IACR Communications in Cryptology
ISSN 3006-5496, Vol. 1, No. 3, 24 pages.
https://doi.org/10.62056/a36cy7qiu

Plaintext-based Side-channel Collision Attack

Lichao Wu[1], Sébastien Tiran[2], Guilherme Perin[3] and Stjepan Picek[4]

[1] Technical University of Darmstadt, Darmstadt, Germany
[2] Independent Researcher, Delft, The Netherlands
[3] Leiden University, Leiden, The Netherlands
[4] Radboud University, Nijmegen, The Netherlands

Abstract. Side-channel Collision Attacks (SCCA) is a classical method that exploits information dependency leaked during cryptographic operations. Unlike collision attacks that seek instances where two different inputs to a cryptographic algorithm yield identical outputs, SCCAs specifically target the internal state, where identical outputs are more likely. Although SCCA does not rely on the pre-assumption of the leakage model, it explicitly operates on precise trace segments reflecting the target operation, which is challenging to perform when the leakage measurements are noisy. Besides, its attack performance may vary dramatically, as it relies on selecting a reference byte (and its corresponding leakages) to "collide" other bytes. A poor selection would lead to many bytes unrecoverable. These two facts make its real-world application problematic.
This paper addresses these challenges by introducing a novel plaintext-based SCCA. We leverage the bijective relationship between plaintext and secret data, using plaintext as labels to train profiling models to depict leakages from varying operations. By comparing the leakage representations produced by the profiling model instead of the leakage segmentation itself, all secret key differences can be revealed simultaneously without processing leakage traces. Furthermore, we propose a novel error correction scheme to rectify false predictions further. Experimental results show that our approach significantly surpasses the state-of-the-art SCCA in both attack performance and computational complexity (e.g., training time reduced from approximately three hours to five minutes). These findings underscore our method's effectiveness and practicality in real-world attack scenarios.

Keywords: Side-channel Analysis · Side-channel Collision Attack · Deep Learning.

1 Introduction

Side-channel analysis (SCA) represents a powerful implementation attack on cryptographic algorithms. Side-channel analysis on symmetric-key cryptography implementations is typically divided into non-profiling and profiling attacks depending on the availability of a cloned device identical (or similar) to the target device. Non-profiling attacks operate without a clone device. The adversary gathers measurements encapsulating secret information, conducting statistical analysis to deduce the secrets. Conversely, profiling attacks assume the adversary has unrestricted access to a cloned device. Using this device, the adversary studies and comprehends the device's side-channel behavior and utilizes this understanding to extract secret information from the target device.

Side-channel collision attacks (SCCA), a unique type of non-profiling attacks, were first introduced in [SWP03]. Benefiting from the independence of any leakage model, the authors used SCA data to detect collisions in the internal state of an AES implementation. Recently, Staib et al. incorporated deep learning into SCCA (DL-SCCA) [SM23]. Like

E-mail: stjepan.picek@ru.nl (Stjepan Picek)

Received: 2024-07-07 Accepted: 2024-09-02

SCCA, DL-SCCA first preprocesses the leakage traces into segments representing the processing of each targeted intermediate data, characterizing these trace segments, and finally recovering the key difference by comparing with a different trace segmentation with deep learning. Unfortunately, the limitations of SCCA remain despite the performance improvement, constraining its application to practical attack scenarios. Indeed, SCCA is significantly less studied and used than other profiling and non-profiling counterparts. We summarize SCCA's main limitation into three aspects:

- **Trace Segmentation.** Trace segmentation is the most intricate procedure for SCCA. Indeed, the traces segment is compared to extract secret information; imprecise segmentation could directly impact attack performance. Conventional SCCA relies on simple power analysis to identify the target operation, which is less precise and unsuitable when the target operation has indistinguishable leakage patterns. The latest DL-SCCA employs a multi-stage trace segmentation method involving deep learning-based sensitivity analysis and visual inspection. Regrettably, this method does not change the nature of SCCA, which relies on a precise leakage measurement and depends on human inspection (e.g., model tuning and finding leakages). Errors may be introduced at each trace segmentation stage, and the generality of this method is questionable.
- **Leakage Comparision.** The key recovery of SCCA relies on the comparison of the trace segment. A direct comparison could be problematic due to the timing misalignment and distortion of the leakage features. DL-SCCA improves this by employing a DL model trained on one trace segmentation (representing the processing of a single intermediate data) to attack a different trace segment. However, this incremental improvement does not eliminate the SCCA restriction. As shown in the original paper [SM23], selecting different trace segments for training significantly varies the success rate of the key recovery, ranging from 0% to 100%. Indeed, both SCCA and DL-SCCA rely on 1) Temporal consistency, the timing of leakage features between two trace segments must be well aligned for the trained model to extract information from features at the identical location; 2) Spatial consistency, the leakage features between two traces segments should be alike. Any discrepancy between features could diminish attack performance, which connects with the well-known SCA portability problem [BCH+20].
- **Error Tolerance.** The effectiveness of SCCA and DL-SCCA attacks relies entirely on the reference trace segment or the trained deep learning model. In practical attack contexts, the target intermediate data may have different physical leakages, challenging SCCA's key recovery capability in recovering all key bytes.

This paper introduces a novel plaintext-based side-channel collision attack (PSCCA) to circumvent the abovementioned limitations. Our main contributions are:

1. We propose a novel approach for SCCA that extracts the target leakages based on the plaintext. This method eliminates the need for trace segmentation, operating directly on the raw traces.
2. We introduce a novel plaintext-based collision method. This method represents and compares the target leakages with the extracted features, overcoming the temporal and spatial consistency issues.
3. Gathering the above two contributions, we employ multi-task learning to expedite the training process while maintaining attack performance. Our method can recover all key differences in five minutes with a single profiling model.
4. We present a novel error correction method that can rectify wrong key guesses based on the relationship of other key bytes.
5. We provide a comprehensive experimental analysis showcasing the exceptional attack performance of the proposed method. Our method outperforms the state-of-the-art

non-profiling SCA and DL-SCCA even without error correction. When applied, the success rate is further increased.

6. We perform hyperparameters evaluation on several critical hyperparameters relevant to our attack: data augmentation, batch size, and training epochs. Moreover, we discuss the efficiency of error correction in different settings.

The source code is available in `https://github.com/lichao-wu9/PSCCA`.

The rest of this paper is organized as follows. Section 2 provides the necessary background information. Section 3 discusses related works. Section 4 details the threat model and then provides a theoretical analysis of the method. Section 5 presents the scheme, and then a case study and discussion on the proposed attack are given. In Section 6, we provide experimental results, benchmarks, and the evaluation of critical hyperparameters. Section 7 discusses the proposed method and potential countermeasures. Section 8 concludes the paper and discusses potential future research directions.

2 Background

This section starts by introducing our notation. Then, it provides the relevant information about the side-channel analysis, side-channel collision attack, and evaluation metrics.

2.1 Notation

We denote sets using calligraphic letters such as $\mathcal{X}$, while the related uppercase letters X represent random variables and random vectors $\mathbf{X}$ that are defined over the domain of $\mathcal{X}$. The corresponding lowercase letters x and $\mathbf{x}$ symbolize realizations of X and $\mathbf{X}$, respectively. Furthermore, functions, represented as f, are given in a sans-serif font.

The symbol k represents a candidate key byte from a key space $\mathcal{K}$. Δ denotes the xor key difference between two different key bytes. A dataset $\mathbf{T}$ comprises traces, symbolized as $\mathbf{t}_i$, which have corresponding associations with plaintext/ciphertext pairs $\mathbf{d}_i \in \mathcal{D}$ and keys $\mathbf{k}_i$, or $k_{i,j}$ and $d_{i,j}$ when partial key recovery on byte j is under consideration. Given our emphasis on byte processing, we exclusively utilize byte vector notation for both plaintext/ciphertext and the key, while the trace index is omitted.

2.2 Side-channel Analysis

As briefly mentioned in the introduction section, side-channel analysis (SCA) on symmetric block ciphers can be broadly classified into two categories: profiling SCA and non-profiling SCA, based on the availability of a fully-controlled cloned device.

Profiling side-channel attacks aim to connect a set of inputs (leakage traces) to outputs (a probability vector of key guesses). Profiling attacks have two stages. In the profiling stage, the adversary builds a profiling model, represented as $\mathsf{f}_{\boldsymbol{\theta}}^{M}$, which is determined by a leakage model M and a group of learning parameters $\boldsymbol{\theta}$. In this paper, the terms $\mathsf{f}_{\boldsymbol{\theta}}^{M}$ and $\mathsf{f}_{\boldsymbol{\theta}}$ are used interchangeably. This model maps inputs (side-channel measurements) to outputs (classes obtained by evaluating the leakage model during a sensitive operation) using a set of N profiling traces. Then, in the attack stage, the trained model processes each attack trace and produces a probability vector $\mathbf{p}$, reflecting the likelihood of the related leakage value or label. The adversary chooses the best key candidate based on this probability vector. Given an effective profiling model, only a few measurements from the target device would be enough to reveal the secret. Typical profiling attacks methods include the template attack [CRR02], stochastic models [SLP05], and supervised machine learning-based attacks [HGM$^+$11, MPP16, PHJ$^+$17].

Non-profiling attacks assume less powerful adversaries. An adversary collects a series of traces created during the cryptographic operation of different plaintexts. Then, the adversary guesses part of the key by analyzing the correlation between the key-related intermediate values and the leakage measurements. Non-profiling attacks often follow a 'divide-and-conquer' approach. Initially, the adversary groups the traces based on the predicted intermediate value tied to the current key guess. If the groups show clear differences (the 'difference' depends on the attack method), it suggests that the current key guess is likely correct. Non-profiling attacks may require multiple measurements (possibly millions) to reveal secret assets. Simple power analysis (SPA) and differential power analysis (DPA) [KJJ99a] are commonly used for non-profiling attacks (but they can also be used for profiling attacks [MOP08]). Other examples include correlation power analysis (CPA) [BCO04], and some machine learning-based attacks [Tim19, DLH+22].

2.3 Side-channel Collision Attack

Side-channel collision attack (SCCA) is considered a non-profiling SCA (as it does not rely on a profiling device) but follows a different attack principle [SWP03, Bog07]. It exploits data inter-dependence leaked during cryptographic procedures by targeting the collision of an internal state, which is more likely to coincide between two cryptographic operations.

Concretely, an adversary monitors the side-channel information while the system processes different inputs and then searches for repeated leakage patterns signifying a collision event. When a collision is detected, the adversary uses this information to infer insights about the inter-dependencies of different key sections or the algorithm's internal state. For instance, let us consider the SubBytes operation of the Advanced Encryption Standard (AES) with the same substitution box (Sbox). The same data has been processed if two different Sbox operations lead to an identical side-channel pattern. Since the Sbox operation is bijective (that is, it establishes a one-to-one correspondence between two sets), we obtain the following equations for plaintext byte and key byte i and j:

$$\begin{aligned} \mathsf{Sbox}(k_i \oplus d_i) &= \mathsf{Sbox}(k_j \oplus d_j) \\ => \;\; k_i \oplus d_i &= k_j \oplus d_j \\ => \;\; k_i \oplus k_j &= d_i \oplus d_j. \end{aligned} \tag{1}$$

We represent the result of $k_i \oplus k_j$ as $\Delta_{i,j}$. Since $\Delta_{i,j}$ represents the correct key difference, it is further denoted as Δ^*. Unlike other SCA techniques focusing on key recovery, SCCA strives to expose the linear difference between various keys. After all key differences are predicted, an adversary guesses one key byte to achieve full key recovery, as the rest of the key can then be calculated based on this linear difference. Therefore, the remaining keyspace shrinks to 2^8.

2.4 Evaluation Metrics

This work employs the maximum log-likelihood distinguisher to obtain a cumulative sum $S(\Delta)$ for each Δ candidate:

$$S(\Delta) = \sum_{i=1}^{Q} \log(\mathbf{prob}(\Delta)), \;\; \Delta \in \mathcal{K}, \tag{2}$$

where $\mathbf{prob}(\Delta)$ denotes the probability of each $\Delta \in \mathcal{K}$ being chosen as the correct value; Q represents the number of attack traces. The result of an attack is represented by a Δ guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$, which is computed for Q traces in the attack phase. This vector orders the Δ candidates in descending likelihood, with g_1 being the most probable candidate and $g_{|\mathcal{K}|}$ the least probable one. The position of Δ^* within the Δ guessing vector $\mathbf{g}$ is utilized to estimate the effort needed to reveal the secret key difference, and the corresponding rank is referred to as the Δ rank. In the experimental section, each attack is conducted ten times, and we calculate the *success rate*, representing the percentage of successful attacks out of ten.

Since the side-channel collision attack engages multiple subbytes, we are also interested in the overall efficiency of the attack in retrieving $\Delta_{i,j}$ with different subbytes. Therefore, we also assess the *overall success rate*, indicating the percentage of successful recoveries of the key difference among all tested $\Delta_{i,j}$.

3 Related Works

The side-channel analysis community has researched profiling attacks for over two decades. The first profiling attack was introduced by Chari et al., which established the template attack [CRR02]. Other "classic" profiling attacks include the stochastic model [SLP05]. The field has also seen a rise in the application of machine learning techniques, initially employing simpler methods like random forest [LMBM13] and support vector machines [HGM+11], as the most common examples. Since 2016, significant attention has been given to deep learning techniques for profiling SCA. The first work utilizing convolutional neural networks (CNNs) established the potential of deep learning in breaking various targets [MPP16]. Current research has made considerable advancements, breaking datasets perceived as challenging a few years ago, even with a single measurement [LZC+21, WPP22a, PWP22]. While deep learning-based SCA provides excellent results, multiple challenges still need to be addressed [PPM+23]. One of the main ones is moving away from the “classical” profiling paradigm.

However, profiling attacks do not apply to black-box adversaries without access to a profiling device. In these cases, non-profiling SCA becomes relevant. The differential power analysis, a classical non-profiling SCA, is widely adopted in academia and industry [KJJ99b]. The concept of using deep learning in non-profiling SCA was first proposed by B. Timon [Tim19], which involved training multiple neural networks corresponding to different key guesses. Later, advancements by Hoang et al. introduced a multi-output classification technique for non-profiling SCA, which significantly improved the efficiency and effectiveness of the attack [HDD22]. Finally, multi-output classification (MOC) and multi-output regression (MOR) models for non-profiling SCA were investigated by Do et al. [DLH+22].

Side-channel collision attacks (SCCA) have received less attention than profiling and non-profiling attacks. The concept of SCCA was first introduced to use SCA data to detect collisions in the internal state of an AES implementation [SWP03]. The advantage of this strategy is its independence from any leakage model. Initially limited to single collision, these attacks were refined to include all possible collisions in the measurement set [MME10, MS16]. Furthermore, SCCAs can bypass countermeasures previously deemed “impossible” through conventional profiling SCA without knowing mask shares [WPP23]. The integration of deep learning into SCCA, known as Deep Learning Side-Channel Collision Attack (DL-SCCA), was recently proposed by Staib et al. [SM23]. This methodology designates a target byte as a reference, employing corresponding trace segments and plaintexts to train a deep learning model. The trained model is subsequently applied to a distinct trace segment to predict the key relationship. However, identifying the trace segment in this method is performed through deep learning-based sensitivity analysis and visual inspection, which could be problematic from a realistic perspective.

4 Plaintext-based Side-channel Collision Attack

This section first introduces the threat model we follow. Afterward, the proposed plaintext-based side-channel collision attack is presented.

4.1 Threat Model

This paper follows the known plaintext attack, an attack model where the attacker can access both the plaintext and ciphertext[1]. Typically, we assume the adversary has access to a device running the target cipher and possesses a fixed yet undisclosed key. The adversary can instruct the device to perform encryption or decryption operations but cannot manipulate their values. An adversary lacks information about the hardware implementation, the countermeasure settings, and the source code. To execute attacks, the adversary captures multiple side-channel leakages using an oscilloscope and subsequently analyzes these leakage traces in conjunction with plaintexts and/or ciphertexts. Finally, we assume that the physical leakages of various intermediate data follow a similar leakage model without any assumptions about the distinguishability of leakage patterns. Additionally, unlike other non-collision SCAs, we do not assume that the leakage of the target implementation adheres to a specific leakage model.

4.2 The Curse of Trace Segmentation

Consider a leaking device using a fixed secret key. Cryptographic processes include a key byte k_i and a string of data d_i, considered as an n-bit word. Typically, $n = 8$ due to the byte-oriented nature of AES, which is widely examined in related works [ZBHV19, WPP22a, SM23]. The physical leakage of the key-related cryptographic operation, denoted as L_i, can be modeled by a leakage function ψ applied to intermediate data $\mathsf{I}(k_i, d_i)$ plus additive noise $Z_i \sim \mathcal{N}(0, \sigma^2)$, shown in Eq. (3) [BCO04].

$$L_i = \psi(\mathsf{I}(k_i,\ d_i)) + Z_i,\ d_i \in \mathcal{D}. \tag{3}$$

The purpose of a side-channel collision attack (SCCA) is to reveal a key difference Δ by identifying the smallest difference between L_i and L_j, denoting the leakage trace from two identical operations processing different data within one cryptographic function:

$$\Delta_{i,j} = \underset{\Delta}{\arg\min}\ \ \psi(\mathsf{I}(k_i, d_i)) - \psi(\mathsf{I}(k_j, d_j)) + (Z_i - Z_j),\ d_j = d_i \oplus \Delta, \tag{4}$$

where k_i and k_j are fixed and unknown values representing the secret subkeys of two trace segments. As the collision between $\mathsf{I}(k_i, d_i)$ and $\mathsf{I}(k_j, d_j)$ is not consistently present in a single encryption/decryption operation, an adversary must first guess the value of $\Delta = k_i \oplus k_j$, then extract L_i and L_j from different traces that satisfy $k_i \oplus d_i = k_j \oplus d_j$, finally making comparison of trace segments. However, as briefly mentioned in Section 1, this process can be problematic. First, a precise identification of target operations necessitates distinct leakage traces. If leakage patterns become indistinguishable due to factors like clock jitters or noise from other hardware components—a common issue in practical SCAs [WPP23]—the trace segmentation could become complex or even unfeasible. Although an extensive understanding of the source code [WPP23] or the ability to alter the source code to, e.g., place separators before and after the target process could be helpful for leakage identification, a stronger attack assumption is required, reducing the chance of a successful attack in a black-box setting following our threat model.

Next, A fundamental assumption of SCCA is that the target cryptographic operations have similar leakage patterns in two trace segments so they can be used for comparison and determine if the underlying intermediate data has a collision. Ideally, the target operation's leakage patterns are unique and noise-free, facilitating easy comparison for an adversary. However, even hardware with a naive cryptographic implementation can be subject to environmental noise or clock jitters. Consequently, the corresponding leakage

[1]Although the proposed method is named plaintext-based side-channel collision attack, we do *not* assume that the plaintext knowledge is mandatory to an adversary. In contrast, since we target the symmetric ciphers, the knowledge of either the plaintext or ciphertext enables the proposed attack.

patterns could differ in the time or amplitude domain even if the underlying cryptographic operation is the same. One might suggest that leakage randomness due to environmental noise could be compensated with a low-pass filter or averaging techniques, but it will also reduce information leakages. Even worse, clock jitter complicates this process and introduces variability in execution time. To counter this, an adversary could apply elastic alignment [vWWB11] or align on a specific feature subset. Still, these methods may ignore features pertinent to the target operations, potentially reducing attack effectiveness.

In conclusion, trace segmentation is the primary hurdle that hinders the broader application of SCCA-related methods. Simply operating on raw leakage features, such as the latest DL-SCCA [SM23], may not suffice to overcome this obstacle. An ideal approach should autonomously identify the target operation, extract relevant information, and generate a *representation* of leakages. Consequently, an adversary could compare these representations with various Δ guesses to determine the correct value.

4.3 Learning from Plaintext

In the context of profiling SCA, a profiling model $\mathsf{f}_{\boldsymbol{\theta}}$ encapsulates the relationship between the physical leakages and intermediate data modeled by a leakage model M. When an attack trace $\mathbf{t}$ is fed into $\mathsf{f}_{\boldsymbol{\theta}}$, the output is a probability vector corresponding to all possible intermediate data for the byte i:

$$\mathbf{p}_i = \mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t}), \tag{5}$$

Since we focus on a specific subkey byte, k_i and k are used interchangeably. Previous works show that plaintext can be severed as an input to augment profiling SCAs [HHO20, HGG18]. In SCCA, as the adversary is unaware of the key in use, $\mathsf{l}(k, d_i)$ cannot be leveraged to estimate $\boldsymbol{\theta}$. Since k remains constant on the target device, the label $\mathsf{l}(k, d_i)$ and d_i are bijective:

$$d_i \mapsto \mathsf{l}(k, d_i). \tag{6}$$

The bijectivity between the label $\mathsf{l}(k, d_i)$ and d_i depends on the choice of l. For example, Sbox output is often employed as a label function (and intermediate data) for AES attacks. Then, given a fixed key k_i, d_i and $\mathsf{Sbox}(d_i \oplus k_i)$ are bijective. If Eq. (6) holds, d_i and $\mathsf{l}(k, d_i)$ can be interchangeably mapped using mapping functions map parameterized by k:

$$\mathsf{l}(k, d_i) = \mathsf{map}_k(d_i). \tag{7}$$

As map is a fixed function that does not influence the profiling process, in the context of profiling models, we obtain:

$$\mathsf{f}_{\boldsymbol{\theta}_d}(\mathbf{t}) = \mathsf{map}'_k\left(\mathsf{f}_{\boldsymbol{\theta}}(\mathbf{t})\right), \tag{8}$$

where $f_{\boldsymbol{\theta}_d}$ refers to the profiling model trained with plaintexts, and map'_k is a fixed function that map the probability of intermediate data $\mathsf{l}(k, d)$ to its corresponding plaintext d, for all $d \in \mathcal{D}$. Given their bijectivity, the same leakage features are learned when profiling with d_i or $\mathsf{l}(k, d_i)$. As a result, the models based on these features, $f_{\boldsymbol{\theta}_d}$ and $\mathsf{map}'_k(\mathsf{f}_{\boldsymbol{\theta}})$, can be regarded as equivalent.

Eq. (8) indicates that a profiling model can be established based on plaintext to extract leakage information. We leverage this characteristic to form the proposed plaintext-based side-channel collision attack (PSCCA), which will be detailed in the next sections.

4.4 Plaintext-based Collision

The plaintext-based collision is performed in two steps: profiling and attacking. An overview is shown in Figure 1. For each subkey, the attacker trains a profiling model by associating a set of raw leakage traces $\mathbf{T}$—containing all intermediate data leakages—with the corresponding plaintext bytes. As described in Section 4.3, the model is trained based on the intermediate data's physical leakage, leveraging the bijectivity between intermediate data and plaintexts.

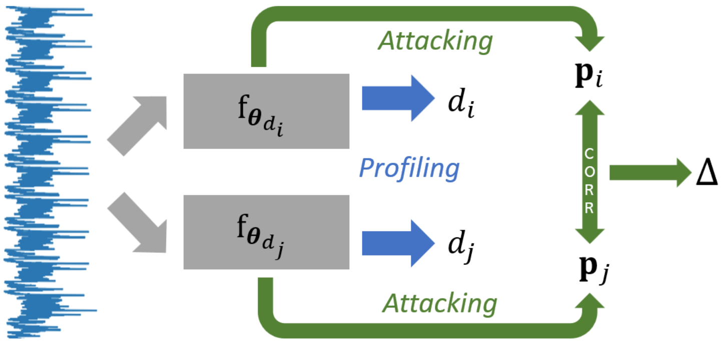

Attacking
$\mathsf{f}_{\boldsymbol{\theta}_{d_i}}$
d_i
$\mathbf{p}_i$
Profiling
CORR
Δ
$\mathsf{f}_{\boldsymbol{\theta}_{d_j}}$
d_j
$\mathbf{p}_j$
Attacking

Figure 1: Plaintext-based correlation.

In the attack phase, the traces used before for profiling are used again for attacking. Providing a subset of traces $\mathbf{t}$ from with known plaintext byte $d_i = u$, denoted as $\mathbf{t}_{d_i=u}$, the profiling model $\mathsf{f}_{\boldsymbol{\theta}_{d_i}}$ outputs an averaged probability vector $\mathbf{p}(d|\mathbf{t}_{d_i=u})$ for all possible d in $\mathcal{D}$. The highest probability score will be attributed to the correct plaintext byte value u. Although this information does not reveal anything to an attacker, the second highest probability will be attributed to a plaintext byte value v that leads to the most similar physical leakage of the intermediate data. Indeed, thanks to the bijectivity between the plaintext and the intermediate data, the top plaintext candidate d_{first} and the second-top plaintext candidate d_{second} outputted by the profiling model would imply the leakage from $\mathsf{l}(k, d_{first})$ and $\mathsf{l}(k, d_{second})$, respectively. The physical leakage from $\mathsf{l}(k, d_{first})$ should resemble that of $\mathsf{l}(k, d_{second})$. Otherwise, d_{second} would not be ranked as the second most probable. Consequently, the profiling model will automatically detect the leakage of the intermediate data without trace segmentation. Formally, following Eq. (3), we have

$$\psi\left(\mathsf{l}(k_i, d_i = u)\right) \sim \psi\left(\mathsf{l}(k_i, d_i = v)\right), \ \{u, v\} \in \mathcal{D}, \tag{9}$$

where $\sim$ denotes that two values are similar. The similarity shown in Eq. (9) depends on the similarity of the physical leakages between the target intermediate data and is, therefore, dependent on the key value. Now, let us expand $\mathbf{p}(d|\mathbf{t}_{d_i=u})$ in a vector considering all possible d_i values in $\mathcal{D}$, we have:

$$\mathbf{p}_i = \{\mathbf{p}(d|\mathbf{t}_{d_i=0}), \cdots, \mathbf{p}(d|\mathbf{t}_{d_i=255})\}, \ d \in \mathcal{D}. \tag{10}$$

For another target cryptographic calculation with plaintext byte d_j, we can repeat the same profiling-attacking process with $\mathbf{T}$ and $\mathsf{f}_{\boldsymbol{\theta}_{d_j}}$. We get:

$$\mathbf{p}_j = \{\mathbf{p}(d|\mathbf{t}_{d_j=0}), \cdots, \mathbf{p}(d|\mathbf{t}_{d_j=255})\}, \ d \in \mathcal{D}. \tag{11}$$

Recall Eq. (1): a side-channel collision satisfy $k_i \oplus k_j = d_i \oplus d_j$. If Δ equals to $k_i \oplus k_j$, we have:

$$\begin{aligned} \mathsf{l}(d_i \oplus k_i) &= \mathsf{l}(d_i \oplus k_i \oplus k_j \oplus k_j) \\ &= \mathsf{l}(d_i \oplus \Delta \oplus k_j) \\ &= \mathsf{l}(d_j \oplus k_j). \end{aligned} \tag{12}$$

Therefore, $\mathbf{p}_i$ holds for the following relationship given a correct Δ guess Δ^*:

$$\mathbf{p}_i = \mathbf{p}_{j,\Delta^*} = \{\mathbf{p}(d \oplus \Delta^*|\mathbf{t}_{d_j=0\oplus\Delta^*}), \cdots, \mathbf{p}(d \oplus \Delta^*|\mathbf{t}_{d_j=255\oplus\Delta^*})\}, \ d \in \mathcal{D}. \tag{13}$$

Eq. (13) encapsulates the core idea of this paper, denoting plaintext-based collision. If the plaintext used in two intermediate values is the same, the plaintext prediction vector from one operation can be converted to that of the other with Δ^*. The most likely $\Delta_{i,j}$ can be retrieved via Eq. (14).

$$\Delta_{i,j} = \underset{\Delta}{\operatorname{argmax}}\ \mathsf{corr}(\mathbf{p}_{i,\Delta},\ \mathbf{p}_j), \tag{14}$$

where corr represents the Spearman correlation [HK11] that evaluates the monotonic relationship between two inputs. The Spearman correlation offers more numerical stability than the Pearson correlation, as it has a high tolerance when the labels and leakage features are not linearly correlated. In this context, we are not solely interested in plaintext prediction (as they are already known) but in the score vector of all classes. For this reason, the Spearman correlation is best suited for this task.

4.5 Error Correction

Plaintext-based collision eliminates the requirements for leakage segmentation and direct comparison, thus could potentially lead to a better attack performance. However, it may introduce incorrect secret guesses due to, for instance, complicated leakage features or insufficient attack traces. Fortunately, the PSCCA predicts a key difference Δ, enabling an adversary to correct prediction outcomes with the help of predictions from other bytes. Specifically, considering $\Delta_{i,j} = k_i \oplus k_j$, the following relationship holds:

$$\begin{aligned} \Delta_{i,j} &= (k_i \oplus k_a) \oplus (k_j \oplus k_a) \\ &= \Delta_{i,a} \oplus \Delta_{j,a},\ a \notin \{i,j\}. \end{aligned} \tag{15}$$

Here, k_a denotes an arbitrary key byte, and $\Delta_{i,j}$ can be expressed as the xor of two key differences $\Delta_{i,a}$ and $\Delta_{j,a}$ that involve an additional byte. In cases where $\Delta_{i,j}$ is wrongly predicted while $\Delta_{i,a}$ and $\Delta_{j,a}$ are accurate, we can ascertain the true value of $\Delta_{i,j}$ using other predictions.

We ensemble the error correction scheme into our attack. The error correction method is inspired by the Low-Density Parity Check (LDPC) decoding method [GS12]. Formally, let the outcome of Eq. (14) denote the probability of each $\Delta \in \mathcal{K}$ being chosen as Δ^*, symbolized as $\mathbf{p}_{i,j}(\Delta)$. We can then correct $\mathbf{p}_{i,j}(\Delta)$ using the following expression:

$$\mathbf{p}_{i,j}(\Delta) = \mathbf{p}_{i,j}(\Delta)\ \cdot \prod_{a \notin \{i,j\}} \max_{\beta}(\mathbf{p}_{i,a}(\beta) \cdot \mathbf{p}_{j,a}(\beta \oplus \Delta)),\ \Delta \in \mathcal{K},\ \beta \in [0, 255]. \tag{16}$$

The right-hand side of the equation comprises two components: the original probability $\mathbf{p}_{i,j}(\Delta)$, and the composite probability of two Δ values that include a third key byte. We choose the maximum combined probability for all possible β, diverging from the original formulation, which proposed summation [GS12]. Ideally, when $\beta = \Delta_{i,a}$ and $\beta \oplus \Delta = \Delta_{j,a}$, $\mathbf{p}_{i,a}(\Delta_{i,a}) \cdot \mathbf{p}_{j,a}(\Delta_{j,a})$ will yield the maximum value. The maximum combined probability more accurately reflects the likelihood of each Δ guess, thereby enhancing the effectiveness of the attack, as demonstrated in the following sections.

5 Attack Scheme

Our attack scheme consists of three steps: 1) profiling with different plaintext bytes; 2) predicting the key difference Δ; and 3) error correction. An adversary will always execute the first two steps. Error correction is only performed when incorrect predictions are detected, as detailed in the later paragraphs.

To speed up the learning process, instead of profiling each plaintext byte separately in the first step, we employ multi-task learning (MTL) to profile all plaintexts simultaneously. MTL is a well-studied machine learning technique that trains multiple learning tasks in parallel [Car97, Rud17]. The main idea of MTL is to learn multiple tasks together to improve the learning of each task by leveraging information shared among related tasks [Car97, ZY21]. For its side-channel applications, previous works from Maghrebi and Masure et al. use MTL to attack multiple secret shares, leading to an enhanced attack performance compared with the single-task learning [Mag20, MS23]. For PSCCA, since the correct Δ is guessed based on the correlation of two prediction vectors. We leverage the MTL to acquire prediction vectors for all (16 for AES) intermediate data in one goal and then make Δ guesses. Note that the proposed PSCCA is not limited to deep learning and can be applied with other profiling methods, such as Gaussian templates. In such cases, each plaintext byte needs to be profiled separately. We demonstrate the Gaussian templates-based PSCCA in the later paragraph.

Figure 2 illustrates the proposed deep learning architecture. The shared layers are responsible for processing the leakage and extracting general features. Then, several task-specific layers are constructed, forming subbranches for each plaintext byte. The shared structure encourages the profiling model to learn the general features of different tasks, potentially leading to improved performance and resilience to overfitting. Compared with DL-SCCA, which relies on a dedicated reference trace segment, we use raw side-channel traces containing all intermediate data leakages and let the MTL framework locate the features of interest, thus saving the efforts of trace segmentation. Compared with most non-profiling SCAs, such as CPA and DDLA, the proposed attack is more computationally efficient, as it only requires training a single model instead of 256.

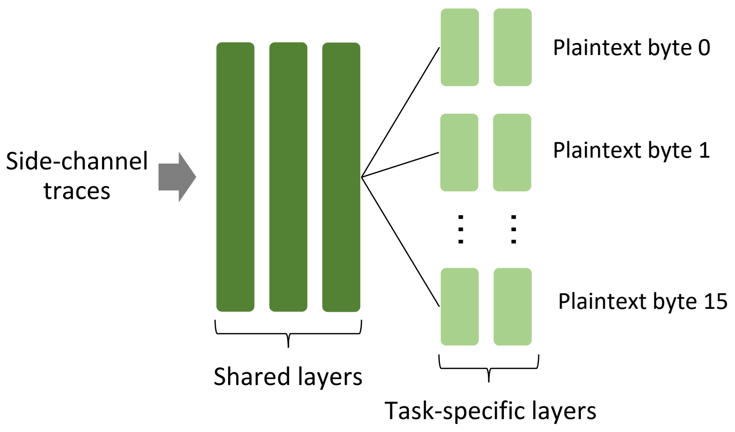


Figure 2: Deep learning architecture of plaintext-based SCCA.

Once the profiling step is completed, we calculate $\mathbf{p}(\Delta)$ for each pair of key bytes k_i and k_j using Eq. (14). Finally, error correction is applied if any of the predicted key difference is incorrect. It is straightforward to detect the presence of incorrect predictions in PSCCA. Assuming an PSCCA targeting 16 bytes $k_0, k_2, \cdots, k_{15}$, a successful secret recovery would satisfy Eq. (17). If the equation does not hold, the error correction method presented in Section 4.5 is applied.

$$\arg\max_{\Delta} \mathbf{p}_{0,15}(\Delta) = \arg\max_{\Delta} \mathbf{p}_{0,1}(\Delta) \oplus \arg\max_{\Delta} \mathbf{p}_{1,2}(\Delta) \oplus \cdots \oplus \arg\max_{\Delta} \mathbf{p}_{14,15}(\Delta). \quad (17)$$

To provide a clear formulation of PSCCA, the pseudocode of the attack is shown in Algorithm 1. Line 1 represents the building of the profiling model using plaintexts. The Δ guess is obtained from Line 2 to Line 6. As shown in Section 4.4, the plaintext-based collision requires d_i and d_j in the same order, e.g., $d_i = d_j$. Therefore, as shown in Line 4, it is important to sort the traces to ensure that each Δ guess satisfies this condition. Lines 7 and 8 are dedicated to the error correction step.

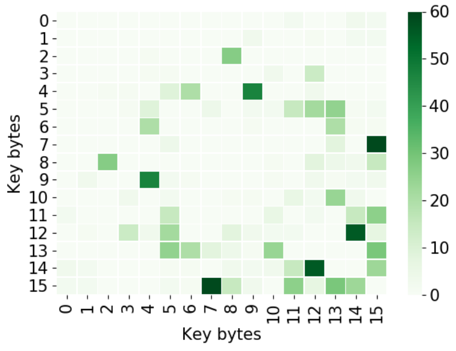
Algorithm 1: Plaintext-based DL Collision Attack

```
Input  : Traces $\mathbf{T}$, Plaintext $d$, byte index $i$, byte length $L$, Profiling model PM
Output : Key difference $\Delta_{i,j},\ i,j \in L$
1 Train model PM $\leftarrow$ Train($\mathbf{T}, d$);
2 while $i < L - 1$ do
3     while $i < j$ and $j < L$ do
4         Sort $\mathbf{T}$ to make $d_i = d_j$;
5         $\mathbf{p}(d|\mathbf{t}_{d_i}), \mathbf{p}(d|\mathbf{t}_{d_j}) \leftarrow$ PM($\mathbf{T}$), $d \in \mathcal{D}$;
6         $\mathbf{p}_{i,j}(\Delta) \leftarrow$ corr($\mathbf{p}(d|\mathbf{t}_{d_i}),\ \mathbf{p}(d \oplus \Delta|\mathbf{t}_{d_j \oplus \Delta})$), $\Delta \in \mathcal{K}$;
7 if $\arg\max_{\Delta} \mathbf{p}_{i,j}(\Delta)$ is incorrect then
8     $\Delta_{i,j} \leftarrow$ Error_Correction($\mathbf{p}_{i,j}(\Delta)$)
```

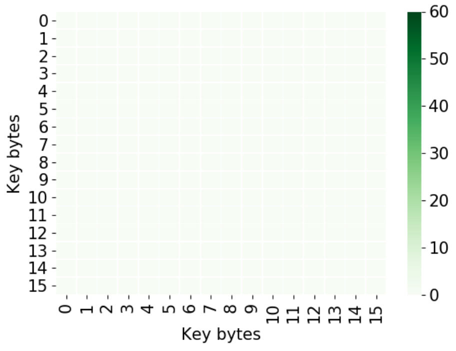
To demonstrate the effectiveness of PSCCA, we present attack results using *simulated* datasets as described by Eq. (18).

$$leakage = \mathsf{Sbox}(d_i \oplus k_i) + Z, \quad i \in [0, 15], \tag{18}$$

where d_i and k_i denote a random plaintext byte and a fixed key byte, respectively. All features are manipuated with noise $Z \sim \mathcal{N}(0, \sigma^2)$. To ensure the noise has the same effect on each test case, we normalize the leakage between 0 and 1. A total of 5 000 traces were simulated and used for profiling. As mentioned in Section 5, PSCCA can be applied to various profiling methods. In this case study, we utilize the Gaussian template as the benchmark due to its simplicity in terms of hyperparameter tuning. The profiling of each plaintext byte is performed separately.



(a) Before error correction.



(b) After error correction.

Figure 3: Rank table of Δ before and after the error correction with the simulated dataset.

We consider two test cases with different levels of noise represented by two σ values: 0.05 and 0.3. In the low-noise test case, the key difference between all key bytes is successfully recovered, highlighting the effectiveness of selective recovery with plaintext learning and plaintext-based collision. When the σ value is increased to 0.3, as shown in Figure 3a, we observe some unsuccessful Δ recoveries for certain key bytes using the plaintext-based collision method. Fortunately, leveraging the Δ dependencies between different key bytes, all key differences are successfully recovered after applying the error correction, as illustrated in Figure 3b. This demonstrates the efficiency of the proposed error correction method in challenging scenarios with higher levels of noise.

6 Experimental Results

In this section, we evaluate the effectiveness of our proposed attack strategy using public side-channel datasets. We aim to perform a plaintext-based side-channel collision attack and recover all 16 subkeys. To achieve this, we employ a deep learning model shown in Section 5 that offers the advantage of attacking all subbytes simultaneously.

The overall architecture of the model is depicted in Figure 2. Specifically, we adopt a Convolutional Neural Network (CNN) as detailed in [PWP22].[2] The network architecture is shown in Table 1. A convolution block consists of a convolution layer, a pooling layer, and a batch normalization layer. The convolution block is used as shared layers; the remaining dense layers are assigned to each subbranch.

Table 1: Deep learning architectures used in the experiments.

Layer	**Kernel number/size**	**Pooling stride/size**	**Neurons**
Convolution block	40/20	2/2	-
Dense	-	-	200
Dense	-	-	200

Regarding other hyperparameters, we use Scaled Exponential Linear Unit ($Selu$) as the activation function for each layer [KUMH17], except for the final layer, which employs $Softmax$ [Bri90] that convert a vector of n real numbers into a probability distribution of n possible outcomes. The DL model is trained for 100 epochs, meaning that the DL training will iterate 100 times to learn and adjust its parameters based on the input data. The batch size is set to 768. A hyperparameter study on training epoch and batch size are provided in Section 6.4. Data augmentation is implemented using a random translation layer following the input layer, randomly shifting the leakage measurements within a predefined level of 5. A comprehensive analysis of data augmentation and desynchronization robustness are presented in Section 6.3.

The deep learning model and its hyperparameters remain constant across all attack methods. Although customizing the model and tuning hyperparameters for each dataset and method may enhance attack performance, such variables can significantly complicate our benchmarking conclusions by introducing model complexity and increasing training effort. Moreover, even after customizing the model and conducting hyperparameter tuning, it is impossible to guarantee the model's generality for a specific scenario. Since the main contribution of this paper lies in introducing a new attack method, the impact of selecting optimal solutions can be disregarded by opting for a model with acceptable performance.

To ensure reliable and robust evaluation, each attack scenario considered in this section is independently tested ten times to reduce the influence of random factors (e.g., random weight initialization) on attack performance [WPP22b]. The results are then averaged to represent the overall performance of each attack method.

We consider three publicly available datasets: ASCAD_F, ASCAD_R [BPS+20], and CHES_CTF[3]. The trace pattern for the first two datasets varies due to different measurement configurations[4]. All contain side-channel measurements from first-order

[2]The deep learning models were implemented using Python 3.6 and TensorFlow library version 2.6.0. Training algorithms were executed on a Nvidia GTX 1080TI GPU, managed by Slurm workload manager version 19.05.4.

[3]https://chesctf.riscure.com/2018/news

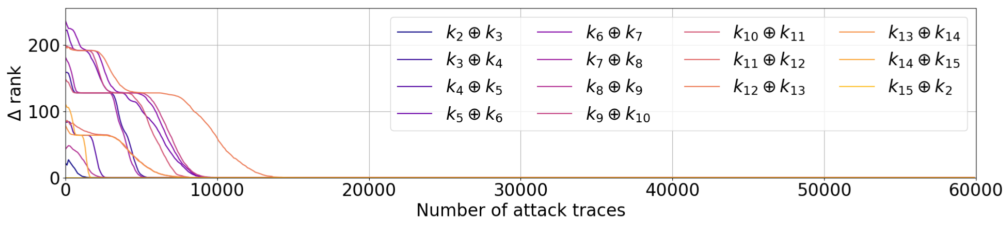
[4]https://github.com/ANSSI-FR/ASCAD/issues/13

masked AES software implementations. The raw traces contain a huge number of sampling points, and handling such large intervals of side-channel measurements can be time-consuming. To address this, we adopt the resampling technique with a resampling window[5] 80, aligned with the approach outlined in [PWP22]. For the first two datasets, we employ 60,000 traces for the attack; for the last one, our experiments consider 40,000 traces for the Δ recovery. As mentioned in Section 4.4, the same trace sets are used for profiling and attacking.

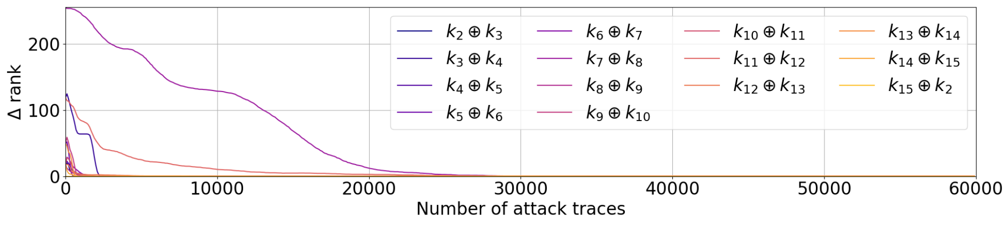
6.1 Performance Analysis

This section provides a detailed analysis of the attack performance on the datasets under consideration. We present the results separately with and without the error correction.

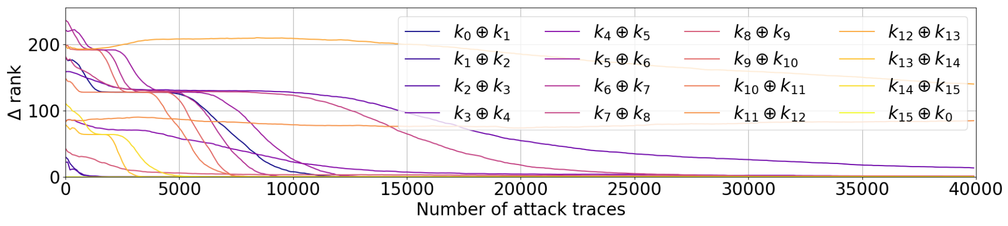
Figure 4 illustrates the Δ ranks for different combinations of key bytes. Note that the ASCAD_F and ASCAD_R datasets' first two bytes are unmasked and, therefore, excluded from the attack due to their relative simplicity. As a result, all key bytes depicted in Figure 4 are mask-protected. The results show that the proposed plaintext-based collision method successfully recovers all key differences for ASCAD_F and ASCAD_R, thereby reducing the remaining key space to 2^8. For the CHES_CTF dataset, 13 of the 16 tested Δ values are successfully recovered.



(a) ASCAD_F.



(b) ASCAD_R.

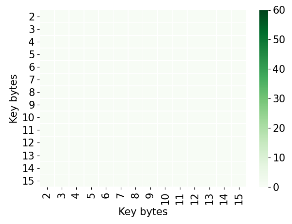


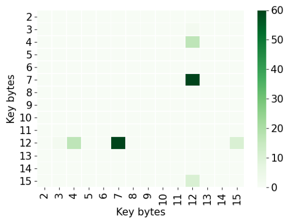
(c) CHES_CTF.

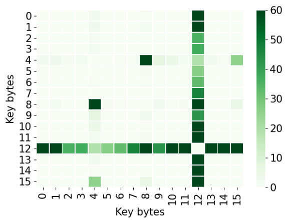
Figure 4: Δ rank for each dataset (no error correction).

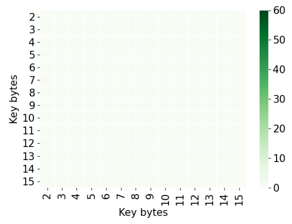
A comprehensive overview of the attack performance for each dataset is depicted in Figure 5, which presents the Δ rank for all possible subkey combinations. Our method

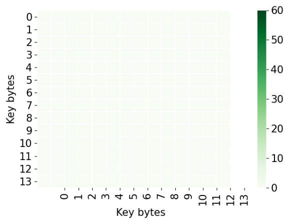
[5]The resampling aims to reduce the dimensionality of the traces; a resampling window indicates the number of trace samples averaged into a single sample.


(a) ASCAD_F: before error correction.


(b) ASCAD_R: before error correction.


(c) CHES_CTF: before error correction.


(d) ASCAD_F: after error correction.


(e) ASCAD_R: after error correction.

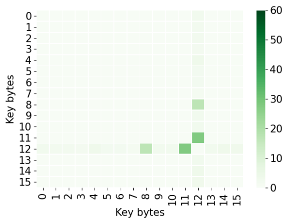

(f) CHES_CTF: after error correction.

Figure 5: Rank table of Δ before and after error correction.

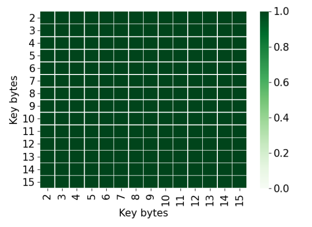
successfully breaks ASCAD_F with all possible subkey combinations, even without error correction. For ASCAD_R and CHES_CTF, the proposed method recovers a significant portion of Δ. As expected, the attack results improved further after applying the error correction method. Interestingly, the 12th subkey of CHES_CTF consistently exhibits mediocre performance in the Δ calculation. Indeed, the physical leakage of intermediate data related to key byte 12 differs significantly from the rest, leading to all key difference delta related to this key byte being unrecoverable. Fortunately, when error correction is applied, the Δ rank experiences a significant improvement.

An overview of the success rate (SR), representing the percentage of successfully recovered $\Delta_{i,j}$ over ten attacks, and overall success rate (OSR) for all key differences, is provided in Figure 6. The error correction method contributes to a higher SR for the ASCAD_R and CHES_CTF datasets. Note that achieving an OSR of 100% is not always necessary for PSCCA to recover Δ. One could perform majority voting by counting the occurrences of each Δ candidate being the most likely Δ over multiple attacks, and the one with the highest count number can be considered the correct key. Accordingly, the OSR for CHES_CTF can reach 100% as well. For instance, as shown in Figure 6f, the SR of $\Delta_{7,12}$ is 60%, meaning that the correct candidate has been selected six times over ten attacks. In this case, an adversary is confident that the correct $\Delta_{7,12}$ is retrieved by choosing the Δ candidate that happens the most frequently.

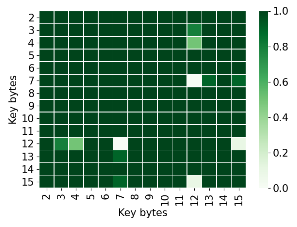
6.2 Performance Benchmark

To showcase the superior attack capabilities of PSCCA, this section includes a benchmark comparison with DL-SCCA, using ASCAD_F as the target dataset. All attack settings are identical to the original paper [SM23]. Additionally, a state-of-the-art non-profiling attack method, DDLA[6], is incorporated into our benchmark, as detailed in Table 2. For DL-SCCA, the choice of reference byte for model training significantly influences the number of bytes recovered (refer to [SM23], Figure 8). Hence, both the worst and best-case overall success rates (OSR) are reported and separated by '/'. Conversely, the performance

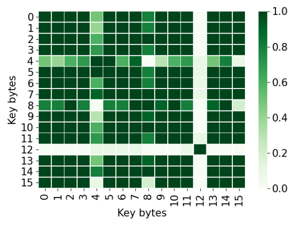
[6]The LSB leakage model is used as it performs better than HW based on our preliminary tests.



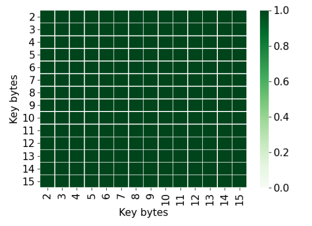
(a) ASCAD_F: before error correction (OSR=100%).



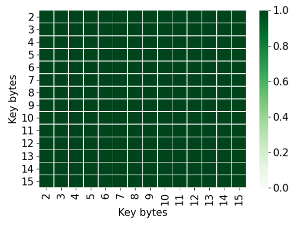
(b) ASCAD_R: before error correction (OSR=96%).



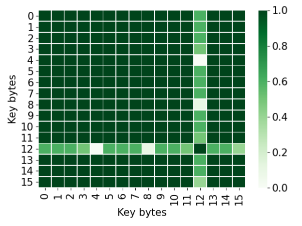
(c) CHES_CTF: before error correction (OSR=78%).



(d) ASCAD_F: after error correction (OSR=100%).



(e) ASCAD_R: after error correction (OSR=100%).



(f) CHES_CTF: after error correction (OSR=88%).

Figure 6: Success rate table of Δ before and after error correction.

of PSCCA is presented as OSR before and after applying error correction.

Table 2: Performance comparison between DDLA, DL-SCCA, and our method.

	Attack setting	Samples	OSR	Computation time
DDLA	Known leakage offset	Refined leakage interval	100%	3h 14min
DDLA	Black box	Full Set	93%	5h 23min
DL-SCCA	Known leakage offset	Full Set	0%/100%	23min
DL-SCCA	Black box	Full Set	0%/76%	3h 47min
This work	Black box	Full Set	100%/100%	5min

Regarding attack efficiency, PSCCA surpasses DL-SCCA in both scenarios, even without error correction. Specifically, DL-SCCA only matches PSCCA's performance when an optimal reference byte with a known leakage offset is used. Under black box conditions, DL-SCCA's highest OSR falls to 76%. In the worst-case scenario, it fails to recover any key byte difference. In contrast, PSCCA, tailored for black box scenarios, achieves a 100% OSR without error correction. DDLA also achieves a 100% OSR with a refined time interval, as per the original study. However, its OSR decreases to 92% when applied to the entire dataset. Notably, DDLA faces two drawbacks compared to PSCCA: it relies on an accurate leakage model, with our tests showing that LSB outperforms the HW model, and it necessitates training 256 models for each byte, which increases computational complexity. In addition, we also examine the number of attack traces required to achieve a key difference of zero [7]. PSCCA demonstrates greater robustness compared to its DL-SCCA counterpart. For ASCAD_F, PSCCA needs, on average, less than 500 traces to recover all key bytes, which is significantly more efficient than DL-SCCA, which largely depends on selecting the appropriate reference key byte for training. Indeed, less than 600 traces are required for DL-SCCA in the best setting, i.e., attacks with the best reference byte. Unfortunately, if the reference byte is poorly selected, no key byte can be recovered with DL-SCCA. The same conclusion is applied to the other two datasets. Consequently, DL-SCCA only

[7] Both PSCCA and DL-SCCA utilize the same set of traces for both profiling and attacking.

achieves parity with PSCCA when the optimal reference key byte is used in a known offset setting. DLLA, in contrast, requires fewer traces (around 350 and 490 for the refined and full sets, respectively). However, it is less directly comparable to our approach as SCCA, which evaluates leakage fragments, substantially differs from non-profiling attacks that focus on a single intermediate data.

Our method is more computationally efficient than DL-SCCA and DDLA. It requires only five minutes to recover all key differences Δ. In contrast, DDLA, attacking all bytes in parallel, requires more than three hours to complete an attack on a single byte over a refined leakage interval, and this duration almost doubles for the full dataset. Both DL-SCCA and PSCCA attack full traces. However, a black box attack using DL-SCCA takes more than five hours, making our method significantly more efficient.[8]. Moreover, PSCCA conducts simultaneous attacks of 16 bytes, completing each byte's attack in under a minute. This performance is on par with state-of-the-art methods that require meticulous hyperparameter adjustments for size reduction [PWP22].

6.3 Data Augmentation and Desynchronization Robustness

Preprocessing of leakage measurements is essential for an efficient attack. For PSCCA, in addition to normalizing the data, data augmentation plays a pivotal role in the success of the proposed attack. The data augmentation is realized by randomly shifting the input traces within a threshold.. This section studies the impact of different data augmentation levels (i.e., the maximum range of the traces desynchronization) on attack performance. Since our data argumentation method is equivalent to adding desynchronization to the traces, this section also showcases the robustness of PSCCA against leakage desynchronization.

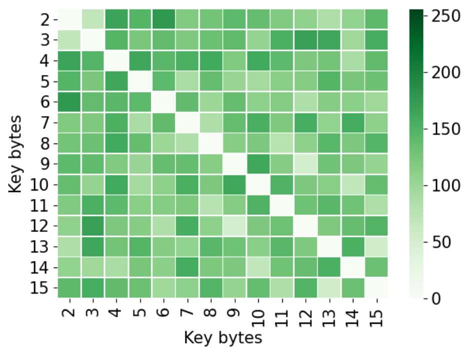
Table 3: The influence of data augmentation/desynchronization on OSR.

	0	5	10	20	50
ASCAD_F	100%	100%	100%	28%	8%
ASCAD_R	7%	96%	99%	94%	65%
CHES_CTF	16%	78%	48%	15%	6%

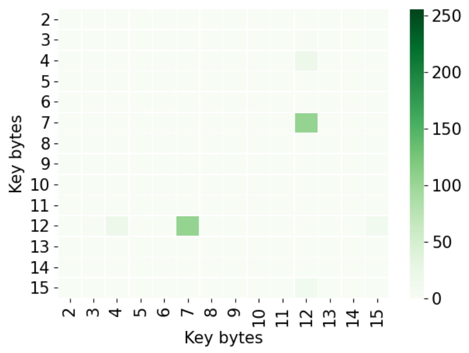
As shown in Table 3, when the desynchronization level is zero, PSCCA achieves a 100% OSR only for the ASCAD_F dataset. However, a significant performance improvement is observed when augmenting the datasets (introducing random shifts). For instance, as shown in Figure 7, PSCCA has a mediocre performance when the desynchronization level is zero; when increased to 5, most of the key differences can be recovered. These observations underscore the role of data augmentation in enhancing the effectiveness of the proposed method. As a regularization technique, data augmentation helps prevent the profiling model from fixating on specific features, enabling it to focus on global features instead. In the context of side-channel attacks, where data leakages exist in only a few features, such techniques prevent the model from overfitting irrelevant features. Looking at it differently, rather than diminishing the attack performance, a not-too-large desynchronization in leakages could potentially enhance the attack effectiveness of PSSCA.

Employing a high level of desynchronization can have adverse effects, leading to a decline in attack performance. As the desynchronization level reaches 20, there is a noticeable deterioration in attack performance across various configurations. This high desynchronization level increases the complexity of fitting the model to the leakage as the timing of the leakages becomes more random. Consequently, longer training periods or larger models may be required, increasing computational effort. When increasing to

[8] [SM23] reports a computation time of 1h 57min with a Nvidia 2080TI GPU; our attack is significantly more computationally efficient even with a lower-end Nvidia GTX 1080TI GPU.



(a) Desynchronization: 0.



(b) Desynchronization: 5.

Figure 7: Rank table of Δ for the ASCAD_R with different desynchronization levels.

desynchronization 50, OSR drops considerably. Although one could fine-tune the model to improve the model resistance to desynchronization, a high desynchronization level influences the attack performance of PSCCA, similar to other SCA methods.

On the other hand, error correction would be a suitable solution to overcome the desynchronization effect. Table 4 presents the effect of desynchronization variation with error correction enabled. A noticeable increase in OSR is observed in the settings with high desynchronization, except when the desynchronization level is zero. This limitation arises because the error correction method relies on key differences involving a third byte. The error correction method becomes non-functional if these key differences are also incorrect. In such scenarios, as outlined at the end of Section 6.1, a potential workaround could be applying an ensemble method with the majority voting. A similar conclusion can be drawn with an overly high desynchronization level (e.g., desynchronization of 50). Even though the error correction method cannot improve OSR (except for ASCAD_R, where the performance is still acceptable, a consequence of higher OSR before error correction), majority voting could be a promising alternative.

Table 4: The influence of error correction on OSR with different desynchronization levels.

	0	5	10	20	50
ASCAD_F	100%	100%	100%	49%	8%
ASCAD_R	7%	100%	100%	100%	92%
CHES_CTF	16%	88%	67%	31%	6%

6.4 Hyperparameters Study

6.4.1 Batch Size

Batch size in a deep learning model refers to the number of examples (input-output pairs) used in a single training iteration. This parameter plays a crucial role in shaping the behavior and overall performance of the deep learning model. Table 5 illustrates that larger batch sizes generally improve attack performance in the proposed method. This can be attributed to larger batch sizes providing a more accurate gradient estimation. Indeed, in deep learning, the gradients guide the model's learning process by indicating the direction in which the model parameters should be updated. With a larger batch size, the gradient estimation becomes more reliable as it incorporates information from more data points, potentially leading to better learning and performance.

Table 5: Hyperparameter evaluation for batch size.

	128	256	512	768	1 024
ASCAD_F	31%	7%	100%	100%	100%
ASCAD_R	7%	95%	97%	96%	97%
CHES_CTF	6%	72%	73%	78%	81%

Moreover, larger batch sizes enable more efficient utilization of computational resources, particularly GPUs, which tend to exhibit optimal performance when processing computations in larger blocks. In our experiments, for example, a batch size of 1 024 completed tests on all datasets four times faster than a batch size of 128, highlighting the advantage of utilizing larger batch sizes in terms of computational efficiency.

Consistent with the preceding section, error correction generally leads to an increase in the Overall Success Rate (OSR) across various test cases, except when the OSR is already low. Notably, the CHES_CTF dataset consistently attains an 88% OSR when the batch size exceeds 256. As explained in Section 6.1, the performance is subpar when calculating the key difference involving byte 12 of the CHES_CTF key compared to other scenarios. For instance, correcting $\Delta_{1,12}$, for instance, involves adjusting both $\Delta_{1,x}$ and $\Delta_{x,12}$, where x denotes a random byte distinct from 1 or 12. Since $\Delta_{x,12}$ is also incorrect, the resulting corrections would be flawed. In such cases, the ensemble method proposed towards the end of Section 6 emerges as a potential solution for recovering the correct key.

Table 6: The influence of error correction on the batch size variation.

Batch size	128	256	512	768	1 024
ASCAD_F	42%	7%	100%	100%	100%
ASCAD_R	7%	100%	100%	100%	100%
CHES_CTF	6%	88%	88%	88%	88%

6.4.2 Training Epochs

Simply increasing the number of training epochs does not always improve a deep learning model's ability to map inputs to outputs. It can lead to overfitting, where the model struggles to generalize to new data. Table 7 illustrates the performance changes in our method with different epoch counts. Training with just 50 epochs is often enough for most setups. More epochs generally yield stable results, except for the CHES_CTF dataset, where increased training leads to a decline in attack performance. This suggests that the model trained on this dataset is more prone to overfitting. PSCCA remains robust to changes in training epochs across tested datasets.

Table 7: Study on the influence of the training epoch.

Training epoch	50	100	200	400	600
ASCAD_F	100%	100%	100%	100%	100%
ASCAD_R	91%	96%	95%	96%	95%
CHES_CTF	71%	78%	73%	70%	67%

To summarize, our analysis highlights the impact of hyperparameter tuning. We emphasize that we employ a single model to attack all considered datasets, achieving consistent performance. This underscores the simplicity of our model's hyperparameter tuning and underlines the robustness to PSCCA, making it a reliable SCCA solution across different attack scenarios.

7 Discussion

The Side-channel Collision Attack (SCCA) distinguishes itself from profiling and non-profiling side-channel attacks by directly comparing leakage features to identify side-channel collisions. SCCA enables the direct cancellation of high-order masks, whereas non-collision attacks often require additional knowledge (e.g., details about the masking scheme) or additional efforts (e.g., leakage recombination). While classical profiling and non-profiling attacks can break certain masking implementations [ZBHV19, WPP24], some, such as affine masking, remain resistant to classical attacks in a black-box setting [BS20]. The unique approach of collision attacks makes the attack particularly efficient and straightforward, as demonstrated in recent studies [WPP23]. However, SCCA has its own set of disadvantages. SCCA commonly requires a thorough understanding of the underlying code for identifying the targeted operation and achieving accurate trace segmentation [WPP23]. As discussed in Section 4.2, this requirement presents challenges for both SCCA and DL-SCCA, including issues of portability, where trace segments must be perfectly aligned, and the targeted operations should exhibit similar leakage patterns. Our approach deviates from conventional collision attacks, including its DL variant (DL-SCCA), by using probability vectors covering all possible label hypotheses. This approach, detailed in Eq. (9), demonstrates that both correct and incorrect labels can reveal essential key dependencies, thereby boosting the effectiveness of our attack strategy. Indeed, as highlighted in [BK02, WWK$^+$23], integrating incorrect labels into the analysis has been shown to enhance the performance of both non-profiling and profiling attacks.

Our method extends beyond the aforementioned benefits, completing the PSCCA with an error correction scheme that renders it suitable for real-world application. First, it is possible that the two intermediate data have very different physical leakages, challenging the learning process. The proposed error correction method leverages different deltas' dependencies to correct prediction errors, leading to more robust performance. Second, our method introduces an error correction scheme by leveraging properties inherent to SCCA and the capability to attack all 16 bytes simultaneously. This allows an attacker to determine the correctness of a key byte and correct errors based on other key bytes, thus potentially guessing more key bytes correctly.

Despite the benefits, it's crucial to preprocess the leakage traces for the attack to be effective. Our approach relies on the bijective relationship between plaintext and intermediate data to understand the physical leakage of the latter. If there are leakages from the plaintext, the profiling model could be misled, hindering its ability to learn about the intermediate data leakage accurately. Simple Power/EM Analysis (SPA/SEMA) techniques can be employed to eliminate leakages related to plaintext loading. In cases of more basic implementations, such as devices with low security that directly load plaintext, a simple yet effective solution might be to conduct a preliminary correlation analysis on the plaintext and then eliminate or mask the relevant leaking features.

PSCCA capitalizes on the bijective relationship between plaintext and the targeted intermediate data to derive leakage features from raw traces automatically. This inherent property opens up the possibility for various countermeasures to undermine the proposed attack's effectiveness. For instance, implementing frequent re-keying strategies, such as changing session keys, could disrupt the bijective link, thereby challenging PSCCA's efficiency. Yet, the efficacy of this countermeasure largely hinges on the frequency of re-keying, indicating that it might not be entirely resistant to such attacks.

Hiding countermeasures could be employed to defend PSCCA. While these may not completely disable the attack, they could substantially increase the number of attack traces required for feature extraction. A similar effect is observed with shuffling, a prevalent countermeasure known to compromise both SCCA [WPP23] and potentially DL-SCCA. High-order masking might raise the difficulty level of the attack. However, given the demonstrated ability of DL to circumvent masking countermeasures and SCCA's capacity to neutralize masking shares [WPP23], the actual effectiveness of this countermeasure varies depending on actual implementations.

We do not test the dataset with parallel computation, so no definitive conclusion can be drawn here. Still, we emphasize that the PSCCA's attack performance relies on the profiling model's feature extraction capability. Generally, suppose the parallel implementation is breakable in a profiling attack setting targeting the intermediate data directly. Then, we expect our attack also to work, as it targets the plaintext bijective to the intermediate data.

From a security evaluator's perspective, our proposed PSCCA framework offers several key benefits for simplifying and enhancing the side-channel analysis process. First, it reduces the expertise and detailed knowledge required about the target by breaking down the SCCA process into three straightforward steps: training, predicting, and comparing. Plaintext labeling allows evaluators to automatically pinpoint targeted leakages through a classifier, effectively extracting relevant information from raw data. By utilizing Multi-Task Learning (MTL), evaluators can train a single model to attack all bytes simultaneously, significantly reducing the time needed for a comprehensive evaluation. Additionally, the collision comparison is performed solely on distilled plaintext vectors, which removes the need for trace segmentation and avoids the related portability challenges. Finally, including an error correction step enables evaluators to more easily assess and validate the attack's effectiveness, making the overall evaluation process more streamlined and reliable.

8 Conclusions and Future Work

This paper presents a novel plaintext-based side-channel collision attack, eliminating the need for trace segmentation and leakage comparison. We also propose an error correction scheme to enhance the accuracy of the correction key difference prediction, thereby improving the overall attack performance. Next, we employ multi-task learning to attack all subkey bytes simultaneously, resulting in efficient key difference recovery. By applying our framework to three masked AES datasets, we achieve profiling attack results significantly surpassing the state-of-the-art DL-SCCA and DDLA in attack performance and computation efficiency. Importantly, our approach demonstrates its generality across various attack scenarios, as minimal effort is required for hyperparameter tuning.

For future work, it would be interesting to explore methods for retrieving the correct key difference, such as refining the Spearman correlation technique. This could lead to a reduction in the number of attack traces required. Secondly, we plan to continue enhancing the error correction method, potentially incorporating ensembles or other techniques. Developing a more robust deep learning model could also strengthen the relationship between the input and each task. For instance, direct connections between the input layer and the model's subbranch may reduce reliance on the main branch and improve feature extraction capabilities. Lastly, given that our method relies on the bijectivity of the plaintext and intermediate data, it is relevant to explore its usages on other ciphers where bijectivity does not hold (e.g., DES Sbox).

Acknowledgments

This work received funding in the framework of the NWA Cybersecurity Call with project name PROACT with project number NWA.1215.18.014, which is (partly) financed by the Netherlands Organisation for Scientific Research (NWO). Additionally, this work was supported in part by the Netherlands Organization for Scientific Research NWO project DISTANT (CS.019).

References

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. 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. doi:10.14722/ndss.2020.24390.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 16–29. Springer, 2004. doi:10.1007/978-3-540-28632-5_2.

[BK02] Régis Bevan and Erik Knudsen. Ways to enhance differential power analysis. In *International Conference on Information Security and Cryptology*, pages 327–342. Springer, 2002. doi:10.1007/3-540-36552-4_23.

[Bog07] Andrey Bogdanov. Improved side-channel collision attacks on aes. In *Selected Areas in Cryptography: 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers 14*, pages 84–95. Springer, 2007. doi:10.1007/978-3-540-77360-3_6.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptographic Engineering*, 10(2):163–188, 2020. doi:10.1007/s13389-019-00220-8.

[Bri90] John S Bridle. Probabilistic interpretation of feedforward classification network outputs, with relationships to statistical pattern recognition. In *Neurocomputing: Algorithms, architectures and applications*, pages 227–236. Springer, 1990. doi:10.1007/978-3-642-76153-9_28.

[BS20] Olivier Bronchain and François-Xavier Standaert. Side-channel countermeasures' dissection and the limits of closed source security evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 1–25, 2020. doi:10.46586/tches.v2020.i2.1-25.

[Car97] Rich Caruana. Multitask learning. *Machine learning*, 28:41–75, 1997. doi:10.1007/978-1-4615-5529-2_5.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002. URL: https://doi.org/10.1007/3-540-36400-5_3, doi:10.1007/3-540-36400-5_3.

[DLH+22] Ngoc-Tuan Do, Phu-Cuong Le, Van-Phuc Hoang, Van-Sang Doan, Hoai Giang Nguyen, and Cong-Kha Pham. 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)*, pages 245–250, 2022. doi:10.1109/ATC55345.2022.9943024.

[GS12] Benoît Gérard and François-Xavier Standaert. Unified and optimized linear collision attacks and their application in a non-profiled setting. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 175–192. Springer, 2012. doi:10.1007/978-3-642-33027-8_11.

[HDD22] Van-Phuc Hoang, Ngoc-Tuan Do, and Van Sang Doan. Efficient non-profiled side channel attack using multi-output classification neural network. *IEEE Embedded Systems Letters*, pages 1–1, 2022. doi:10.1109/LES.2022.3213443.

[HGG18] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Profiled power analysis attacks using convolutional neural networks with domain knowledge. In *International Conference on Selected Areas in Cryptography*, pages 479–498. Springer, 2018. doi:10.1007/978-3-030-10970-7_22.

[HGM+11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptogr. Eng.*, 1(4):293–302, 2011. doi:10.1007/s13389-011-0023-x.

[HHO20] Anh-Tuan Hoang, Neil Hanley, and Maire O'Neill. Plaintext: A missing feature for enhancing the power of deep learning in side-channel analysis? breaking multiple layers of side-channel countermeasures. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 49–85, 2020. doi:10.46586/tches.v2020.i4.49-85.

[HK11] Jan Hauke and Tomasz Kossowski. Comparison of values of pearson's and spearman's correlation coefficients on the same sets of data. *Quaestiones geographicae*, 30(2):87, 2011. doi:10.21203/rs.3.rs-4380975/v1.

[KJJ99a] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag. URL: http://dl.acm.org/citation.cfm?id=646764.703989.

[KJJ99b] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999. URL: https://doi.org/10.1007/3-540-48405-1_25, doi:10.1007/3-540-48405-1_25.

[KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017. doi:10.5555/3294771.3294864.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A Machine Learning Approach Against a Masked AES. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2013. Berlin, Germany. doi:10.1007/978-3-319-14123-7_5.

[LZC+21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 235–274, 2021. doi:10.46586/tches.v2021.i3.235-274.

[Mag20] Houssem Maghrebi. Deep learning based side-channel attack: a new profiling methodology based on multi-label classification. *Cryptology ePrint Archive*, 2020.

[MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-enhanced power analysis collision attack. In *Cryptographic Hardware and Embedded Systems, CHES 2010: 12th International Workshop, Santa Barbara, USA, August 17-20, 2010. Proceedings 12*, pages 125–139. Springer, 2010. doi:10.1007/978-3-642-15031-9_9.

[MOP08] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008. doi:10.1007/978-0-387-38162-6.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016. doi:10.1007/978-3-319-49445-6_1.

[MS16] Amir Moradi and François-Xavier Standaert. Moments-correlating dpa. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, pages 5–15, 2016. doi:10.1145/2996366.2996369.

[MS23] Loïc Masure and Rémi Strullu. Side-channel analysis against anssi's protected aes implementation on arm: end-to-end attacks with multi-task learning. *Journal of Cryptographic Engineering*, pages 1–19, 2023. doi:10.1007/s13389-023-00311-7.

[PHJ+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. 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*, pages 4095–4102, 2017. doi:10.1109/ijcnn.2017.7966373.

[PPM+23] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Computing Surveys*, 55(11):1–35, 2023. doi:10.1145/3569577.

[PWP22] Guilherme Perin, Lichao Wu, and Stjepan Picek. 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. URL: https://tches.iacr.org/index.php/TCHES/article/view/9842, doi:10.46586/tches.v2022.i4.828-861.

[Rud17] Sebastian Ruder. An overview of multi-task learning in deep neural networks. *arXiv preprint arXiv:1706.05098*, 2017.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 30–46. Springer Berlin Heidelberg, 2005. URL: https://doi.org/10.1007/11545262_3, doi:10.1007/11545262_3.

[SM23] Marvin Staib and Amir Moradi. Deep learning side-channel collision attack. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2023(3):422–444, Jun. 2023. URL: https://tches.iacr.org/index.php/TCHES/article/view/10969, doi:10.46586/tches.v2023.i3.422-444.

[SWP03] Kai Schramm, Thomas Wollinger, and Christof Paar. A new class of collision attacks and its application to des. In *Fast Software Encryption: 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003. Revised Papers 10*, pages 206–222. Springer, 2003. doi:10.1007/978-3-540-39887-5_16.

[Tim19] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 107–131, 2019. doi:10.46586/tches.v2019.i2.107-131.

[vWWB11] Jasper GJ van Woudenberg, Marc F Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In *Topics in Cryptology–CT-RSA 2011: The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, pages 104–119. Springer, 2011. doi:10.1007/978-3-642-19074-2_8.

[WPP22a] Lichao Wu, Guilherme Perin, and Stjepan Picek. The best of two worlds: Deep learning-assisted template attack. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 413–437, 2022. doi:10.46586/tches.v2022.i3.413-437.

[WPP22b] Lichao Wu, Guilherme Perin, and Stjepan Picek. On the evaluation of deep learning-based side-channel analysis. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 49–71. Springer, 2022. doi:10.1007/978-3-030-99766-3_3.

[WPP23] Lichao Wu, Guilherme Perin, and Stjepan Picek. Not so difficult in the end: Breaking the lookup table-based affine masking scheme. In *International Conference on Selected Areas in Cryptography*, pages 82–96. Springer, 2023. doi:10.1007/978-3-031-53368-6_5.

[WPP24] Lichao Wu, Guilherme Perin, and Stjepan Picek. Weakly profiling side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2024. URL: https://ches.iacr.org/2024/papers-issue-4/4_73.pdf.

[WWK$^+$23] Lichao Wu, Léo Weissbart, Marina Krček, Huimin Li, Guilherme Perin, Lejla Batina, and Stjepan Picek. Label correlation in deep learning-based side-channel analysis. *IEEE Transactions on Information Forensics and Security*, 2023. doi:10.1109/tifs.2023.3287728.

[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019. URL: https://tches.iacr.org/index.php/TCHES/article/view/8391, doi:10.13154/tches.v2020.i1.1-36.

[ZY21] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Transactions on Knowledge and Data Engineering*, 34(12):5586–5609, 2021. doi:10.1109/TKDE.2021.3070203.