Hiding in Plain Sight: Non-profiling Deep Learning-based Side-channel Analysis with Plaintext/Ciphertext

Lichao Wu[1], Guilherme Perin[2], and Stjepan Picek[3]

[1] Delft University of Technology, The Netherlands
[2] Leiden University, The Netherlands
[3] Radboud University, The Netherlands

Abstract. Deep learning-based profiling side-channel analysis is widely adopted in academia and industry thanks to the ability to reveal secrets protected with countermeasures. To leverage its capability, the adversary needs to have access to a clone of an attack device to obtain the profiling measurements. Moreover, the adversary needs to know secret information to label these measurements. Non-profiling attacks avoid those constraints by not relying on secret information to label data but rather by trying all key guesses and taking the most successful one. Deep learning approaches also form the basis of several non-profiling attacks. Unfortunately, such approaches suffer from high computational complexity and low generality when applied in practice.

This paper proposes a novel non-profiling deep learning-based side-channel analysis technique. Our approach relies on the fact that there is (commonly) a bijective relationship between known information, such as plaintext and ciphertext, and secret information. We use this fact to label the leakage measurement with the known information and then mount attacks. Our results show that we reach at least 3× better attack performance with negligible computational effort than existing non-profiling methods. Moreover, our non-profiling approach rivals the performance of state-of-the-art deep learning-based profiling attacks.

Keywords: Non-profiling side-channel analysis · Deep learning · Data augmentation · Plaintext · Ciphertext.

1 Introduction

Side-channel analysis (SCA) on symmetric-key cryptography is commonly divided into non-profiling and profiling attacks. Non-profiling attacks assume the adversary has no access to a clone of a device to be attacked. The adversary obtains measurements containing secret information processing and runs a statistical analysis to obtain the best guess. On the other hand, profiling attacks assume the adversary has full control of a device clone to be attacked. With that clone, the adversary characterizes the side-channel behavior and then uses this knowledge to reveal secret information from the device under attack. Recent

years witnessed tremendous progress with profiling attacks where deep learning plays the main role. Indeed, already the first paper using convolutional neural networks for SCA in 2016 reported an "overwhelming advantage" [28]. Seven years and more than 200 papers later, the advantage is even clearer as we can break various protected targets with only a single measurement [31]. Unfortunately, deep learning in the context of non-profiling SCA received much less attention, and the current results are not impressive. In the rest of the paper, for brevity, we will use the acronym DLSCA for deep learning-based profiling SCA and NP-DLSCA for deep learning-based non-profiling SCA.

B. Timon proposed the first deep learning-based NP-DLSCA in 2019, called Differential Deep Learning Algorithm (DDLA). While the author reported successful attacks on several targets, there were also some considerable issues. First, the attack has huge computational complexity, necessitating a neural network for every possible key guess. Attacking even a single key byte of AES results in 256 neural networks. Second, similar to other partition-based attacks [20,5], the leakage function should be carefully selected to break the bijectivity between key guesses and labels, making the performance less attractive than DLSCA, where the Identity leakage model represents a common choice. Third, the neural networks used in the attack still require tuning, making the computational effort even more significant. Naturally, there has been some progress in the last few years, but NP-DLSCAs commonly aim to fix the above-mentioned issues with DDLA, with the reported results far from excellent (and not even close to DLSCA).

The ability to run powerful side-channel analysis is extremely relevant as this is the common way to assess the security of an implementation and gain certification, making it important for the industry. While we discussed that DLSCA achieves excellent attack performance and is well-explored in academia, the industry perspective is somewhat different. There are still no official standards on what kind of DLSCA to run, making non-profiling attacks the core tool. Indeed, correlation power analysis is still the workhorse of practical side-channel evaluations [14,9].

After the first work on NP-DLSCA, several more works explore that direction.[4] As already indicated, most of those works consider improving DDLA concerning one or more issues discussed above. Furthermore, while the reported results are commonly good, the resulting algorithms cannot be considered practical or approach the performance of DLSCA. More recently, in 2022, several authors proposed radically different approaches for NP-DLSCA [13,17]. Those approaches are based on multi-output classification and multi-output regression (MOR). While further improving over the state-of-the-art DDLA variants, the proposed approaches still have significant computational complexity, and more importantly, the attack performance is not promising.

In this paper, we address the issues with the NP-DLSCA and propose a new approach that outperforms previously discussed non-profiling attacks (deep

[4] More precisely, we found 14 papers. While not negligible, it is far from more than 200 papers on DLSCA.

learning-based ones and "classical" correlation power analysis (CPA)). The core of our approach lies in realizing it is possible to profile in non-profiling SCA by removing the need for key-related intermediate data. Commonly, an adversary only knows plaintexts or ciphertexts. Researchers tend to believe they are not ideal profiling targets as 1) plaintexts/ciphertexts leakages do not exist in encryption or decryption of critical assets, e.g., hardware root key. Otherwise, an adversary could reveal these data by profiling them, and 2) knowing plaintexts/-ciphertexts does not contribute to retrieving the secret key. However, one should note that the secret key is commonly fixed in a non-profiling attack setting. Therefore, using a proper labeling function, the plaintexts/ciphertexts and the key-related intermediate data form a bijective relationship. By leveraging this bijectivity, we can characterize the leakage measurements with plaintext/ciphertext and then use the characterization results to retrieve the key. Our approach, named Plaintext Labeling Deep Learning (PLDL), is easy to tune and train and provides superior attack performance even compared to DLSCA. Our main contributions are:

1. We propose a novel approach for NP-DLSCA. We showcase that PLDL works well even if there is no specific tuning of the deep learning architecture.
2. We provide theoretical explanations of how our attack is built and why it works.
3. We provide a detailed experimental analysis showing PLDL outperforming three non-profiling SCAs: CPA, DDLA, and MOR. Moreover, we provide results comparing our results with DLSCA and exhibiting similar performance, a result not previously reported in the literature.
4. We showcase a new variant of NP-DLSCA where we consider the portability factor [4]. We train our deep learning model with one set of traces, showing we can easily obtain the secret information for other traces coming from the same distribution (device).
5. We provide an ablation study where we investigate several relevant factors for our attack: data augmentation, the number of measurements, and the number of training epochs. Our results suggest PLDL is robust to diverse settings, and data augmentation is a crucial factor in mounting powerful NP-DLSCA.

The source code is available in the anonymous Github `https://anonymous.4open.science/r/PLDL-91F3/`.

The rest of this paper is organized as follows. In Section 2, we provide the necessary background information. Section 3 discussed related works. Section 4 provides details about the threat model, our labeling procedure, and the attack scheme. In Section 5, we provide experimental results. Finally, in Section 6, we conclude the paper and discuss potential future research directions.

2 Preliminaries

This section introduces the notation we follow. Afterward, we provide relevant information about side-channel analysis and machine learning.

2.1 Notation

We use calligraphic letters like $\mathcal{X}$ to denote sets and the corresponding upper-case letters X to denote random variables and random vectors $\mathbf{X}$ over $\mathcal{X}$. The corresponding lower-case letters x and $\mathbf{x}$ denote realizations of X and $\mathbf{X}$, respectively. We use a sans serif font for functions (e.g., f).

The term k represents a key byte candidate that takes its value from the key space $\mathcal{K}$. k^* is the correct key byte or the key byte assumed to be correct by the adversary.[5]

A dataset $\mathbf{T}$ is a collection of traces $\mathbf{t_i}$, with each trace $\mathbf{t_i}$ associated with a label y_i. A complete set of labels with c classes is denoted by $\mathcal{Y} = \{y^1, y^2, \cdots, y^c\}$. In a dataset $\mathbf{T}$, each trace $\mathbf{t_i}$ is associated with a plaintext/ciphertext $\mathbf{d}_i \in \mathcal{D}$ and a key $\mathbf{k}_i$, or $k_{i,j}$ and $d_{i,j}$ when considering a partial key recovery on byte j. In this work, we consider attacking only a single key byte and, thus, omit the byte vector notation in equations. We will divide the dataset $\mathbf{T}$ into the profiling set of size N and the attack set of size Q. $\boldsymbol{\theta}$ denotes the vector of parameters to be learned in a profiling model.

2.2 Side-channel Analysis

Based on the availability of a fully-controlled cloned device, side-channel analysis (SCA) can be generally categorized into two types: profiling SCA and non-profiling SCA. Non-profiling side-channel analysis leverages the correlation between some key-related intermediate values and leakage measurements. To perform such attacks, the adversary gathers a collection of traces throughout a series of encryptions of different plaintexts. If the selected intermediate value is sufficiently correlated with leakage measurements, he can use these measurements to verify guesses for a small part of the key. In particular, for each possible value of the relevant part of the key (or the key itself), the adversary will follow a "divide-and-conquer" strategy:

1. **Divide/Partition.** The adversary divides the traces into groups according to the intermediate value predicted by the current key guess.
2. **Attack.** If each group differs noticeably from the others (the definition of 'difference' is based on the attack methods), the current key guess is likely correct.

Non-profiling attacks assume less powerful adversaries with no access to a clone of a device to be attacked, so they might require millions of measurements to obtain secret information. Common examples of such attacks are simple power analysis (SPA) and differential power analysis (DPA) [21].

Profiling side-channel attack maps a set of inputs (e.g., side-channel traces) to a set of outputs (e.g., probability vector of key hypotheses). Profiling side-channel attacks run in two phases:

[5] Note that the subkey candidates can have any number of bits being guessed. While here we assume the AES cipher scenario, the concept is algorithm-independent.

1. **Profiling phase.** The adversely builds a profiling model $\mathsf{f}_{\boldsymbol{\theta}}^{M}$, parameterized by a leakage model M and a set of learning parameters $\boldsymbol{\theta}$, to map the inputs (side-channel measurements) to the outputs (classes as obtained by evaluating the leakage model on the sensitive operation) on a set of N profiling traces. We use the notions $\mathsf{f}_{\boldsymbol{\theta}}^{M}$ and $\mathsf{f}_{\boldsymbol{\theta}}$ interchangeably.
2. **Attack phase.** The trained model processes each attack trace $\mathbf{t_i}$, outputting a vector of probabilities $\mathbf{p}_j$, representing the probability of the associated leakage value or label j. Based on this vector of probabilities, the adversary decides on the best key candidate, as discussed in Section 2.6.

If the adversary builds a good model, only a few measurements from the device under attack could suffice to break it. Examples of attacks are the template attack [7], stochastic models [36], and machine learning-based attacks [18,28].

2.3 Machine Learning

The capability of a system to acquire its knowledge by extracting patterns from raw data is known as machine learning [15]. Commonly, it is used to extract knowledge from given data and learn how to perform complicated tasks that conventional algorithms have difficulty handling, such as medical applications [34], email spam filtering [11], speech recognition [29], and computer vision [37]. Machine learning algorithms can be divided into several categories based on their learning style.

Supervised learning algorithms have access to labeled training data. Based on the data and labeling, such algorithms train a model f to predict labels on previously unseen data. Most of the supervised learning methods follow the Empirical Risk Minimization (ERM) framework, where the model parameters $\boldsymbol{\theta}$ are obtained by solving the optimization problem:

$$\underset{\boldsymbol{\theta}}{\arg\min}\ \frac{1}{N}\sum_{i}^{N}\mathsf{L}(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t_i}), y_i), \tag{1}$$

where L is the loss function. Supervised machine learning happens in two phases: training and testing. Commonly considered problems are classification (where the output is discrete) and regression (where the output is continuous).

Semi-supervised learning follows a similar paradigm as supervised learning, but we assume that the number of labeled examples is limited.

Unsupervised learning works under a setup with no access to labeled data. Then, a model is prepared by deducing structures in the input data. Common problems include clustering and dimensionality reduction.

2.4 Machine Learning in the Context of SCA

There is an intuitive mapping between machine learning algorithms based on the learning style and side-channel analysis. Still, some subtleties necessitate the need for precise definitions, especially in the context of non-profiling attacks.

Note that the discussion given next considers SCA on symmetric key cryptography only.[6] Moreover, we limit our attention to SCA on block ciphers (AES) in this work.

Definition 1. *Profiling attacks rely on a labeling function that uses the secret information from a device controlled by the adversary. The adversary leverages the measurements from the controlled device to build a model and uses that model to guess the key for measurements that are not labeled due to not knowing the secret information.*

Remark 1. Since there is a labeling function with secret information, supervised machine learning belongs to the profiling attack setting. This category also includes semi-supervised learning, as there are no differences in how the attack is done but only on the assumption of how many labeled measurements can be acquired. Such attacks happen in two phases, also called two-stage attacks.

Definition 2. *Non-profiling attacks rely on a labeling function without knowledge of the secret information. The adversary uses the measurements from the attacked device to obtain the secret information.*

Remark 2. Since there is still a labeling function (while not using secret information), supervised machine learning (as well as semi-supervised) also belongs to the non-profiling attack setting.

Remark 3. If the labeling function is used on the measurements to be attacked, we consider this a "standard" non-profiling attack. It is also possible to use the labeling function on one set of data and attack another set of data, a setting we call a portability non-profiling attack. Non-profiling attacks have one or two phases.

Remark 4. Unsupervised machine learning can be used in profiling and non-profiling attacks for feature engineering. Still, this does not change the nature of the attacks discussed above, as the attack happens after feature engineering.

2.5 Non-profiling SCA Techniques

Correlation Power Analysis – CPA. Correlation power analysis is one of the most classical non-profiling SCA techniques [5]. It is based on the Pearson correlation that measures the linear relationships between the leakage features and hypothetical labels. More specifically, we calculate the correlation coefficient vector $\mathbf{r}$ by testing all key candidates in $\mathcal{K}$ with Equation 2.

$$r_k = \frac{\sum_{i=1}^{N}(\mathbf{t_i}^k - \overline{\mathbf{t}^k})(y_i^k - \overline{y^k})}{\sqrt{\sum_{i=1}^{N}(\mathbf{t_i}^k - \overline{\mathbf{t}^k})^2 \sum_{i=1}^{N}(y_i^k - \overline{y^k})^2}},\ k \in \mathcal{K}, \tag{2}$$

[6] SCA on public-key cryptography also uses unsupervised learning.

where $\mathbf{t_i}^k$ and y_i^k are the leakage trace and the corresponding hypothetical label based on k, respectively. The averaged leakage traces and labels are represented by $\overline{\mathbf{t}^k}$ and $\overline{y^k}$. The most likely key k^* can be obtained by:

$$k^* = \arg\max \ \mathbf{r}. \tag{3}$$

Differential Deep Learning Analysis – DDLA. DDLA is a non-profiling SCA method based on deep learning, thus NP-DLSCA [38]. Generally speaking, one can consider DDLA as a deep learning version of CPA, as they both label the leakage traces based on different key guesses, then "correlate" these hypothetical labels with leakage traces. While the adversary uses Pearson correlation in CPA, DDLA relies on the empirical risk measured by a loss function L.

To perform DDLA, the adversary first repeats Equation 1 for each key candidate k to estimate $\boldsymbol{\theta}_k$; y_i in Equation 1 equals to y_i^k, which is calculated with the current key guess. Then, the most likely key (the key assumed to be correct) k^* can be distinguished by finding the one that generates the least empirical risk measured by the negative log-likelihood (NLL) loss:

$$k^* = \arg\min_k -\frac{1}{N}\sum_i^N \log(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t_i})), \ k \in \mathcal{K}, \tag{4}$$

where $\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t_i})$ represents the conditional probability of y_i^k given a input $\mathbf{t_i}$ and model parameters $\boldsymbol{\theta}_k$, denoted as $\mathbf{p}(y_i^k|\mathbf{t_i};\boldsymbol{\theta}_k)$.

Given the same training effort, only $y_i^{k^*}$ can lead to fast convergence of the empirical risk due to its correlation with leakage traces. Compared to CPA, thanks to the employment of deep learning, DDLA can break the SCA countermeasures such as Boolean masking. Meanwhile, it is more robust to noise introduced by, for instance, time jitters and random delay interrupts.

Multi-output DLSCA – MOR. Do *et al.* developed a new method called Multi-output regression DLSCA (MOR) [13] to reduce the computation efforts of DDLA. Instead of training 256 models with each model trying to correctly *classify* the hypothetical labels with probabilities, MOR employs the idea of multi-output regression, which aims to *regress* the prediction outputs to the actual label values. Specifically, a model is trained by mapping the input leakage traces to the actual values of all possible y_i^k. Like DDLA, the most likely key k^* is revealed by finding the smallest loss measured by mean squared error (MSE).

$$k^* = \arg\min_k \frac{1}{N}\sum_i^N (y_i^k - \mathsf{f}_{\boldsymbol{\theta}}^k(\mathbf{t_i}))^2, \ k \in \mathcal{K}, \tag{5}$$

where $\mathsf{f}_{\boldsymbol{\theta}}^k(\mathbf{t_i})$ denotes the prediction value of model to approximate y_i^k. Compared with DDLA, the computation effort is reduced as the adversary only trains one model.

2.6 Attack Performance Evaluation

Since the goal of an adversary is to guess the correct key k^*, he calculates a key guessing vector $\mathbf{g}$, which is a vector representing the likelihood of each key candidate k:

$$\mathbf{g} = \mathsf{sort}\left(\mathcal{L}(k)\right), k \in \mathcal{K}, \tag{6}$$

where $\mathcal{L}(k)$ varies for each attack method. As mentioned in Sections 2.2 and 2.5, CPA relies on the correlation coefficient; MOR and DDLA use loss value; profiling attack is based on probability. sort is the function sorting array elements in order of decreasing values of their probabilities. The elements in $\mathbf{g}$ represent the likelihood of the corresponding key candidate being the correct key candidate. g_0 and $g_{|\mathcal{K}|-1}$ are the first (best) and last (worst) element of $\mathbf{g}$, respectively.

In a known-key setting, the key rank is the number of (most likely) keys an adversary needs to brute force until recovering the correct key. Among various key enumeration techniques [33], one of the most straightforward methods is to try every key given its likelihood after generating a key guessing vector. In this scenario, the key rank is the position of the correct key in $\mathbf{g}$. If an attack method reaches the key rank of zero (meaning that the correct key ranks first), we calculate the required number of attack traces for this key rank.

3 Related Work

The SCA community has maintained research for more than 20 years in the context of profiling attacks. The first work on profiling attacks is done by Chari *et al.*, where the authors proposed the template attack [8]. This attack is the most powerful one from the information-theoretic perspective, but it relies on assumptions that are difficult to fulfill (unbounded number of profiling traces and noise following the Gaussian distribution) [26]. Another "classic" example of profiling attacks is the stochastic model [36]. Finally, a very active direction for profiling attacks considers machine learning techniques. The first works used simpler techniques like random forest [24] and support vector machines [18]. The results with those techniques already indicated performance surpassing the template attack or stochastic models. During the same period, there appeared examples of semi-supervised machine learning-based [25,32] and unsupervised machine learning [2] for SCA.

From 2016 and seminal work on machine learning-based SCA by Maghrebi *et al.* [28], the attention of a large part of the SCA community has shifted toward deep learning, especially multilayer perceptron (MLP) and convolutional neural networks (CNNs). Indeed, while simpler machine learning techniques rivaled the performance of the template attack, they still struggled in the presence of countermeasures and required significant effort in the feature engineering part. With deep learning, such issues became less important but at the cost of difficulties tuning the deep learning algorithms. Early works by Cagli *et al.* [6], and Kim *et al.* [19] demonstrated how convolutional neural networks can break protected

targets rather efficiently and how various regularization techniques further improve the attack performance. Zaid *et al.* [42] and Wouters *et al.* [39] proposed methodologies to design convolutional neural networks and demonstrated previously unseen attack performance on datasets protected with masking and hiding countermeasures. The attacks were further improved by Perin *et al.* by demonstrating how ensembles of neural networks can achieve much better attack performance than a single neural network [30]. Furthermore, the authors showed that even a random search of neural network architectures could provide excellent results. Wu *et al.* and Rijsdijk *et al.* approached the problem of neural network tuning from a different perspective and explored automated tuning techniques. Rijsdijk *et al.* used reinforcement learning to find well-performing and small neural network architectures [35]. While the authors reported excellent attack performance, the computational complexity of their approach is high, making it less usable in practice. To remedy this, Wu *et al.* used Bayesian optimization to tune neural networks [41]. They managed to achieve similar results but with much less computational effort.

More recently, Lu *et al.* showed how working with raw traces instead of intervals of features provides even better attack performance [27]. Still, the cost to work with such large measurements was to use big neural networks that are not trivial to tune. Finally, Perin *et al.* used resampled raw traces and small neural networks and still obtained outstanding attack performance [31]. The authors reported multiple scenarios where only a single attack trace was needed to break the target.

Already from this brief overview, it is clear that profiling attacks are actively researched and achieve excellent performance for various targets. What about NP-DLSCA?

B. Timon was the first to propose using deep learning in the non-profiling SCA context [38]. The approach (DDLA) consisted of training as many neural networks as there are key guesses and taking the one with the best result as the correct key guess. While the approach works, it is computationally expensive and requires the leakage function not to be bijective. Kuroda *et al.* conducted a follow-up investigation on DDLA considering the structure of neural networks and attack points, concluding it is better to use simple architectures and a wide range of points of interest [22]. Alipour *et al.* explored the performance of DDLA against a hiding-based AES countermeasure that utilizes correlated noise generation [1]. The authors concluded that the DDLA is less powerful than CPA in such a setting. Kwon *et al.* proposed several improvements for DDLA to 1) make the process faster and 2) change the used neural network architectures to be more performant [23]. Do *et al.* analyzed DDLA in more detail to understand how the technique behaves in more complex scenarios and how to improve the performance with different data preparation techniques and neural network architectures [12]. Hoang *et al.* introduced a non-profiling SCA technique using multi-output classification that achieved up to 30 times faster and 20% better results than DDLA [17]. Finally, Do *et al.* investigated multi-output classification (MOC) and multi-output regression (MOR) models for non-profiling

SCA [13]. The authors concluded that MOC reduces the execution time compared to DDLA. Furthermore, they showed that MOR, while slower than MOC, also works for the Identity leakage model. Both MOC and MOR outperformed DDLA by at least 25%.

Non-profiling efforts mainly went toward improving the efficiency of B. Timon's work. While the issues with the bijective leakage model and training complexity are mostly resolved, the performance improvement over the original DDLA approach is less evident, especially for complex datasets with countermeasures.

4 Plaintext/Ciphertext-based Non-profiling SCA

This section discussed the threat model we follow. Afterward, we provide information on plaintext/ciphertext labeling, plaintext distribution, our attack scheme, and differences between it and profiling attacks.

4.1 Threat Model

Our threat model is the same as conventional non-profiling SCA. An adversary has a device with a target cipher (white-box implementation) and a fixed but unknown key. The adversary could send commands to perform encryption/decryption operations. We assume the adversary can only observe the used plaintext/ciphertext but cannot control their values. The adversary has no information about the hardware implementation, the countermeasures settings, or the source code. To launch attacks, the adversary measures multiple side-channel leakages with a high-speed oscilloscope and then analyzes leakage traces with plaintexts and/or ciphertexts.

4.2 From Intermediate data to Plaintext and Ciphertext

Recall that supervised learning learns a mapping $\mathsf{f} : \mathcal{X} \rightarrow \mathcal{Y}$. In profiling SCA, the output variables are represented by sensitive operations, such as Sbox input or output of AES cipher. We denote the model trained with intermediate data as *intermediate data-based model*. If plaintexts or ciphertexts are also involved in such operations, based on Equation 1, we can specify the learning objective in Equation 7.

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i}^{N} \mathsf{L}(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t_i}), \mathsf{l}(k_i, d_i)), \tag{7}$$

where l denotes the labeling function that returns the leakage value according to a known key candidate k_i and a plaintext/ciphertext d_i. Then, the adversary estimates the conditional probability $\mathbf{p}(\mathsf{l}(k, d_i)|\mathbf{t_i}; \boldsymbol{\theta})$ given a leakage trace with the unknown key k. With the knowledge of d_i and l, the adversary selects the label value with the highest probability, and the k value can be easily retrieved [40,31,42].

However, in non-profiling SCA, since the adversary does not know the key being used, he cannot use $\mathsf{l}(k_i, d_i)$ to estimate $\boldsymbol{\theta}$. Fortunately, the key is fixed (we denote it as k) for all leakage traces following our threat model. Then, the label $\mathsf{l}(k, d_i)$ and d_i would satisfy:

$$d_i \longmapsto \mathsf{l}(k, d_i). \tag{8}$$

The bijectivity of the label $\mathsf{l}(k, d_i)$ and d_i depends on the selection of l. For instance, Sbox output is a common label function (and intermediate data) for AES attacks. In this case, d_i and $\mathsf{Sbox}(d_i \oplus k_i)$ are bijective given a fixed key k_i. If Equation 8 holds, d_i and $\mathsf{l}(k, d_i)$ can be mapped to each other with mapping functions map parameterized by k.

$$\mathsf{l}(k, d_i) = \mathsf{map}_k(d_i), \tag{9}$$

Since the adversary targets a known cipher, the mapping functions are known or can be easily calculated by knowing the correct key k^*. Then, we can rewrite Equation 7 as:

$$\boldsymbol{\theta} = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i}^{N} L(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t_i}), \mathsf{map}_{k^*}(d_i)), \tag{10}$$

where k^* denotes the (unknown) correct key. Indeed, due to the bijectivity between labels $\mathsf{l}(k, d_i)$ and d_i, the estimation of $\mathbf{p}(\mathsf{l}(k, d_i)|\mathbf{t_i}; \boldsymbol{\theta})$ is equivalent to an estimation of $\mathbf{p}(d_i|\mathbf{t_i}; \boldsymbol{\theta})$. From the perspective of model training, although map_{k^*} is unknown to the adversary, it is a deterministic function and thus does not influence the optimization of $\boldsymbol{\theta}$. Therefore, we can remove the mapping function in Equation 10 and train a model with dataset $\mathbf{T}$ labeled with plaintexts/ciphertexts only:

$$\boldsymbol{\theta}_d = \arg\min_{\boldsymbol{\theta}} \frac{1}{N} \sum_{i}^{N} L(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t_i}), d_i). \tag{11}$$

We denote the model trained with plaintexts/ciphertexts labels as plaintext (or ciphertext) -based model.[7] Since there is no leakage on d_i on leakage traces, both $\mathsf{f}_{\boldsymbol{\theta}_d}$ and $\mathsf{f}_{\boldsymbol{\theta}}$ should focus on the same features that correspond to the processing of $\mathsf{l}(k, d_i)$. If $\mathsf{f}_{\boldsymbol{\theta}_d}$ and $\mathsf{f}_{\boldsymbol{\theta}}$ generalize equally well on these features, they can be converted to each other with a similar mapping function map'_k where k equals to k^*:

$$\mathsf{f}_{\boldsymbol{\theta}_d}(\cdot) = \mathsf{map}'_k(\mathsf{f}_{\boldsymbol{\theta}}(\cdot)), \tag{12}$$

where map'_k converts the probability of the input $\mathsf{l}(k, d_i)$ to its corresponding output d_i. If the adversary correctly guesses the k as k^*, map'_{k^*} can be easily calculated with the knowledge of cipher implementation.

Equation 12 represents the core idea of the proposed method, which is also visualized in Figure 1. The goal of an adversary is to find a map'_k that maps $\mathsf{f}_{\boldsymbol{\theta}}$ to $\mathsf{f}_{\boldsymbol{\theta}_d}$ in the best way. If $\mathsf{f}_{\boldsymbol{\theta}}$ is known, k^* can be brute-forced by trying all possible

[7] For simplicity, we denote it as *plaintext-based model*.

map'_k and finding the best match to $\mathsf{f}_{\boldsymbol{\theta}_d}$. Although the adversary cannot learn $\mathsf{f}_{\boldsymbol{\theta}}$ in a non-profiling setting due to the lack of intermediate data knowledge, in the next subsection, we propose an estimation of $\mathsf{map}'_k(\mathsf{f}_{\boldsymbol{\theta}}(\cdot))$ by calculating the plaintext or ciphertext distribution for all possible key candidates. Knowing this, an adversary can find the best key by brute-forcing the key-related distributions.

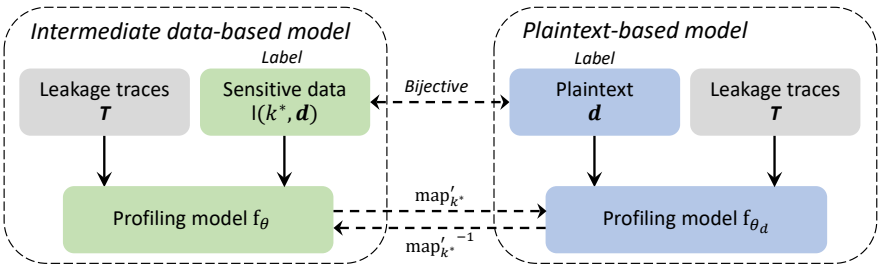


Fig. 1: The relationship between intermediate data-based model and plaintext-based model.

4.3 Plaintext/Ciphertext Distribution

In profiling SCA, as introduced in Section 2.2, a profiling model $\mathsf{f}_{\boldsymbol{\theta}}$ represents the relation between the input leakage measurement and output intermediate data processed by a leakage model M. When inputting an attack trace to $\mathsf{f}_{\boldsymbol{\theta}}$, it outputs a probability vector for all possible intermediate data:

$$\mathbf{Pr}_i(y) = \mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t_i}), \tag{13}$$

where $y = \mathsf{M}(\mathsf{I}(k, d_i)), k \in \mathcal{K}$. An adversary aims for the highest probability of the correct intermediate data y^*, while the rest of the values' probabilities and distributions are commonly ignored (e.g., NLL loss). However, we argue that these probabilities are far from arbitrary. If a profiling model is generalized well on the leakage traces, the probability of the incorrect value is closely correlated with y^*. To explain it, let us assume a dataset where the leakage features follow the Gaussian distribution with the mean value for each class *linearly* correlated with its actual labels. Then, the conditional probability of a label y_j being selected given a correct label y^* can be represented by a probability density function.

$$\mathbf{p}(y_i|y^*) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{a\|y^* - y_i\|}{\sigma^2}\right)}, \ y_i \in \mathcal{Y}, \tag{14}$$

where $\|\cdot\|$ represents the squared Euclidean distance between two variables, we denote it as label distance; a denotes the linear correlation coefficient; and σ denotes the variance of the leakage features corresponding to y^*. Since y_i is

parameterized by d_i and k, Equation 14 can be easily extended to calculate $\mathbf{p}(d_i|d^*)$ given a correct key k^*:

$$\mathbf{p}(d_i|d^*) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}\left(\frac{a\left\|\mathsf{M}(\mathsf{I}(k^*,d^*))-\mathsf{M}(\mathsf{I}(k^*,d_i))\right\|}{\sigma^2}\right)}, d_i \in \mathcal{D}. \tag{15}$$

In a non-profiling context, an adversary cannot estimate σ as he does not know k. Here, we assume it is a constant. Now, since the label distance is the only variable of $\mathbf{p}(d_i|d^*)$, we simplify Equation 15 to Equation 16 by only keeping the label distance and the negative sign:

$$\mathsf{PD}_k^{\mathsf{M}}(d^*) = -\left\|\mathsf{M}(\mathsf{I}(k,d^*)) - \mathsf{M}(\mathsf{I}(k,d_i))\right\|, \ d_i \in \mathcal{D}, \ k \in \mathcal{K}. \tag{16}$$

We denote the output vector of Equation 16 as *plaintext distribution* (could be also called *ciphertext distribution*), representing the likelihood of $d_i \in \mathcal{D}$ being selected given a d^*. Note that when it is clear from the context, we use the notations $\mathsf{PD}_k^{\mathsf{M}}(\cdot)$ and PD_k interchangeably.

4.4 Attack Scheme

Section 4.3 introduces plaintext distribution PD_k, an approximation of the likelihood of each value being selected as the correct plaintext/ciphertext. Now, if the adversary builds a profiling model $\mathsf{f}_{\boldsymbol{\theta}_d}$ by fitting the leakage traces with plaintexts/ciphertexts, the output probability vector $\mathbf{Pr}(d_i)$ and PD_k with k equals to k^* would have a higher correlation value compared with other key candidates. Following this, to retrieve the correct key k^*, the adversary would calculate PD_k with all possible keys; the one leading to the highest correlation with the prediction vector would be the most likely key (and assumed to be the correct key):

$$k^* = \arg\max_k \mathsf{corr}(\mathsf{PD}_k^{\mathsf{M}}(\mathbf{d}), \ \mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{T})), \ k \in \mathcal{K}, \tag{17}$$

where corr represents the Spearman correlation [8] [16] that evaluates the monotonic relationship with two inputs. Since the adversary knows $\mathbf{d}$, the brute force effort equals the key space size. For instance, if attacking a subkey (a single byte), the adversary must calculate PD_k 256 times to find the correct key. The entire process is illustrated in Figure 2.

Equation 17 represents the basis of the proposed attack method. However, several practical considerations must be made when performing actual attacks. First, similar to conventional DLSCA, the pre-processing of leakage measurements is mandatory to have an efficient attack. Besides normalizing the data, we notice that data augmentation is the key part that makes the proposed attack successful. Indeed, data augmentation, as a regularization technique, can prevent the profiling model from focusing on specific features and better focus on

[8] The Spearman correlation offers more numerical stability than the Pearson correlation, as it has high tolerance when the labels and leakage features are not linearly correlated.

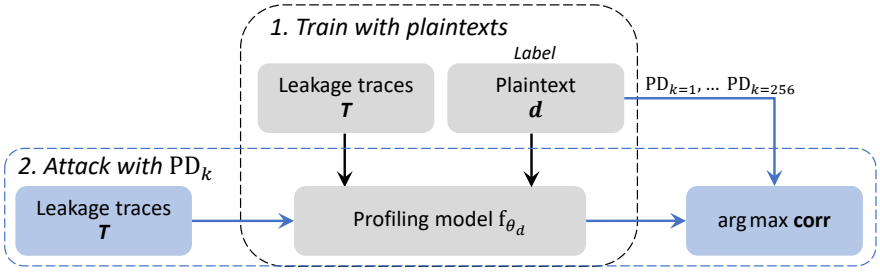


Fig. 2: Attack scheme of the Plaintext Labeling Deep Learning (PLDL).

global features. In the SCA context, since the data leakages only exist in a few features, such techniques can prevent the model from overfitting on non-relevant features. An ablation study on data augmentation is presented in Section 5.4. An alternative could be to measure and train with more leakage traces. In the same section, we tune the number of training traces and observe its effect on the attack performance. Second, although our attack method uses the plaintext/ciphertext as labels during the training phase, in the key recovery phase, the leakage model is involved in labeling the intermediate data and calculating PD_k (Equation 16). Like other SCA attacks, a precise estimation of the leakage model could significantly increase the attack performance. In Section 5, multiple leakage modes are tested, and one can observe significant differences in attack performance. Considering the above discussions, we formulate the proposed attack scheme in Algorithm 1.

Algorithm 1 Plaintext/Ciphertext-based NP-DLSCA

Input: traces $\mathbf{T}$, plaintext/ciphertext bytes $\mathbf{d}$, leakage model M, augmentation level γ
Output: most-likely key k^*
1: $\mathsf{f}_{\boldsymbol{\theta}_d} = \mathsf{train}(\mathbf{T}, \mathbf{d}, \gamma)$
2: $\mathbf{Pr} = \mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{T})$
3: **for** k **in** $\mathcal{K}$ **do**
4: $\quad \mathbf{corr}_k = \mathsf{corr}(\mathbf{Pr}, \mathsf{PD}_k^{\mathsf{M}}(\mathbf{d}))$
5: **end for**
6: $k^* = \mathsf{arg\,max}\,\mathbf{corr}$

4.5 Profiling Attacks vs. Our Method

From Algorithm 1, our attack method follows a similar strategy as the profiling attack, which first trains a model by mapping the input leakage measurements with output labels, then performs key recovery based on the output probability and key guesses. Based on Equation 12, to estimate $\mathsf{f}_{\boldsymbol{\theta}}$, we introduce label distance to approximate the likelihood of each label being selected as the correct

label, then extend it to the plaintext/ciphertext distribution PD_k to estimate $\mathsf{map}'_k(\mathsf{f}_{\boldsymbol{\theta}})$. Finally, given a correct key k^*, the following approximation is satisfied:

$$\mathsf{PD}^{\mathsf{M}}_{k^*}(\mathbf{d}) \approx \mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{T}). \tag{18}$$

However, $\mathsf{PD}_k(\mathbf{d})$ is not perfect from the practical point of view: the leakage features do not necessarily follow a linear relationship with the intermediate label. We introduce a function C that evaluates the capability of the model mapping its input leakage traces to output labels:

$$\mathsf{C}(\mathsf{PD}^{\mathsf{M}}_{k^*}(\mathbf{d})) \lesssim \mathsf{C}(\mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{T})). \tag{19}$$

Then, extending to the profiling attack, we have:

$$\mathsf{C}(\mathsf{PD}^{\mathsf{M}}_{k^*}(\mathbf{d})) \lesssim \mathsf{C}(\mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{T})) = \mathsf{C}(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{T})). \tag{20}$$

Therefore, when evaluating the key recovery efficiency with a function E, the following inequality is satisfied:

$$\mathsf{E}(\mathsf{corr}(\mathsf{PD}^{\mathsf{M}}_{k}(\mathbf{d}),\ \mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{T}))) \leq \mathsf{E}(\mathsf{corr}(\mathsf{map}'_{k^*}(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{T}),\ \mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{T}))) = \mathsf{E}(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{T})). \tag{21}$$

Equation 21 represents the relationship between our method and profiling attacks. Assuming $\mathsf{f}_{\boldsymbol{\theta}_d}$ and $\mathsf{f}_{\boldsymbol{\theta}}$ generalize equally well to the leakage measurements, the attack performance of profiling attack ($\mathsf{E}(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{T}))$) is the upper bond of our attack method ($\mathsf{E}(\mathsf{corr}(\mathsf{PD}^{\mathsf{M}}_{k}(\mathbf{d}),\ \mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{T}))))$. Indeed, $\mathsf{f}_{\boldsymbol{\theta}}$ maps the leakage traces directly to the intermediate data. Although our method learns from plaintexts/-ciphertexts that are bijective to the intermediate data, it requires an extra step to correlate with imperfect PD_k to find out the real mapping function map'_{k^*}, thus leading to a degraded attack performance. In Section 5.3, we empirically validate this assumption.

5 Experimental Results

In this section, we benchmark the attack performance of different attack methods. Besides comparing with various non-profiling SCA methods, including correlation power analysis (CPA) [5], Multi-output DLSCA (MOR) [13], and Differential Deep Learning Analysis (DDLA) [38], we also benchmark with DLSCA to demonstrate the attack capability of PLDL. Since non-profiling and profiling attacks follow different attack strategies, the benchmarks are performed with two different settings. When performing hyperparameter evaluation in Section 5.4, we consider the first setting.

Benchmark with non-profiling attacks. The *same* leakage traces are used for training/partitioning and attack. The number of traces for each dataset is detailed in Table 2.

Benchmark with profiling attack. A set of leakage traces is used for model training, and the numbers follow Table 2. Then, a *new* set of traces is used for the key recovery; the corresponding data sizes are detailed in Table 1.

Table 1: The number of attack traces for the profiling setting.

	ASCAD_F	ASCAD_R	CHES_CTF	AES_RD	AES_HD
Trace num.	30 000	30 000	5 000	20 000	20 000

The deep learning model and hyperparameters are the same for all attack methods. [9] Consequently, we employ a convolution neural network (CNN) from [31] due to its excellent performance in various attack settings.[10] The network consists of a convolution block with a convolution layer (kernel number: 4; size: 40; stride: 20), an average pooling layer (size: 2; stride: 2), and a batch normalization layer, followed by two dense layers with 400 neurons and an output layer with 256 neurons. $Selu$ is used for the layer activation except for the last layer that uses $Softmax$; the batch size is 800. Regarding training epochs, the DL model for DDLA has trained for 50 epochs for each key guess [38] [11]; the rest of the models are trained for 250 epochs. An evaluation of training epochs can be found in Section 5.4. Data augmentation is applied to all *DL-based* attack methods for a fair comparison, which is realized by randomly shifting the leakage measurement within a pre-defined augmentation level equal to 10. A study of data augmentation is given in Section 5.4.

The rest of the attack settings follow Table 2. For DDLA, following the original paper, the selected leakage models are HW and LSB; for the rest, we consider HW and ID leakage models in our benchmarks. Note that AES_HD leaks in the Hamming distance on the last round of AES. Thus, we employ the Hamming distance labeling for all attack methods except ours, as our method only requires ciphertext information. As mentioned in Section 2.6, key rank is used to assess the attack performance of each method. When an attack cannot break the target with the given number of attack traces, its performance is measured by its final key rank value (such as KR10, meaning the key rank of the correct key is 10). Otherwise, it is evaluated by calculating the required number of attack traces to reach the key rank of zero. To reduce the effect of random factors (e.g., random weight initialization) on the attack performance, each attack method (except CPA) is executed ten times independently. The

[9] We acknowledge that the customization of deep learning models for each dataset and method would potentially lead to better attack performance. However, it also introduces more variables, such as model complexity and training effort, making our benchmark extremely complex. Additionally, benchmarking with customized DL could introduce more uncertainty, as one cannot guarantee that it is optimal for a specific setting.

[10] The deep learning models were implemented in Python version 3.6, using TensorFlow library version 2.6.0. The model training algorithms were run on an Nvidia GTX 1080TI graphics processing unit (GPU), managed by Slurm workload manager version 19.05.4.

[11] We have also tried with 100 epochs but notice that it performs worse than the model trained with 50 epochs.

attack results are averaged to represent the general attack performance of an attack method.

5.1 Datasets

Our experiments consider the following five datasets. Four are software targets, and one is a hardware target. All software targets are protected - three with masking and one with a hiding countermeasure.

ASCAD_F. The ASCAD datasets contain the measurements from an 8-bit AVR microcontroller running a masked AES-128 implementation [3].

ASCAD_R. This dataset uses the same measurement setup as ASCAD_F [3]. The difference is that ASCAD_R also provides traces with random keys. We only use the leakage traces with a fixed key (thus, the dataset part commonly used for testing with DLSCA).

AES_RD. For this dataset, the target smartcard is an 8-bit Atmel AVR microcontroller. The protection uses random delay countermeasures described by Coron and Kizhvatov [10]. Adding random delays to the normal operation of a cryptographic algorithm affects the misalignment of important features, making the attack more difficult to conduct.

CHES_CTF [12]**.** This dataset refers to the CHES Capture-the-flag (CTF) AES-128 measurements released in 2018 for the Conference on Cryptographic Hardware and Embedded Systems (CHES). The traces consist of masked AES-128 encryption running on a 32-bit STM microcontroller.

AES_HD [13]**.** This dataset is first introduced in [19], targeting an unprotected hardware implementation of AES-128 written in VHDL in a round-based architecture. Side-channel traces were measured using a high sensitivity near-field EM probe, placed over a decoupling capacitor on the power line on Xilinx Virtex-5 FPGA of a SASEBO GII evaluation board.

The detailed attack settings of these datasets are presented in Table 2. The last two columns list the intermediate data considered in the literature and our work, accompanied by the corresponding maximum Pearson correlation values (calculated by correlating labels with each leakage feature for all traces, then selecting the maximum correlation coefficient). The considered datasets contain almost no leakages on the targeted intermediate data. Note that for the AES_HD dataset, we attack the last round of AES with the knowledge of ciphertexts $\mathbf{c}$; for the rest, we attack the first round of AES with plaintexts $\mathbf{p}$.

5.2 Leakage Models

The leakage model simulates the hypothetical power consumption to process one byte (as we attack the AES cipher that is byte-oriented). Different leakage models can be adopted in practice, and their results may vary depending on the target device. In total, we consider four leakage models in this work. One simple option is to consider the most significant bit (MSB) or the least significant

Table 2: Summary of the tested datasets.

Dataset	Traces/Samples	Protection	Literature/Corr.	This work/Corr.
ASCAD_F	30 000/1 400	Boolean mask	$\mathsf{Sbox}(p_2 \oplus k_2)$/0.018	p_2/0.023
ASCAD_R	30 000/5 000	Boolean mask	$\mathsf{Sbox}(p_2 \oplus k_2)$/0.026	p_2/0.035
CHES_CTF	40 000/2 200	Boolean mask	$\mathsf{Sbox}(p_0 \oplus k_0)$/0.013	p_0/0.033
AES_RD	30 000/3 500	Random delay	$\mathsf{Sbox}(p_0 \oplus k_0)$/0.019	p_0/0.016
AES_HD	30 000/1 250	None	$\mathsf{Sbox}^{-1}(c_7 \oplus k_7) \oplus c_{11}$/0.069	c_7/0.018

bit (LSB) of a byte (we use LSB in this work). This leakage model results in two classes. For the HW leakage model, the adversary assumes the leakage is proportional to the sensitive variable's Hamming weight. This leakage model results in nine classes for a single intermediate byte for the AES cipher. Another type of leakage model results from the XOR between two values. Commonly, the Hamming Weight of this XOR is calculated and is referred to as the Hamming Distance. A typical approach for AES is to compute the XOR (or HW of the XOR, i.e., HD) between the final output and the S-box input of the last round. Like the HW leakage model, the HD leakage model results in nine classes or a single intermediate byte for the AES cipher. For the Identity (ID) leakage model, an adversary considers the leakage as an intermediate cipher value. This leakage model results in 256 classes for a single intermediate byte for the AES cipher.

5.3 Performance Evaluation

Attack Performance. In this section, we evaluate the attack performance of different attack methods. The benchmark results are shown in Table 3 and Table 4. Results for different leakage models (HW/HD and ID/LSB) are separated by '/'. As mentioned, our method has been benchmarked with non-profiling and profiling attacks. The best results obtained in the non-profiling settings are marked in **bold**. For profiling settings, they are marked in ***bold italic***.

Table 3: Performance benchmark with non-profiling attacks.

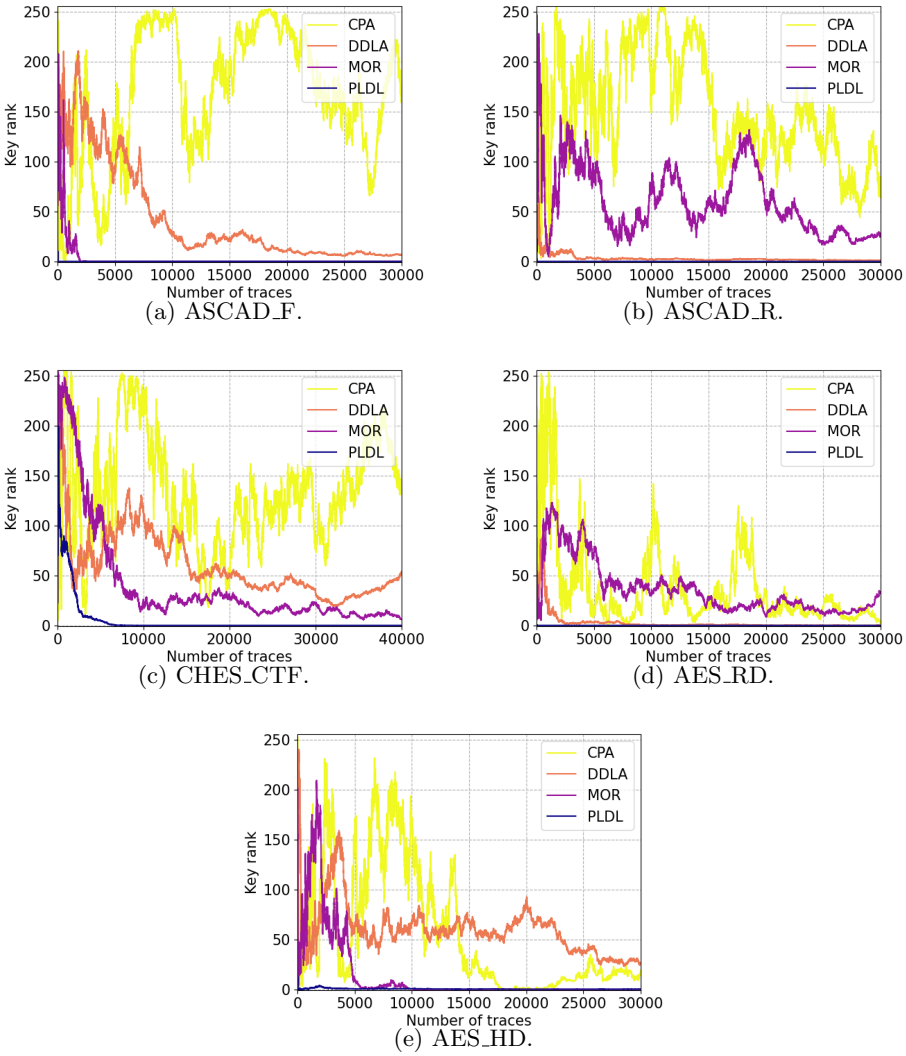
Dataset	CPA	MOR	DDLA	PLDL
ASCAD_F	KR161/KR47	1 957/638	KR7/309	**8/111**
ASCAD_R	KR64/KR8	KR28/KR9	27 266/KR48	**20/19**
CHES_CTF	KR139/KR220	KR6/KR31	KR54/KR85	**6 121/KR2**
AES_RD	KR2/KR31	KR33/3 112	2 541/KR2	**1/57**
AES_HD	KR19/KR145	5 593/KR10	KR26/KR20	**60/KR6**

PLDL performs significantly better in all test cases when benchmarking with non-profiling attack methods, as shown in Table 3. For instance, when attacking the AES_RD dataset with the HW leakage model, PLDL only requires a single trace to reveal the key, while the second best, DDLA, requires more than 2 500 traces. Although DDLA may perform better by optimizing model hyperparameters and training settings, it always suffers from significant training effort (10 hours per leakage model). MOR is considered an improved version of DDLA, as an adversary only needs to train one model instead of 256 models for a subkey byte. However, from the attack performance perspective, MOR improves marginally compared to DDLA. Indeed, both methods rely on the model to differentiate the correct label guess from the wrong guesses. Although MOR only requires training a single model, the learning objective of the model could be rather vague, as only one of the 256 outputs corresponds to the predicted label associated with the correct key k^*. When training, the model would try to fit on each label; the 255 wrong labels would negatively contribute to the model's generalization to the dataset, thus degrading the attack performance. Finally, CPA failed to reveal the correct key in all settings. We have also tested second-order CPA with feature recombination on the masked datasets. Although it leads to a faster convergence of key rank, the number of leakage traces used for benchmarking is insufficient to reveal the correct key. The evolution of the key rank for each method is shown in Figure 3 and Figure 4. PLDL leads to the fastest key rank convergence, indicating its strong attack capability in various attack settings.

Table 4: Performance benchmark with DLSCA.

Dataset	DLSCA	PLDL
ASCAD_F	464/147	***8*/*44***
ASCAD_R	458/62	***30*/*214***
CHES_CTF	1 943/***KR25***	***230***/KR90
AES_RD	530/163	***22*/*136***
AES_HD	7 077/KR21	***3 173*/*KR4***

When benchmarking with profiling attacks, as shown in Table 4, we see outstanding performances for both PLDL and DLSCA. Surprisingly, our attack outperforms the profiling attack in seven test cases. To our knowledge, this is the first time a non-profiling SCA method outperforms the profiling attack. Indeed, as discussed in subsection 4.5, profiling attack is theoretically the upper bound of our methods. From a practical point of view, if $\mathsf{f}_{\boldsymbol{\theta}_d}$ generalized better than $\mathsf{f}_{\boldsymbol{\theta}}$ ($\mathsf{C}(\mathsf{f}_{\boldsymbol{\theta}_d}) > \mathsf{C}(\mathsf{f}_{\boldsymbol{\theta}})$) due to, for instance, traces pre-preprocessing and hyperparameter tuning, it is not a surprise that PLDL outperforms the profiling



(a) ASCAD_F. (b) ASCAD_R.

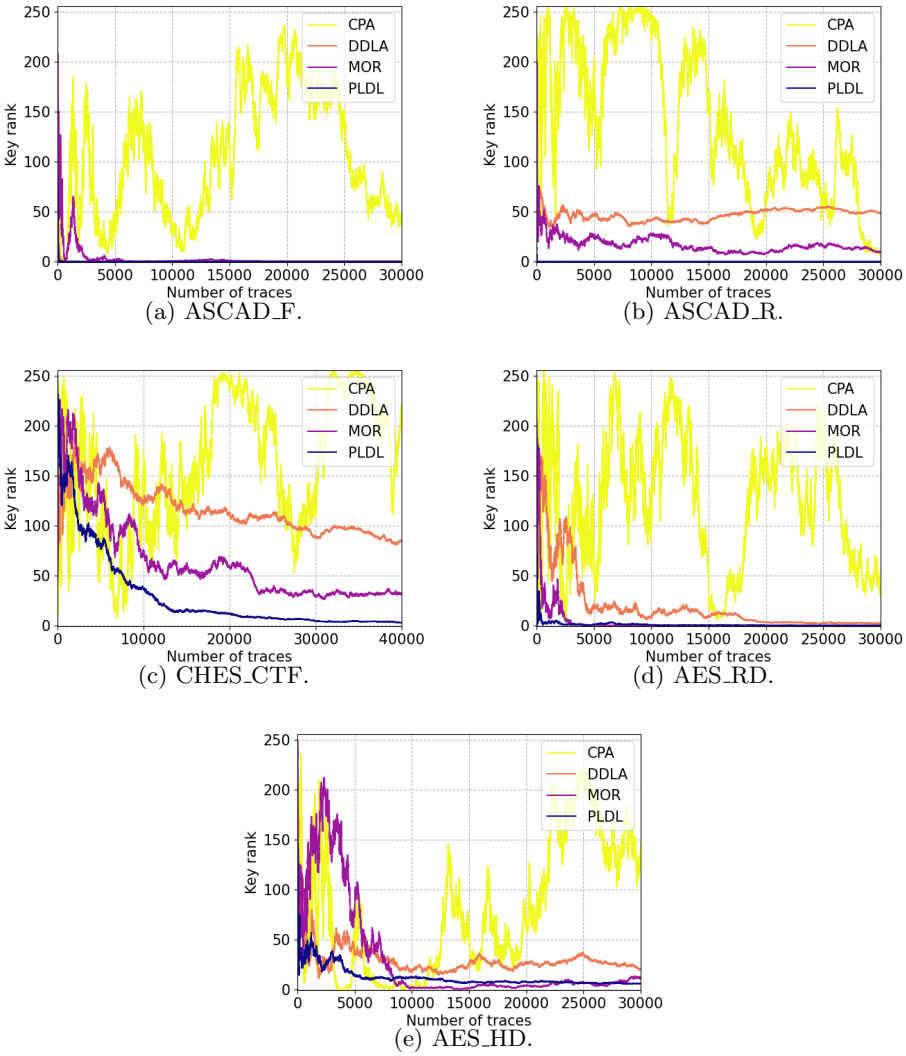
(c) CHES_CTF. (d) AES_RD.

(e) AES_HD.

Fig. 3: Non-profiling attack benchmark with HW leakage model (ASCAD_F, ASCAD_R, CHES_CTF, and AES_RD) and HD leakage model (AES_HD).

attack. Still, with sufficient training traces and tuning, we expect better attack performance for DLSCA than PLDL.

Robustness to Noise. Two desynchronization levels, 50 and 100, are considered to simulate the time-jitter effect. CPA is removed from the benchmark due to its poor attack performance in the first set of experiments. The results are



(a) ASCAD_F. (b) ASCAD_R.

(c) CHES_CTF. (d) AES_RD.

(e) AES_HD.

Fig. 4: Non-profiling attack benchmark with the ID leakage model. Note that for DDLA, since the ID leakage model cannot work, we use the LSB leakage model.

shown in Table 5 and Table 6. Again, the best results obtained for the non-profiling and profiling settings are marked in **bold** and ***bold italic***, respectively.

PLDL performs significantly better than its counterparts in non-profiling settings. It performs the best in nine of ten attack scenarios; for the cases where the key rank reaches zero (the correct key ranks the first), it performs at least 6× better than other methods. MOR outperforms our method in one test case. Compared with its attack performance with no noise Table 3, MOR is slightly

Table 5: Performance benchmark with desynchronization 50.

Dataset	MOR	DDLA	PLDL	DLSCA	PLDL
ASCAD_F	2 910/KR34	KR152/KR118	**10**/**112**	985/***280***	***14***/531
ASCAD_R	KR12/KR62	KR69/KR156	**54**/**17**	KR9/***223***	***22***/942
CHES_CTF	KR5/KR120	KR124/KR123	**KR4**/**KR116**	KR41/***KR56***	***KR14***/KR135
AES_RD	KR21/3 200	2 517/KR7	**1**/**555**	429/97	***21***/***95***
AES_HD	6 098/**KR8**	KR94/KR116	**951**/KR14	***14 127***/KR27	KR2/***KR13***

influenced by added noise when attacking AES_HD, while PLDL experiences a considerable performance reduction. Indeed, when targeting the ciphertext with PLDL, the deep learning model has to find the leakages related to the Sbox^{-1} output. However, this data has limited leakage on this intermediate data, thus increasing the learning difficulties for PLDL. MOR, on the other hand, utilizes the Hamming distance labeling directly. The stronger leakage could lead to robust performance. Moving to the profiling setting, DLSCA reaches a similar attack performance compared with PLDL: out of ten test cases, DLSCA is better in four cases, and PLDL is better in the rest of the cases. Compared with the results in Table 3, DLSCA performs better in more test cases. Indeed, the introduction of time randomness reduces both $\mathsf{C}(\mathsf{f}_{\boldsymbol{\theta}_d})$ and $\mathsf{C}(\mathsf{f}_{\boldsymbol{\theta}})$, which could potentially reduce the capability difference between these two models. Additionally, one should note that PLDL is based on the correlation between PD_k and $\mathsf{f}_{\boldsymbol{\theta}_d}$. A reduced $\mathsf{C}(\mathsf{f}_{\boldsymbol{\theta}_d})$ would require more leakage traces to compensate for the performance loss of PLDL.

Similarly, PLDL performs the best in most scenarios in non-profiling settings when increasing the desynchronization level from 50 to 100. As shown in Table 6, with a single deep learning model, PLDL reaches a key rank of zero in seven out of ten cases, and the attack performance is superior compared to its counterparts. Compared with DLSCA, there are more cases (five) where DLSCA performs better, which confirms our previous assumption.

Table 6: Performance benchmark with desynchronization 100.

Dataset	MOR	DDLA	PLDL	DLSCA	PLDL
ASCAD_F	KR5/KR34	KR153/KR149	**8**/**3 373**	2 139/1 695	***17***/***1 408***
ASCAD_R	KR12/KR62	KR68/KR130	**154**/**867**	KR24/***1 055***	***30***/2 245
CHES_CTF	KR13/KR137	KR125/173	**KR12**/**KR99**	KR149/***KR59***	***KR9***/KR91
AES_RD	KR54/4 521	4876/KR4	**1**/**561**	732/***198***	***22***/KR2
AES_HD	6 077/**KR9**	KR131/KR142	**166**/KR10	***10 691***/***KR8***	19 146/KR16

5.4 Hyperparameter Evaluation

In this section, we explore the influence of various hyperparameters on the PLDL attack performance.

Data Augmentation. As mentioned in Section 4.4, data augmentation is necessary for PLDL. Next, we perform an ablation study on data augmentation and investigate the augmentation level's influence on the attack performance. The results are shown in Table 7.

Table 7: Ablation study on data augmentation (DA).

Dataset	DA-0	DA-5	DA-10	DA-20
ASCAD_F	KR10/KR46	8/118	8/111	8/429
ASCAD_R	KR64/KR143	19/13	20/19	2/12
CHES_CTF	8 573/KR74	8 506/KR10	6 121/KR2	11 742/KR53
AES_RD	KR50/KR131	1/22 085	1/57	1/54
AES_HD	KR35/KR66	11 916/KR15	60/KR6	256/KR8

When setting the data augmentation level to zero, PLDL could only recover the key in one test case. Then, one can observe a performance boost when introducing random shifts to datasets. The ranges between DA-5 and DA-10 are optimal for most test cases. When the augmentation level reaches 20, the attack performance worsens in several settings. We can conclude the necessity of data augmentation for PLDL. Still, a too-large data augmentation level would reduce the attack performance, as it would increase the difficulties of the deep learning model fitting the leakage, necessitating longer training and larger models.

Dataset Size. DL-based methods are known to be data-hungry. In the SCA context, more leakage traces would be helpful to compensate for noise and reveal the underlining distribution of leakage features. When looking at Table 8, as expected, more training traces lead to better attack performance for PLDL.

One could observe that CHES_CTF and AES_HD require more traces than the other datasets. For AES_HD, as discussed before, the possible cause would be the limited leakage on the Sbox output. On the other hand, the performance for CHES_CTF could be explained by its leakage type. According to the literature [41,35], the CHES_CTF dataset mainly contains HW leakages, while when attacking with the ID leakage model, it is less likely to reveal the key with the same number of attack traces. Since the proposed method is trained with plaintext/ciphertext values, the method's efficiency intrinsically relies on the intermediate data's ID leakages (HW of the intermediate data is not bijective to

Table 8: Study on the influence of the data size.

Dataset	5 000	10 000	20 000	30 000
ASCAD_F	KR3/KR46	202/KR37	7/848	8/111
ASCAD_R	KR4/KR40	56/KR2	45/60	20/19
CHES_CTF	KR212/KR187	KR72/KR121	KR36/KR114	KR10/KR52
AES_RD	707/KR50	1/KR14	1/2 332	1/57
AES_HD	KR18/KR48	KR11/KR25	4 709/KR4	60/KR6

plaintexts/ciphertext). When a dataset mainly has HW leakages, the model $\mathsf{f}_{\boldsymbol{\theta}_d}$ would struggle in mapping the plaintext to the (HW) leakage features, finally leading to a reduced attack performance.

Training Epochs. Unlike data size, more training epoch is unnecessary to improve the mapping capability of a deep learning model from input to output. In contrast, it could reduce the generalization ability of the deep learning model on the *unseen* dataset, referred to as *overfitting*. Fortunately, in the non-profiling context, the concerns about deep learning generalization and overfitting are less important, as an adversary would use all leakage traces to launch attacks. Table 9 shows the performance variation of PLDL when training with different numbers of epochs. Training with 50 epochs is insufficient for most of the settings; with extra 100 epochs of training (150), eight out of ten attacks lead to successful key recovery. For the CHES_CTF dataset, one could observe a steady decrease in key rank value, indicating that the deep learning model is gradually transferring the HW-related feature and learning to connect with plaintext labels. Finally, this observation confirms our assumption made in Section 5.4 about CHES_CTF performance and the limitation of PLDL.

Table 9: Study on the influence of the training epoch.

Dataset	50	100	150	200
ASCAD_F	34/451	8/237	8/112	8/81
ASCAD_R	KR3/KR6	54/19	18/13	19/14
CHES_CTF	KR22/KR162	KR9/KR160	28 065/KR73	6 855/KR17
AES_RD	1/KR36	1/KR2	1/54	1/64
AES_HD	77/KR6	74/KR6	19/KR5	58/KR4

6 Conclusions and Future Work

This paper introduces a novel plaintext/ciphertext-based non-profiling SCA method called PLDL leveraging the bijectivity between plaintext/ciphertext and key-related intermediate data. We define the plaintext distribution (PD_k) to approximate the likelihood of each plaintext being selected as the correct plaintext given a key guess k and then use this approximation to correlate with the prediction output of a profiling model trained with plaintext/ciphertext to retrieve the correct key. PLDL shows outstanding performance compared with state-of-the-art non-profiling SCA methods, as depicted in Table 10. Moreover, the attack performance of PLDL is comparable with DLSCA under a more restricted attack assumption (the availability of a fully controlled cloned device).

Table 10: Comparison of non-profiling attacks. Since each attack scenario takes a different amount of time, the time complexity is measured by averaging the time consumption with all (ten) attack settings.

Method	Time complexity	Leakage model limitation	Attack performance
CPA	low (1 min)	no	low to moderate
DDLA	high (246 min)	yes (non-bijective only)	moderate
MOR	moderate (10 min)	no	moderate
This work	moderate (6 min)	no	high

Several directions can be investigated following this work. First, knowing that PD_k is imperfect, one could investigate a better approximation of $\mathsf{f}_{\boldsymbol{\theta}_d}$ with, for instance, stochastic models. Second, since the proposed method relies on the intermediate data's ID leakages, it would be interesting to investigate an approach that enables this method on datasets that only leaks HW values. Finally, an interesting option to further improve attack performance could be to make ensembles of neural networks to be used in NP-DLSCA.

References

1. Alipour, A., Papadimitriou, A., Beroulle, V., Aerabi, E., Hély, D.: On the performance of non-profiled differential deep learning attacks against an aes encryption algorithm protected using a correlated noise generation based hiding countermeasure. In: Proceedings of the 23rd Conference on Design, Automation and Test in Europe. p. 614–617. DATE '20, EDA Consortium, San Jose, CA, USA (2020)
2. Archambeau, C., Peeters, E., Standaert, F.X., Quisquater, J.J.: Template attacks in principal subspaces. In: Lecture Notes in Computer Science, pp. 1–14. Springer Berlin Heidelberg (2006). https://doi.org/10.1007/11894063_1, `https://doi.org/10.1007/11894063_1`

3. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptographic Engineering **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8, https://doi.org/10.1007/s13389-019-00220-8
4. 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), https://www.ndss-symposium.org/ndss-paper/mind-the-portability-a-warriors-guide-through-realistic-profiled-side-channel-analysis/
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 16–29. Springer (2004)
6. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2017. pp. 45–68. Springer International Publishing, Cham (2017)
7. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: CHES. LNCS, vol. 2523, pp. 13–28. Springer (August 2002), San Francisco Bay (Redwood City), USA
8. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3, https://doi.org/10.1007/3-540-36400-5_3
9. CommonCriteria: Common criteria v3.1 (2017). https://www.commoncriteriaportal.org/cc/index.cfm?.
10. Coron, J.S., 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. pp. 156–170. Springer (2009)
11. Dada, E.G., Bassi, J.S., Chiroma, H., Adetunmbi, A.O., Ajibuwa, O.E., et al.: Machine learning for email spam filtering: review, approaches and open research problems. Heliyon **5**(6), e01802 (2019)
12. Do, N.T., Hoang, V.P., Doan, V.S., Pham, C.K.: On the performance of non-profiled side channel attacks based on deep learning techniques. IET Information Security **n/a**(n/a). https://doi.org/https://doi.org/10.1049/ise2.12102, https://ietresearch.onlinelibrary.wiley.com/doi/abs/10.1049/ise2.12102
13. Do, N.T., Le, P.C., Hoang, V.P., Doan, V.S., Nguyen, H.G., Pham, C.K.: Mo-dlsca: Deep learning based non-profiled side channel analysis using multi-output neural networks. In: 2022 International Conference on Advanced Technologies for Communications (ATC). pp. 245–250 (2022). https://doi.org/10.1109/ATC55345.2022.9943024
14. EMVCo: Emv specifications (2001). https://www.emvco.com/
15. Goodfellow, I.J., Bengio, Y., Courville, A.: Deep Learning. MIT Press, Cambridge, MA, USA (2016), http://www.deeplearningbook.org
16. Hauke, J., Kossowski, T.: Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data. Quaestiones geographicae **30**(2), 87 (2011)

17. Hoang, V.P., Do, N.T., Doan, V.S.: Efficient non-profiled side channel attack using multi-output classification neural network. IEEE Embedded Systems Letters pp. 1–1 (2022). https://doi.org/10.1109/LES.2022.3213443
18. Hospodar, G., Gierlichs, B., Mulder, E.D., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. J. Cryptogr. Eng. **1**(4), 293–302 (2011). https://doi.org/10.1007/s13389-011-0023-x, https://doi.org/10.1007/s13389-011-0023-x
19. 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 Transactions on Cryptographic Hardware and Embedded Systems pp. 148–179 (2019)
20. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19. pp. 388–397. Springer (1999)
21. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. pp. 388–397. CRYPTO '99, Springer-Verlag, London, UK, UK (1999), http://dl.acm.org/citation.cfm?id=646764.703989
22. Kuroda, K., Fukuda, Y., Yoshida, K., Fujino, T.: Practical aspects on non-profiled deep-learning side-channel attacks against aes software implementation with two types of masking countermeasures including rsm. In: Proceedings of the 5th Workshop on Attacks and Solutions in Hardware Security. p. 29–40. ASHES '21, Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3474376.3487285, https://doi.org/10.1145/3474376.3487285
23. Kwon, D., Hong, S., Kim, H.: Optimizing implementations of non-profiled deep learning-based side-channel attacks. IEEE Access **10**, 5957–5967 (2022). https://doi.org/10.1109/ACCESS.2022.3140446
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. Lerman, L., Medeiros, S.F., Veshchikov, N., Meuter, C., Bontempi, G., Markowitch, O.: Semi-supervised template attack. In: Constructive Side-Channel Analysis and Secure Design, pp. 184–199. Springer Berlin Heidelberg (2013). https://doi.org/10.1007/978-3-642-40026-1_12, https://doi.org/10.1007/978-3-642-40026-1_12
26. Lerman, L., Poussier, R., Bontempi, G., Markowitch, O., Standaert, F.X.: Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 20–33. Springer (2015)
27. Lu, X., Zhang, C., Cao, P., Gu, D., Lu, H.: Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 235–274 (2021)
28. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 3–26. Springer (2016)
29. Nassif, A.B., Shahin, I., Attili, I., Azzeh, M., Shaalan, K.: Speech recognition using deep neural networks: A systematic review. IEEE access **7**, 19143–19165 (2019)

30. Perin, G., Chmielewski, L., Picek, S.: Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(4), 337–364 (Aug 2020). https://doi.org/10.13154/tches.v2020.i4.337-364, https://tches.iacr.org/index.php/TCHES/article/view/8686
31. Perin, G., Wu, L., Picek, S.: Exploring feature selection scenarios for deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(4), 828–861 (Aug 2022). https://doi.org/10.46586/tches.v2022.i4.828-861, https://tches.iacr.org/index.php/TCHES/article/view/9842
32. Picek, S., Heuser, A., Jovic, A., Legay, A., Knezevic, K.: Profiled SCA with a new twist: Semi-supervised learning. IACR Cryptol. ePrint Arch. p. 1085 (2017), http://eprint.iacr.org/2017/1085
33. Poussier, R., Standaert, F.X., Grosso, V.: Simple key enumeration (and rank estimation) using histograms: An integrated approach. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 61–81. Springer (2016)
34. Rajkomar, A., Dean, J., Kohane, I.: Machine learning in medicine. New England Journal of Medicine **380**(14), 1347–1358 (2019)
35. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2021**(3), 677–707 (Jul 2021). https://doi.org/10.46586/tches.v2021.i3.677-707, https://tches.iacr.org/index.php/TCHES/article/view/8989
36. Schindler, W., Lemke, K., Paar, C.: A stochastic model for differential side channel cryptanalysis. In: Cryptographic Hardware and Embedded Systems – CHES 2005, pp. 30–46. Springer Berlin Heidelberg (2005). https://doi.org/10.1007/11545262_3, https://doi.org/10.1007/11545262_3
37. Sebe, N., Cohen, I., Garg, A., Huang, T.S.: Machine learning in computer vision, vol. 29. Springer Science & Business Media (2005)
38. Timon, B.: Non-profiled deep learning-based side-channel attacks with sensitivity analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 107–131 (2019)
39. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 147–168 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.147-168, https://tches.iacr.org/index.php/TCHES/article/view/8586
40. Wu, L., Perin, G., Picek, S.: The best of two worlds: Deep learning-assisted template attack. IACR Transactions on Cryptographic Hardware and Embedded Systems **2022**(3), 413–437 (Jun 2022). https://doi.org/10.46586/tches.v2022.i3.413-437, https://tches.iacr.org/index.php/TCHES/article/view/9707
41. Wu, L., Perin, G., Picek, S.: I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. IEEE Transactions on Emerging Topics in Computing (2022)
42. 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), 1–36 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.1-36, https://tches.iacr.org/index.php/TCHES/article/view/8391