
23. Picek, S., Heuser, A., Guilley, S.: Template attack versus Bayes classifier. Journal of Cryptographic Engineering **7**(4) (Nov 2017) 343–351
24. 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
25. Heuser, A., Picek, S., Guilley, S., Mentens, N.: Lightweight ciphers and their side-channel resilience. IEEE Transactions on Computers **PP**(99) (2017) 1–1
26. 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
27. 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`.
28. 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`.
29. 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
30. Picek, S., Heuser, A., Guilley, S.: Profiling side-channel analysis in the restricted attacker framework. Cryptology ePrint Archive, Report 2019/168 (2019) `https://eprint.iacr.org/2019/168`.
31. Alippi, C.: Intelligence for Embedded Systems: A Methodological Approach. Springer Publishing Company, Incorporated (2014)
32. Rudin, W.: Real and Complex Analysis, 3rd Ed. McGraw-Hill, Inc., New York, NY, USA (1987)
33. 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
34. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association **58**(301) (1963) 13–30
35. TELECOM ParisTech SEN research group: DPA Contest (4^{th} edition) (2013–2014) `http://www.DPAcontest.org/v4/`.
36. 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
37. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptology ePrint Archive **2018** (2018) 53

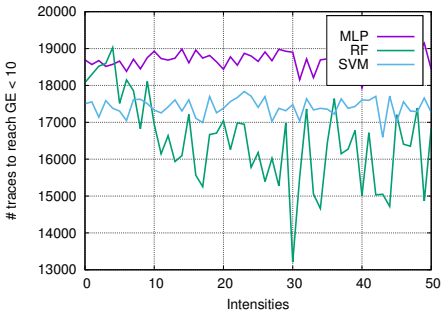
38. Regazzoni, F., Cevrero, A., Standaert, F.X., Badel, S., Kluter, T., Brisk, P., Leblebici, Y., Ienne, P.: A design flow and evaluation framework for DPA-resistant instruction set extensions. In Clavier, C., Gaj, K., eds.: CHES09. Volume 5747 of LNCS. Springer (September 2009) 205–19
39. Standaert, 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
40. 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
41. Friedman, N., Geiger, D., Goldszmidt, M.: Bayesian Network Classifiers. Machine Learning **29**(2) (1997) 131–163
42. 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
43. Breiman, L.: Random Forests. Machine Learning **45**(1) (2001) 5–32
44. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016) `http://www.deeplearningbook.org`.
45. James, G., Witten, D., Hastie, T., Tibsihrani, R.: An Introduction to Statistical Learning. Springer Texts in Statistics. Springer New York Heidelbert Dordrecht London (2001)

A Influence of Hyperparameter Change

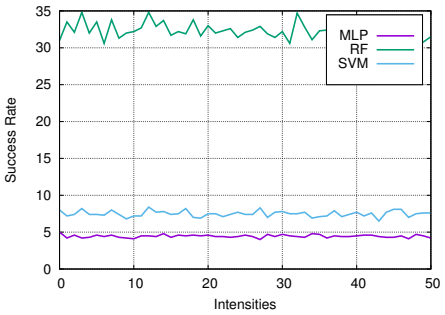
We evaluate the attack performance with changed hyperparameters to examine the behavior of our framework when using suboptimal profiling methods. More precisely, we select the second-best performing hyperparameter combinations for several classifiers. Here, we use SVM with $C = 0.001$ and $\gamma = 0.001$, RF with 100 trees, and MLP with feature selection and ReLU activation function, *adam* solver, and $(50, 30, 50)$ configuration of layers/nodes. Figures 6a and 6b show results for ASCAD with guessing entropy, and DPAv4 with success rate, respectively. For ASCAD, we see that changed hyperparameters only slightly decreased performance, but now, adding noise makes the results much less stable. For DPAv4, we depict results for success rate, and we observe stable behavior in the presence of added noise. The success rate can be less influenced by added noise than guessing entropy since it uses cumulative probabilities (which adds stability) but does not consider less likely guesses (where added noise more easily causes miss-classification).

B Influence of the Dataset Size Change

Finally, we investigate the influence of dataset sizes. Here, we use 10 000 measurements for the profiling phase, and 5 000 for the attack phase. Figure 7 depicts several characteristic behaviors. Observe how smaller dataset size makes the model less reliable and more prone to the influence of noise. Indeed, if having



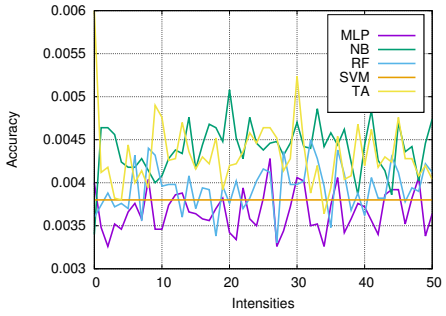
(a) Suboptimal hyperparameters for ASCAD, HW leakage model, guessing entropy.



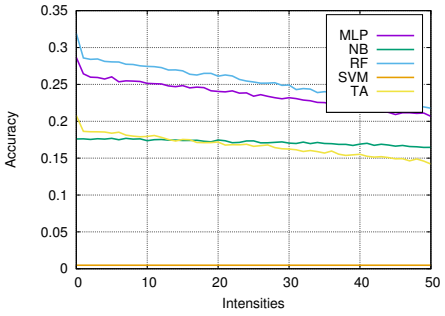
(b) Suboptimal hyperparameters for DPAv4, HW leakage model, success rate.

Fig. 6: Changes in hyperparameters.

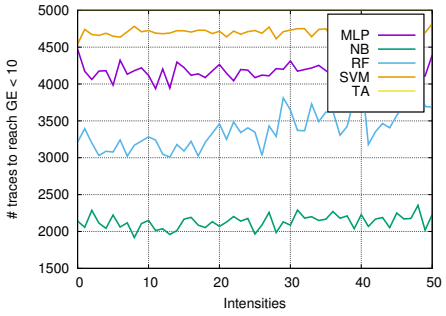
already noisy datasets like AES_HD (Figure 7a), adding perturbations causes erratic behavior, while if the dataset is easy to break (like DPAv4, Figure 7b), adding more noise reduces the performance. The erratic behavior is more likely to be observed with accuracy as there is no compensation in the form of cumulative probabilities, but even for guessing entropy, we observe a slight but steady decrease in the attack performance when adding more perturbations.



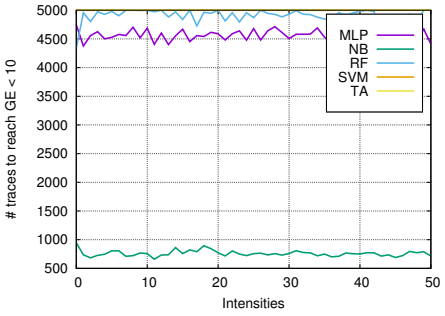
(a) Smaller dataset size, AES_HD, intermediate value leakage model, accuracy.



(b) Smaller dataset size, DPAv4, intermediate value leakage model, accuracy.



(c) Smaller dataset size, ASCAD, HW leakage model, guessing entropy.



(d) Smaller dataset size, AES_HD, HW leakage model, guessing entropy.

Fig. 7: Changes in the dataset size.