DL-SITM: Deep Learning-Based See-in-the-Middle Attack on AES

Tomáš Gerlich[1], Jakub Breier[2], Pavel Sikora[1], Zdeněk Martinásek[1], Aron Gohr[3], Anubhab Baksi[4], and Xiaolu Hou[5]

[1] *Brno University of Technology, Czechia, {Tomas.Gerlich,martinasek}@vut.cz, Pavel.Sikora@vutbr.cz*
[2] *TTControl GmbH, Vienna, Austria. jbreier@jbreier.com*
[3] *Independent Researcher*
[4] *Nanyang Technological University, Singapore, anubhab.baksi@ntu.edu.sg*
[5] *Slovak University of Technology, Slovakia, houxiaolu.email@gmail.com*

Abstract

The see-in-the-middle (SITM) attack combines differential cryptanalysis and the ability to observe differential patterns in the side-channel leakage traces to reveal the secret key of SPN-based ciphers. While SITM presents a fresh perspective to side-channel analysis and allows attacks on deeper cipher rounds, there are practical difficulties that come with this method. First, one must realize a visual inspection of millions of power traces. Second, there is a strong requirement to reduce noise to a minimum, achieved by averaging over 1000 traces in the original work, to see the patterns. Third, the presence of a jitter-based countermeasure greatly affects pattern identification, making the visual inspection infeasible.

In this paper we aim to tackle these difficulties by using a machine learning approach denoted as DL-SITM (deep learning SITM). The fundamental idea of our approach is that, while a collision obscured by noise is imperceptible in a manual inspection, a powerful deep learning model can identify it, even when a jitter-based countermeasure is in place. As we show with a practical experiment, the proposed DL-SITM approach can distinguish the two valid differentials from over 4M differential traces with only six false positives. Extrapolating from the parameters of this experiment, we get a rough estimate of 2^{43} key candidates for the post-processing step of our attack, which places it easily in the practical range. At the same time, we show that even with a jitter countermeasure shifting the execution by ± 15 samples, the testing f1-score stays at a relatively high (0.974).

1 Introduction

Over the last couple of decades, side-channel analysis (SCA) has secured a stable position in cryptographic research in academic and practical applications in industry. The seminal work by Kocher, Jaffe and Jun is regarded as its inception where it was shown that the power consumption during the sensitive computation leaks enough information so that the secret key under operation can be successfully recovered [KJJ99]. The analysis presented in the paper, which is termed as the *differential power analysis* (DPA), starkly contrasts the classical cryptanalysis, since for a classically secure cipher (like, AES), it can easily recover the entire key from a small number of cipher calls. Since then, the research in this direction has been in full swing, as the research community has kept exploring the field from various angles, finding more powerful and efficient methods based on SCA. Two main directions emerged – the *profiled* and *non-profiled* attacks. In the profiled attack scenario, the attacker first creates a leakage profile of a device identical to the target device. This leakage profile is then used to attack the target device [CRR03], generally reducing the number of attack traces required to recover the key. In the non-profiled attack scenario, the assumption is that the attacker does not possess an extra device to allow profiling and all the collected traces are used for the attack. Common non-profiled attack methods include *correlation power analysis* (CPA) [BCO04].

Over time, deep learning has also found its way into device security-related research in the past few years. Especially, after the introduction of the ASCAD dataset [BPS+20], many research works have competed to propose more accurate models to efficiently extract the side channel information from the traces (a good overview of the current state-of-the-art is given, for example, in [MDP20, PPM+23, BCH+22]. The main direction is towards improving profiled SCA [KPH+19, ZBHV20], thus competing with standard profiled attacks such as template attacks. Several works also focus on the use of deep learning methods in non-profiled SCA [Tim19, WHJ+21a, WPP23]. These methods generally use neural networks either in a similar way to how CPA uses simpler statistical distinguishers, or obtain labels for standard supervised learning from observable outputs of the cipher that relate in a simple way to internal secrets once the key is fixed.

Recently, a new side-channel analysis method was proposed, combining the way differential cryptanalysis utilizes the difference propagation in the cipher, and the ability to observe differential patterns in the leakage traces similar to *simple power analysis* (SPA) [KJJ99]. The method was titled *side-channel assisted differential plaintext attack* (SCADPA) [BJB18]. Initially, it was proposed for bit-permutation-based ciphers such as PRESENT [BKL+07], and it was shown it can be utilized for reverse engineering of secret S-boxes [BJHB20]. Later, it was extended to an attack called see-in-the-middle (SITM) [BBH+20] that targets inner rounds of SPN-based block ciphers. SITM, like the previous SCADPA methods, exploits the properties of the underlying diffusion function. Due to the fact that a certain differential needs to be observed after the propagation, SITM allows deeper round attacks, of up to 5 rounds for AES-128, and

for example, up to 12 rounds for SKINNY-64 or SKINNY-128. In a nutshell, while standard workflows in differential cryptanalysis induce a special differential state inside a cipher and then look for surviving statistical biases in the output to recognize that event a few rounds later, SITM uses a side channel for the same purpose but does not automate the evaluation of the resulting leakage.

The illustration of the SITM attack in [BBH+20] was based on the averaging of 1,000 power traces for each plaintext to reduce the noise and allow a visual inspection. In a real-world scenario, the attacker might not have access to traces with such a low noise. Moreover, distinguishing active columns by the naked eye depends on the analyst's skills and is prone to errors, especially since the attack requires inspection of 2^{22} differential traces on average to find the collision. If we assume the analyst can process one trace in 5 seconds, this would still result in 242 worker-days of analysis. In addition, human error rates in such a process are very hard to reliably quantify.

In this paper, we focus on a more practical realization of the SITM, where the attacker faces more noise in the measured traces and achieves complete automation. We also look at the scenario where the power traces are misaligned, thwarting the visual inspection of the active columns completely. Artificially induced jitter is a powerful countermeasure to attacks that rely on heavy averaging of traces because averaging of misaligned traces acts as a low-pass filter on the signal, which can destroy even easily interpretable, linear parts of the signal of interest if they are tightly localized in the data. On top of that, averaging of trace data naturally also destroys non-linear parts of the side-channel signal, irrespective of its low-pass filtering effects under jitter.

To make the attack feasible under these circumstances, we employ a deep-learning model to analyze the differential side channel traces, thus naming the approach DL-SITM. In the best-case scenario when we average over 4 traces, DL-SITM is capable of identifying the correct collisions for AES-128 from 2^{22} traces, with only 6 false positives.

The main contributions presented in this paper can be summarized as follows:

- We propose DL-SITM, a deep learning-based improvement on top of the See-in-the-Middle attack that offers a high level of automation and yields high accuracies for distinguishing correct collisions in side-channel traces.
- Instead of the visual inspection of SITM traces, requiring averaging of 1,000 power traces, DL-SITM achieves a 99.7% accuracy with just the average of 4 traces and is capable of handling misaligned traces with jitter of up to ± 15 samples without a significant accuracy degradation.
- In the realistic scenario where the attacker has to identify on average one collision among 2^{22} differential traces for AES-128, the DL-SITM false-positive rate stays extremely low – at just 6 wrongly identified collisions. This results in a remaining brute-force complexity of $\approx 2^{43}$. In principle, Considering the speed of AES-NI which can reach up to 927 MB/s on a relatively modest Intel i5 processor [AKH20], the master key can be recovered within a single day.
- We provide the datasets and DL models used in this paper in a public

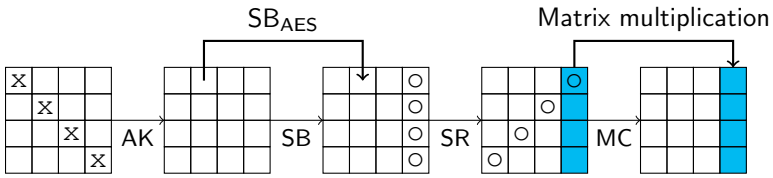


Figure 1: AES round function.

repository, to reproduce our results or improve upon them[1].

2 Background

In this section, we first provide an overview of AES-128 in Subsection 2.1 followed by the working principle of SITM on AES in Subsection 2.2. Subsection 2.3 explains the basic concepts of deep learning within the context of side-channel analysis. Finally, we give an overview of related work in Subsection 2.4.

2.1 AES-128

Advanced Encryption Standard (AES) is a standardized algorithm for symmetric block cipher that works with a block length of $n = 128$ bits. The number of rounds can be $10, 12, 14$ with the corresponding master key lengths of $128, 192, 256$ bits. In this paper, we look at AES-128, meaning the master key has a length of 128 bits, and the encryption/decryption runs for 10 rounds.

AES-128 encryption starts with the pre-whitening AddRoundKey. Then each of the following 9 rounds consists of SubBytes, ShiftRows, MixColumns, and AddRoundKey. The 10^{th} round consists of Subbytes, ShiftRows and AddRoundKey. The decryption algorithm involves inverses of SubBytes, ShiftRows, and MixColumns, denoted by InvSubBytes, InvShiftRows and InvMixColumns respectively.

As depicted in Figure 1, for illustration, we represent each cipher state as 4×4 squares, where each square corresponds to one byte. Following the convention, we refer to the bytes at positions $(0,0)$, $(1,1)$, $(2,2)$ and $(3,3)$ as the *main diagonal* of the cipher state (marked by "×" in Figure 1).

AddRoundKey is a bitwise `XOR` computation between the cipher state and the round key. SubBytes applies AES Sbox, denoted as $\mathrm{SB}_{\mathrm{AES}}$, to each byte of the cipher state, where $\mathrm{SB}_{\mathrm{AES}} : \mathbb{F}_2^8 \to \mathbb{F}_2^8$ is a permutation function on $8-$bit binary strings. InvSubBytes is the inverse of the permutation $\mathrm{SB}_{\mathrm{AES}}$.

ShiftRows rotates the bytes in each row of the cipher state. In particular, the first row is not changed, the second row is rotated to the left by one byte, the third row is rotated to the left by two bytes and the last row is rotated to

[1]The data sets are available at https://drive.google.com/drive/folders/1x1fUmSDCWDBqinRnUfoytc_QSLAmP9Zs?usp=sharing

the left by three bytes. InvShfitRows can be deduced from this and not included here for brevity.

The MixColumns operation multiplies each column of the cipher state by the matrix (in hexadecimal). The InvMixColumns operation multiplies each column of the cipher state by another matrix. The multiplication between two bytes is done by considering each byte as an element in the particular field $\mathbb{F}_2[x]/(f(x))$, where $f(x) = x^8 + x^4 + x^3 + x + 1$.

MixColumns matrix:

$$\begin{pmatrix} \texttt{02} & \texttt{03} & \texttt{01} & \texttt{01} \\ \texttt{01} & \texttt{02} & \texttt{03} & \texttt{01} \\ \texttt{01} & \texttt{01} & \texttt{02} & \texttt{03} \\ \texttt{03} & \texttt{01} & \texttt{01} & \texttt{02} \end{pmatrix}$$

InvMixColumns matrix:

$$\begin{pmatrix} \texttt{0E} & \texttt{0B} & \texttt{0D} & \texttt{09} \\ \texttt{09} & \texttt{0E} & \texttt{0B} & \texttt{0D} \\ \texttt{0D} & \texttt{09} & \texttt{0E} & \texttt{0B} \\ \texttt{0B} & \texttt{0D} & \texttt{09} & \texttt{0E} \end{pmatrix}$$

We refer the readers to the original document [DR99] for more details on the algorithm design and AES key schedule. It is important to understand that the very first AddRoundKey operation of AES is computed with the master key, all round keys are derived from the master key. Each iteration of the SITM attack reduces the key space of four bytes of the master key, therefore an attacker has to execute the SITM attack in four steps in order to reveal all 16 bytes of the master key.

2.2 SITM on AES-128

Before we go into details of SITM attack methodology, we state the attacker model that was followed from [BBH+20]:

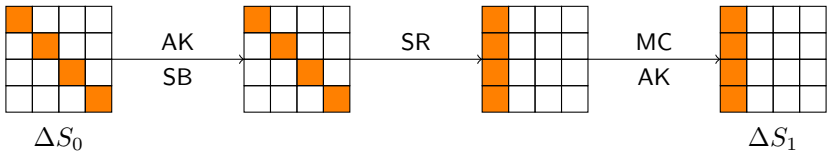
- Implementation is done in software.
- SubBytes operation is implemented column by column.
- The attacker can query encryptions with their chosen plaintext and a fixed unknown master key.
- The attacker can observe side-channel leakages to conclude if the intermediate values have changed between two encryptions with different plaintexts. This can be done by post-processing the traces.

Model of the Attacker

In this paper, we keep the first three assumptions of the attack, but we explore how changes in intermediate values can be deduced from side-channel measurements with the help of neural networks. We consider the encryption of two plaintext blocks, say S_0 and S'_0. The corresponding cipher states at the end of round i are denoted by S_i and S'_i respectively ($i = 1, 2, \ldots, 10$). The `XOR` difference between S_i and S'_i is represented by ΔS_i ($i = 0, 1, 2 \ldots, 10$). A byte (or a column) in ΔS_i is called *active* if it is nonzero. A sequence of ΔS_i's is called a *differential pattern*. SITM exploits certain differential patterns.

In this paper, we target differential patterns $\{\Delta S_0, \Delta S_1, \Delta S_2\}$ such that

1. The active bytes in ΔS_0 are along the main diagonal, and
2. There is only one active byte in ΔS_1.



AK, SB, SR, MC stand for AddRoundKey, SubBytes, ShiftRows, MixColumns respectively. Orange boxes correspond to (potentially) active bytes.

Figure 2: An illustration of how the active bytes in ΔS_0 can propagate to ΔS_1.

When such a differential pattern is found, the SITM attack can reduce the key space for the four bytes in the diagonal of the master key to around 2^8 (as opposed to 2^{32}). We note that similar attacks can be carried out for other diagonals of the key.

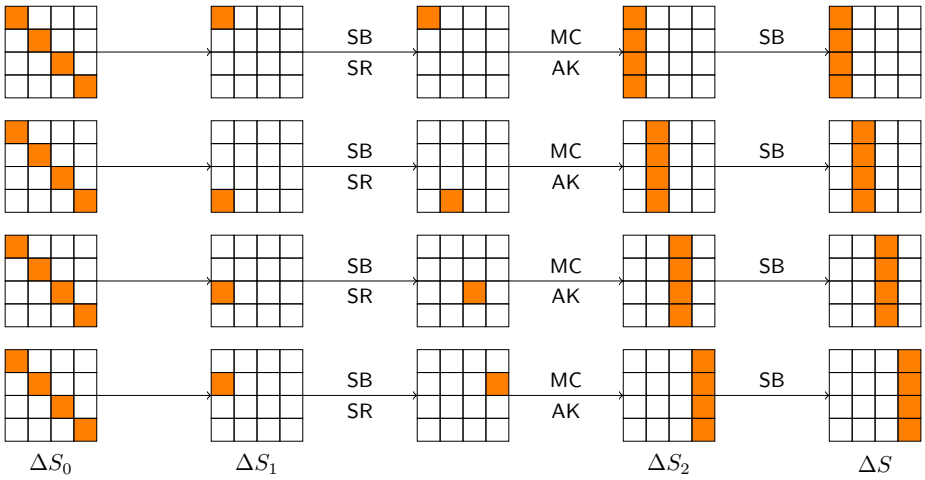
Figure 2 illustrates how the active bytes in ΔS_0 can propagate to ΔS_1. AddRoundKey and SubBytes do not change the total number and positions of the active bytes. ShiftRows will shift the active bytes from the main diagonal to the first column. Finally, MixColumns keeps the active bytes to stay in the first column, but can potentially reduce the number of active bytes. We can see that there are only four possible differential patterns that satisfy conditions 1 and 2 above – the active byte in ΔS_1 can be either at position $(0,0)$, $(1,0)$, $(2,0)$ or $(3,0)$.

Figure 3 illustrates all four possible differential patterns, where ΔS denotes the cipher state `XOR` difference in round 3 before ShiftRows. We can see that:

$$\begin{cases} \text{byte } (0,0) \text{ active in } \Delta S_1 \longleftrightarrow \text{first column active in } \Delta S \\ \text{byte } (3,0) \text{ active in } \Delta S_1 \longleftrightarrow \text{second column active in } \Delta S \\ \text{byte } (2,0) \text{ active in } \Delta S_1 \longleftrightarrow \text{third column active in } \Delta S \\ \text{byte } (1,0) \text{ active in } \Delta S_1 \longleftrightarrow \text{fourth column active in } \Delta S \end{cases} \tag{1}$$

The original paper [BBH+20] showed that by observing side-channel leakages, we can conclude which column is active in ΔS. Those leakages are measured during the third round of AES encryption for S_0 and S_0'. Typically a few traces are collected for each encryption to cancel out the noise. The difference between those two averaged traces is computed for the time interval corresponding to the 3rd round SubBytes. Assuming the AES SubBytes operation is implemented column by column, we expect the peaks in the trace difference corresponding to those four columns to appear in sequential order. In particular, we expect the peak that corresponds to the first column being active to appear earlier than that corresponding to the second column being active.

After the active column is identified in ΔS, by Equation (1), we will know which byte in ΔS_1 is active. Let Δ denote the differential value of the active byte in S_1 and let k_{ij} denote the key byte at position (i,j) of the master key K. We will detail how to reduce key space for k_{00}, k_{11}, k_{22}, and k_{33}, when only the first column is active in S, i.e., when only the byte in position $(0,0)$ is active in ΔS_1. Similar attack methods hold for other active columns.



Orange boxes correspond to active bytes.

Figure 3: An illustration of all four possible differential patterns such that the active bytes in ΔS_0 are along the main diagonal, and there is only one active byte in ΔS_1.

Since AddRoundKey and ShiftRows do not have any influence on the differential values, by Equation (2.1), the corresponding differential values for the four bytes after SubBytes in the first round (the second state in Figure 2) would be:

$$\texttt{0E} \cdot \Delta, \quad \texttt{09} \cdot \Delta, \quad \texttt{0D} \cdot \Delta, \quad \texttt{0B} \cdot \Delta.$$

Let $\delta_1,\ \delta_2,\ \delta_3,\ \delta_4$ denote the four differential values in ΔS_0. Similarly, let p_{ij} denote the byte at (i, j) position in S_0. We have:

$$\begin{cases} \mathrm{SB}_{\mathrm{AES}}(p_{00} \oplus k_{00} \oplus \delta_1) \oplus \mathrm{SB}_{\mathrm{AES}}(p_{00} \oplus k_{00}) = \texttt{0E} \cdot \Delta & (2) \\ \mathrm{SB}_{\mathrm{AES}}(p_{11} \oplus k_{11} \oplus \delta_2) \oplus \mathrm{SB}_{\mathrm{AES}}(p_{11} \oplus k_{11}) = \texttt{09} \cdot \Delta & (3) \\ \mathrm{SB}_{\mathrm{AES}}(p_{22} \oplus k_{22} \oplus \delta_3) \oplus \mathrm{SB}_{\mathrm{AES}}(p_{22} \oplus k_{22}) = \texttt{0D} \cdot \Delta & (4) \\ \mathrm{SB}_{\mathrm{AES}}(p_{33} \oplus k_{33} \oplus \delta_4) \oplus \mathrm{SB}_{\mathrm{AES}}(p_{33} \oplus k_{33}) = \texttt{0B} \cdot \Delta. & (5) \end{cases}$$

Before we continue with the attack details, we introduce the notion related to the difference distribution. from [Bak22, Chapter 7].

Definition 1 *For an Sbox SB: $\mathbb{F}_2^{\omega_1} \to \mathbb{F}_2^{\omega_2}$, the expanded difference distribution table (DDT) of SB is a $2-$dimensional table T of size $(2^{\omega_1} - 1) \times 2^{\omega_2}$ such that for any $0 < \delta < 2^{\omega_1}$ and $0 \leq \Delta < 2^{\omega_2}$, the entry of T at the Δth row and δth column is given by*

$$T[\Delta, \delta] = \{\mathbf{a} | \mathbf{a} \in \mathbb{F}_2^{\omega_1}, SB(\mathbf{a} \oplus \delta) \oplus SB(\mathbf{a}) = \Delta\}.$$

We refer to δ as the input difference, and Δ as the output difference.

Thus, Equations (2), (3), (4) and (5) imply that:

$$p_{00} \oplus k_{00}, \quad p_{11} \oplus k_{11}, \quad p_{22} \oplus k_{22}, \quad p_{33} \oplus k_{33}$$

are AES Sbox inputs that give output differences

$$\texttt{0E} \cdot \Delta, \quad \texttt{09} \cdot \Delta, \quad \texttt{0D} \cdot \Delta, \quad \texttt{0B} \cdot \Delta$$

with input differences $\delta_1, \delta_2, \delta_3, \delta_4$ respectively. Then, by using the difference distribution table for AES Sbox and with the knowledge of the plaintexts, we can reduce the key hypotheses for $k_{00}, k_{11}, k_{22}, k_{33}$.

For example, let us take the following pair of plaintexts that results in the first differential pattern in Figure 3:

$$\texttt{4C3C3F54C7AAD34E607110C753C5E990}, \quad \texttt{033C3F54C725D34E607131C753C5E90F},$$

with the master key given by

$$\texttt{34463146344638383341464542413731}.$$

The values of the main diagonal of the two plaintexts are $\texttt{4C}$, $\texttt{AA}$, $\texttt{10}$, $\texttt{90}$ and $\texttt{03}$, $\texttt{25}$, $\texttt{31}$, $\texttt{0F}$. In particular, $p_{00} = \texttt{4C}$, $p_{11} = \texttt{AA}$, $p_{22} = \texttt{10}$, $p_{33} = \texttt{90}$. The differences between those two diagonals are then

$$\begin{aligned} \delta_1 &= \texttt{4C} \oplus \texttt{03} = \texttt{4F} \\ \delta_2 &= \texttt{AA} \oplus \texttt{25} = \texttt{8F} \\ \delta_3 &= \texttt{10} \oplus \texttt{31} = \texttt{21} \\ \delta_4 &= \texttt{90} \oplus \texttt{0F} = \texttt{9F}. \end{aligned}$$

Thus, $\texttt{4C} \oplus k_{00}$, $\texttt{AA} \oplus k_{11}$, $\texttt{10} \oplus k_{22}$, $\texttt{90} \oplus k_{33}$ are the AES Sbox inputs that give output differences

$$\texttt{0E} \cdot \Delta, \quad \texttt{09} \cdot \Delta, \quad \texttt{0D} \cdot \Delta, \quad \texttt{0B} \cdot \Delta$$

with input differences $\texttt{4F}$, $\texttt{8F}$, $\texttt{21}$, $\texttt{9F}$ respectively. To find the possible values of k_{00}, k_{11} k_{22}, and k_{33}, we first find values of Δ that gives nonempty entries in the DDT for columns $\texttt{4F}$, $\texttt{8F}$, $\texttt{21}$, $\texttt{9F}$ and corresponding rows $\texttt{0E}\cdot\Delta, \texttt{09}\cdot\Delta, \texttt{0D}\cdot\Delta, \texttt{0B}\cdot\Delta$. There are in total 13 of them, as shown in Table 1.

Then with the knowledge of the plaintext, we can find all the possible values for the four key bytes, as shown in Table 2. The remaining number of key hypotheses is given by $2^4 \times 12 + 2^3 \times 4 = 224 \approx 2^8$.

Now if we compute using our plaintext and master key, the cipher states at the end of round 1 are:

$$S_1 = \texttt{1F1DABAE4071BDD502563FBF63841BAE},$$

$$S_1' = \texttt{C81DABAE4071BDD502563FBF63841BAE}.$$

We can see that the active byte in S_1 is given by $\Delta = \texttt{D7}$, which is among the values we have found in Tables 1 and 2 (in blue). The correct key byte values are given in the corresponding row in Table 2 (in blue).

Table 1: Possible values of Δ and the corresponding possible values for $k_{00} \oplus \texttt{4C}, k_{11} \oplus \texttt{AA}, k_{22} \oplus \texttt{10}, k_{33} \oplus \texttt{90}$.

Δ	$k_{00} \oplus$ 4C	$k_{11} \oplus$ AA	$k_{22} \oplus$ 10	$k_{33} \oplus$ 90
1A	16, 59	65, EA	CF, EE	62, FD
29	96, D9	3E, B1	85, A4	78, E7
42	28, 67	58, D7	59, 78	16, 89
5D	AB, E4	40, CF	81, A0	45, DA
66	03, 4C	2C, A3	D8, F9	1D, 82
71	AF, E0	78, F7	DE, FF	39, A6
74	82, CD	5D, D2	07, 26	4E, D1
95	07, 48	43, CC	87, A6	65, FA
9C	97, D8	00, 3D, 8F, B2	44, 65	7F, E0
CC	1D, 52	37, B8	93, B2	5F, C0
D7	37, 78	63, EC	56, 77	3E, A1
E7	3A, 75	7B, F4	1B, 3A	63, FC
EB	BB, F4	34, BB	CD, EC	54, CB

2.3 Deep Learning and Side-Channel Analysis

Side-channel analysis (SCA) has emerged as a serious threat to cryptographic implementation security, exploiting physical emissions from cryptographic devices to extract secret information [HB24]. These emissions, such as power consumption, electromagnetic radiation, and timing information, help attackers recover cryptographic keys in a fraction of the time compared to cryptanalysis attacks. Traditional SCA methods, such as differential power analysis (DPA) [KJJ99] and correlation power analysis (CPA) [BCO04], rely heavily on statistical techniques and require significant domain expertise in data preparation for the analysis, especially in the presence of countermeasures.

With the rise of deep learning (DL), SCA has undergone a significant transformation. Deep neural networks have shown remarkable capabilities in learning complex patterns and features from raw side-channel data [BPS+20]. Convolutional neural networks (CNNs) are especially useful for this purpose as they can directly process misaligned traces [PSK+18]. It was also shown that traditional SCA pre-processing techniques can be directly plugged in as a part of the model to reduce the need for having multiple steps [WHJ+21b]. This shift has not only enhanced the efficiency and success rates of side-channel attacks but also reduced the dependency on the analyst's knowledge and expertise, making SCA more accessible to a wider population. In the rest of the paper, we refer to deep learning-based SCA as DL-SCA.

There are two general approaches to DL-SCA, *profiled* and *non-profiled*. We will outline both of these in more detail below:

- **Profiled DL-SCA:** Profiled attacks are similar to the supervised learning [HTF+09] in the context of machine learning. The approach involves two phases:

Table 2: Possible values of Δ and the corresponding key hypotheses for $k_{00} \oplus \texttt{4C}$, $k_{11} \oplus \texttt{AA}$, $k_{22} \oplus \texttt{10}$, $k_{33} \oplus \texttt{90}$.

Δ	k_{00}	k_{11}	k_{22}	k_{33}
1A	5A, 15	CF, 40	DF, FE	F2, 6D
29	DA, 95	94, 1B	95, B4	E8, 77
42	64, 2B	F2, 7D	49, 68	86, 19
5D	E7, A8	EA, 65	91, B0	D5, 4A
66	4F, 00	86, 9	C8, E9	8D, 12
71	E3, AC	D2, 5D	CE, EF	A9, 36
74	CE, 81	F7, 78	17, 36	DE, 41
95	4B, 04	E9, 66	97, B6	F5, 6A
9C	DB, 94	AA, 97, 25, 18	54, 75	EF, 70
CC	51, 1E	9D, 12	83, A2	CF, 50
D7	7B, 34	C9, 46	46, 67	AE, 31
E7	76, 39	D1, 5E	0B, 2A	F3, 6C
EB	F7, B8	9E, 11	DD, FC	C4, 5B

The correct key bytes are marked in blue.

1. *Profiling phase:* The attacker is assumed to have access to a device similar to the device under test (DUT) and uses it to collect a large number of traces, along with the corresponding plaintexts and secret keys. These together are used as a training dataset for the model to learn the relationship between the side-channel emissions and the secret key.
2. *Attack phase:* The model trained in the previous phase is now used to analyze the traces from the actual DUT to recover the secret key.

The majority of literature focuses on profiled DL-SCA as these attacks are highly effective and powerful. The model can be fine-tuned to learn patterns and features specific to the device and the implementation.

- **Non-profiled DL-SCA:** Non-profiled attacks are analogous to unsupervised learning [CA16]. The assumption is that the attacker does not have access to a similar device for assembling a labeled dataset for training. Instead, the traces from the DUT are directly analyzed without a profiling phase. These attacks are more challenging because they must infer the key-related information directly from the observed traces. One of the non-profiled techniques is differential deep learning analysis (DDLA), analogous to DPA [Tim19].

There are other directions that utilize deep learning for some tasks within SCA, such as leakage assessment [MWM21] or stream cipher analysis [KDB+22]. For a more comprehensive overview of current advances in DL-SCA, we refer interested readers to [PPM+23].

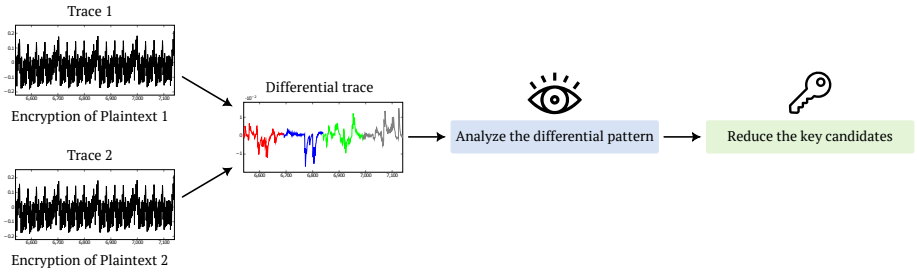


Figure 4: Illustration of one iteration of SCADPA methodology developed in [BJB18]. The attacker can repeat this process until the master key is found or until it can be searched with a reasonable time complexity.

2.4 Related Works

The original SCADPA was proposed in 2018 [BJB18] with a case study on PRESENT-80 block cipher. In this chosen-plaintext SCA attack, the attacker first encrypts two plaintexts that differ in exactly one nibble and observes the side-channel leakage during the first round S-box computation. By doing that, the attacker obtains two leakage traces that are subtracted from each other to get a difference trace. Based on the peaks in the difference trace, the attacker can reduce the search complexity of the key nibble. This process is illustrated in Figure 4. In an optimal case, the attacker can recover the entire 80-bit secret key with 17 encryptions and a search complexity of 2^{16}. The attack was demonstrated on an 8-bit AVR microcontroller.

In [BJHB20], SCADPA was extended to GIFT, and multiple implementations were considered, including bitslice ones that run on 32-bit architectures. Apart from that, a method to reverse-engineer secret S-boxes based on SCADPA was proposed. The experiments were done on a 32-bit ARM microcontroller. Three related attacks have been developed which loosely follow the SCADPA methodology. A *semi-blind combined middle-round attack* (SBCMA) [HBB22] utilizes fault injection and subsequent side-channel measurement in the middle rounds. A *differential no-fault analysis* (DNFA) [HBB21] exploits statistical properties of the ciphertext distribution together with side-channel information. Both SBCMA and DNFA target GIFT-based AEAD schemes. An attack titled SCADFA [PDJ+19], side-channel assisted differential fault analysis explored another application of SCADPA – observation of difference propagation in the middle rounds after the fault was introduced in the cipher.

The SITM attack was a generalization of SCADPA to the SPN-based ciphers [BBH+20]. SITM pushed the side-channel analysis part into the middle rounds (up to 12 for some ciphers) thanks to the more sophisticated utilization of differential characteristics of ciphers. Provided experiments showed the feasibility of the attack on 8-bit and 32-bit microcontrollers. Furthermore, a way to defeat shuffling countermeasures was shown in the paper. SITM attack was later extended to ARIA and DEFAULT [PK22]. An improvement to the

attack to extend masking to 4 rounds in the case of AES-128 and 12 rounds for AES-256 was proposed in [PKK21]. Reverse-engineering part of SCADPA was further improved in [CBB21].

The main drawback of the attack is that it requires a visual inspection of power traces to recognize whether a plausible difference propagation happened in the cipher. To be able to recognize collisions, it is often necessary to average a large number of traces to minimize the noise – for example, SITM [BBH+20] used an average of 1000 traces. Measurement of power traces and their visual inspection makes the SITM attack very hard to mount in practice. For example, in the case of AES-128, the attacker needs to analyze 2^{22} differential traces on average to find a collision. Another issue with the attack is that a simple jitter will effectively stop the attacker from being able to visually analyze the differences.

3 DL-SITM Methodology

In the following, we describe the working principle of the DL-SITM attack. The initial sequence of the attack steps is the same as in the original SITM (see Section 2.2). First, the differential trace is calculated as a difference of power traces obtained by measuring two plaintexts (S_0, S'_0 that differ in the main diagonal). In the second step, the neural network model determines the collision from the differential trace – the four possible patterns that help in the key recovery are stated in Figure 3. Based on the knowledge of plaintexts and collisions, the possible values of Δ and the corresponding key hypotheses are calculated in the last step. While we implemented the DL-SITM for the main diagonal, it is directly applicable to the remaining three diagonals.

3.1 Generation of Plaintext Pairs

To generate plaintext pairs for side-channel measurement, we take a closer look at the differential patterns in Figure 3. We can see that instead of generating random plaintexts S_0, S'_0 that differ in the main diagonal and check if they satisfy any of the patterns, we can generate random states S_1 and S'_1 that differ in only one byte in the first column. After that, we simply decrypt the two cipher states S_1 and S'_1 using the inverse of the first round of AES to find pairs of plaintexts. If the resulting pair only has differences along the diagonal, we record them as one pair for our experiment. Furthermore, to generate other plaintext pairs that do not result in any of our target differential patterns, we simply generate random states S_1 and S'_1 that differ in more than one byte in the first column.

Since the goal of DL-SITM is to identify if any of the four differential patterns shown in Figure 3 have occurred, we separate our plaintext pairs into five classes:

- *Class* 0 consists of plaintext pairs that result in the first differential pattern in Figure 3. In particular, only the first column is active in round 3 before ShiftRows;

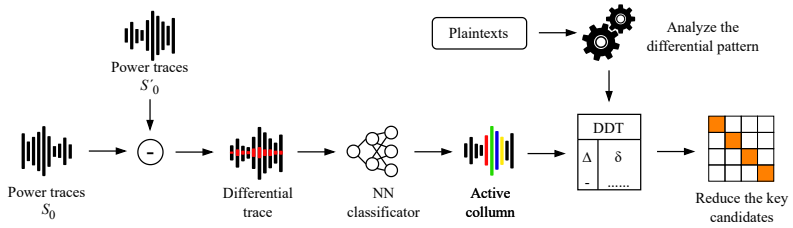


Figure 5: DL-SITM flowchart.

- *Class* 1 consists of plaintext pairs that result in the second differential pattern in Figure 3. In particular, only the second column is active in round 3 before ShiftRows;
- *Class* 2 consists of plaintext pairs that result in the third differential pattern in Figure 3. In particular, only the third column is active in round 3 before ShiftRows;
- *Class* 3 consists of plaintext pairs that result in the fourth differential pattern in Figure 3. In particular, only the fourth column is active in round 3 before ShiftRows;
- *Class* 4 consists of all other plaintext pairs.

Each dataset in our experiments was divided into training, validation, and testing in the 60 : 20 : 20 ratio, respectively. To identify points of interest, we utilized a differential power analysis attack for SubBytes of the first three rounds. In other words, we chose the output of the above-mentioned operations as the intermediate value of the DPA attack based on the correlation coefficient (in the following text denoted as CPA) [MOP07]. The result of the CPA analysis in Figure 6 shows that the third round SubBytes operation can be analyzed by restricting the interval to samples $215,000 - 245,000$. The sample points in this interval were then used for the subsequent steps of the experiment to reduce the complexity of the data. The direct input of the CNN was the difference between two such signals (see Figure 5).

3.2 Model Architecture

Our model architecture was based on the current state-of-the-art in DL-SCA. The conclusion from the current literature is that CNNs are very effective and accurate when applied to power trace classification, e.g., in [BPS+20, SM23]. The initial CNN model consisted of convolutional blocks, which were composed of one convolutional layer and one max pooling layer. Then a dropout layer was added, and immediately after the dense layers were placed. The network output was formed by the last dense layer which classifies the five classes. Keras Tuner was used to find the most suitable architecture for our experiments by utilizing Bayesian optimization with the Gaussian process. For each dataset, we

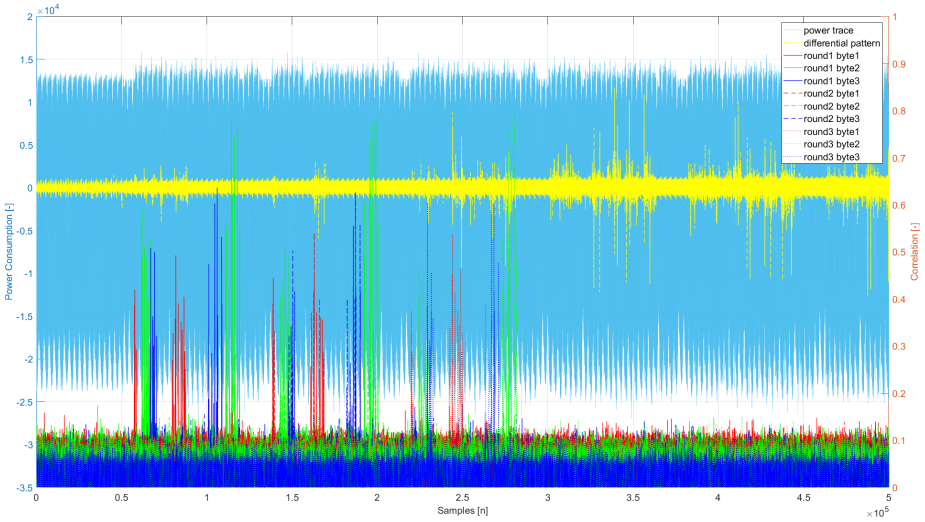


Figure 6: Verification of points of interest.

trained $1,000$ models. The most suitable one was selected based on the f1-score metric for the validation and testing part of the dataset. *Argmax* was utilized for datasets testing throughout the entire process. Argmax selects the predicted class based on the highest probability.

3.3 Model Metrics

The following metrics were used to evaluate the effectiveness of our models:

(i) True positive (TP): NN predicted the *active column* $= X$, and the active column was X.

(ii) False positive (FP): NN predicted the *active column* $= X$, and the active column was Y.

(iii) True negative (TN): template attack predicted the *active column* $= Y$, and the active column was Y.

(iv) False negative (FN): NN predicted the *active column* $= Y$, and the active column was X

where $X \neq Y \wedge X, Y \in [\![0, 4]\!]$ represent *active column* value.

The precision rate is the ratio of correct classifications and all classifications made for individual differential traces (see each row):

$$\text{TA precision} = \frac{TP}{TP + FP}. \tag{6}$$

The recall rate of TA shows the ratio of correct classifications and all classifications made for the correct *active column* (column of matrix):

$$\text{TA recall} = \frac{TP}{TP + FN}. \tag{7}$$

Naturally, the accuracy rate indicates the ratio of all correct classifications and the number of all classifications made:

$$\text{Accuracy rate} = \frac{TP + TN}{TP + FP + TN + FN}. \tag{8}$$

The F1-score is the harmonic mean of precision and recall. It provides a balance between these two metrics and is especially useful when the datasets are imbalanced. We use this metric for comparing the performance of NN classification models.

$$\text{F1-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{9}$$

4 Experimental Evaluation

4.1 Experimental Setup

Our experimental testbed used for power analysis measurements was based on Sasebo Sakura G and Sakura W boards and a software AES implementation written in C. The device under test was a programmable smart card with an ATMega8515 microcontroller. For measurements, we used Keysight MSOS104A digital oscilloscope. The sampling rate was set to 200 MS/s.

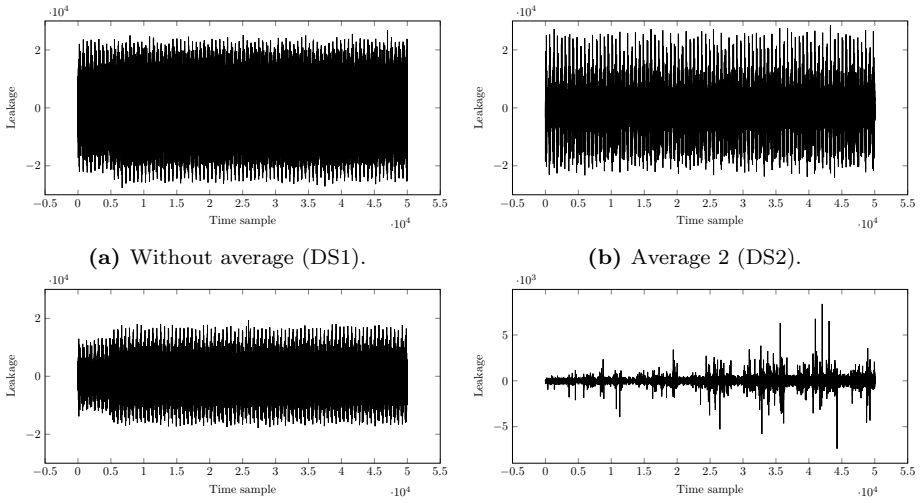
The workstation used for machine learning was equipped with AMD Ryzen 5 5600X 3,7 GHz CPU, Kingston 3200 MHz DDR4 32 GB RAM, and Nvidia GeForce RTX 3080 10 GB GPU. The NNs were programmed using the Keras library, which uses the TensorFlow backend. The following software packages were installed to work with the AI models in our attack: OS Ubuntu, CUDA 11.6, cuDNN 8.2.4, Miniconda 4.12.0, Python 3.10.4 (Tensorflow 2.9.1, Keras 2.9.0).

4.2 Test Scenario: Noise Impact – Fixed Key

To verify the usability and effectiveness of the DL-SITM proposed, we prepared a total of 4 datasets that differed in the amount of measurement noise. We used averaging of power traces to reduce the noise, the number of power traces utilized to calculate the average was set directly on the oscilloscope. In the first data set the power traces were stored without averaging, in the second, third, and fourth datasets, the number of averaging was set to 2, 4, and 1024, respectively[2]. In the following parts, we label these datasets DS1, DS2, DS3, and DS4. Each dataset consists of 10000 power traces that cover the first five rounds of the AES-128 encryption process. We measured 2000 power traces for each active column. Examples of the differential traces for individual datasets are depicted in Figure 7.

One can easily observe the relationship between the number of averaging and the visibility of the differential patterns. The patterns are visible only in Figure 7d, allowing the SITM attack even without the help of the NN model,

[2]The averaging can be set on the MSOS104A oscilloscope to 0, 2, 4, 16, 32 and 1024.



(a) Without average (DS1).

(b) Average 2 (DS2).

(c) Average 4 (DS3).

(d) Average 1024 (DS4).

Figure 7: Comparison of differential traces.

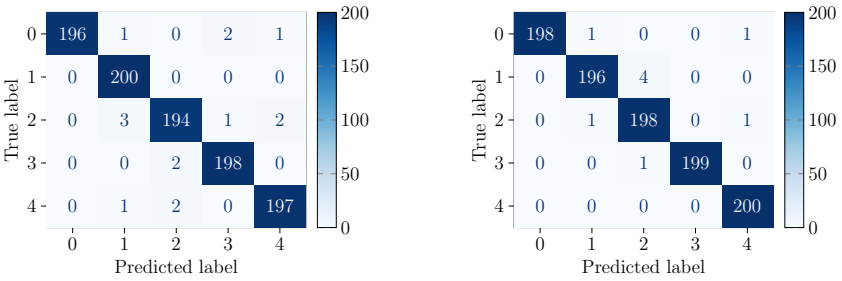
as described in [BBH+20]. However, this is not possible for lower averaging (Figure 7a, 7b, 7c) as one cannot observe the patterns by visual inspection.

We provide the *confusion matrices*[3] to evaluate all the classifications carried out by matching templates[4]. The confusion matrices in Figure 8 cross predictions with true values obtained from the whole test set classification. Each column of the table corresponds to the predicted active column, and each row corresponds to the correct active column. For example, in Figure 8c, all zero active bytes were classified correctly, as was the second active column. The first active column had one incorrect prediction, where one differential trace was classified as a zeroth active column. The third active column was also misclassified into the zero class. Additionally, one instance of the fourth active column was misclassified as belonging to the first active column.

Table 3 provides a detailed analysis of the best configurations for convolutional neural networks across our four datasets. The number of convolutional units and groups significantly influenced model performance. While the best values varied, a higher number of convolutional units (between 12 and 14) generally yielded better results. For example, DS1 and DS3 achieved optimal performance with 14 convolutional units, whereas DS2 and DS4 performed best with 12 units. This underscores the importance of fine-tuning these parameters to balance model complexity and performance.

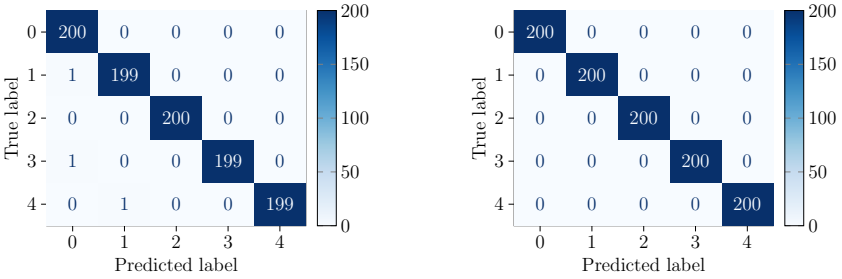
[3]The interested reader can consult [SL09] to obtain additional explanations about performance measurements for classification, e.g. *confusion matrix*, *precision*, *recall*.

[4]It is usually more suitable to use *guessing entropy* as a metric to compare different key recovery side-channel attack implementations [SMY09], but we focused on *classification of differential traces*, not the secret key guessing, therefore we used a confusion matrix, which is also often used during profiling PA attacks [FLD12].



(a) Without average (DS1).

(b) Average 2 (DS2).



(c) Average 4 (DS3).

(d) Average 1,024 (DS4).

Figure 8: Confusion matrices for test scenario: noise impact – fixed key.

Dropout rates and regularization strategies also had a substantial impact on the models. For DS4, a higher dropout rate of 0.6 was the most favorable, suggesting a need for stronger regularization to prevent overfitting. In contrast, DS1 and DS3 required lower dropout rates of 0.2. Convolutional regularization consistently improved performance across all datasets, while dense regularization was particularly beneficial for DS3.

The sizes of the convolutional kernels and pooling layers were also crucial. Smaller kernel sizes $(8 - 12)$ were preferred for DS1, DS2, and DS3, while DS4 required a larger kernel size of 15. Similarly, the optimal pool size ranged from 5 to 6 across the datasets. These findings highlight the importance of adjusting these parameters based on the specific characteristics of each dataset.

The models achieved remarkably high f1-scores across all datasets, with all f1-scores consistently near 1.0, with DS4 and DS3 achieving perfect validation scores and DS2 achieving a test f1-score of 0.99. These metrics reflect the robustness and accuracy of the optimized models. DS4 achieved a perfect score of 100%, likely because with such extensive averaging, classification becomes possible even by the human eye. In contrast, for the other datasets, the averages are not as easily distinguishable by the naked eye, which contributes to a reduction in accuracy even for neural networks.

Table 3: CNN architectures for different averages for fixed key scenario.

Hyperparameters	Tested Values	DS1 0 average	DS2 2 average	DS3 4 average	DS4 1,024 average
Convolutional groups	[1, 2, 3, 4, 5]	4	5	3	4
Convolutional units	[4, 8, 10, 12, 14, 16]	14	12	14	12
Dense layers	[1, 2, 3, 4]	1	1	3	1
Dense units	[128, 256, 512, 1,024]	512	512	128	512
Convolutional pool size	[1, 2, 3, 4, 5, 6]	6	6	6	5
Convolutional strides	[1, 2, 3, 4]	3	2	3	2
Convolutional kernel size	[2, 8, 10, 11, 12, 15]	12	8	12	15
Convolutional regularization	[Yes, No]	Yes	Yes	Yes	Yes
Dense regularization	[Yes, No]	No	No	Yes	No
Dropout	[0.2, 0.4, 0.6, 0.8]	0.2	0.4	0.2	0.6
Optimizer	[Adam, RMSprop]	RMSprop	RMSprop	RMSprop	Adam
Activation function	[ReLU, SELU]	ReLU	ReLU	ReLU	SELU
Accuracy	**Train f1-score**	0.999	0.999	0.999	1.0
	Valid f1-score	0.99	0.993	0.999	1.0
	Test f1-score	0.985	0.99	0.997	1.0

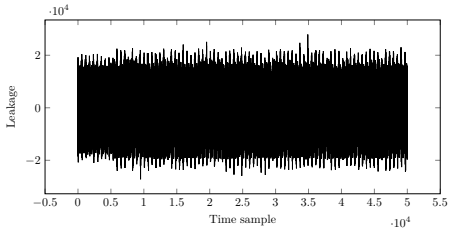
4.3 Test Scenario: Jitter Impact – Constant Key

In this part, we investigated the impact of jitter on the classification accuracy of CNN models. The datasets used in this study were derived from dataset DS4 described in Section 4.2. Four datasets were generated, where their labels correspond to the maximum shift of samples between the two input traces, which were then used to create the differential trace. These datasets are defined by a random mutual shift in power traces by ± 5, 10, 15, and 20 samples. Figure 9 illustrates the impact of varying jitter values on differential traces. Each subplot shows a differential trace subjected to a distinct jitter level. An increment in the jitter value increases the distortion observed in the differential traces. This amplification of distortion directly impacts the classification accuracy of the CNN employed for analysis. The introduction of jitter reduces the signal-to-noise ratio, thereby complicating the task of the CNN to accurately classify the data. One can see that, compared to Figure 7(d), showing the original differential trace for dataset DS4, the differential patterns are not possible to be determined by the naked eye even for the lowest jitter value.

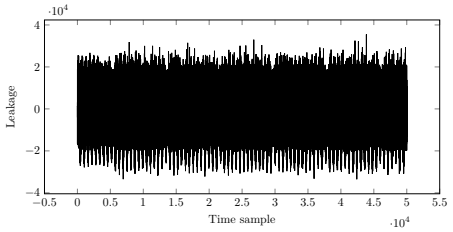
Table 4 presents the values obtained for the CNN models that achieved the highest classification accuracy. The number of convolutional units and groups significantly influenced model performance under different jitter levels.

Notably, 16 convolutional units were optimal for jitter 5, while $12 - 14$ units performed better for jitters 10, 15 and 20. Consistently, 5 convolutional groups yielded the best results across all jitter levels, suggesting a robust structure for handling various jitter intensities. The configuration of dense layers and units varied with jitter levels. A single dense layer was sufficient for jitters 5 and 10, whereas 4 dense layers were necessary for jitters 15 and 20. Dense units also fluctuated, with higher values $(1,024)$ optimal for jitter 5 and lower values $(128 - 256)$ for higher jitter levels.

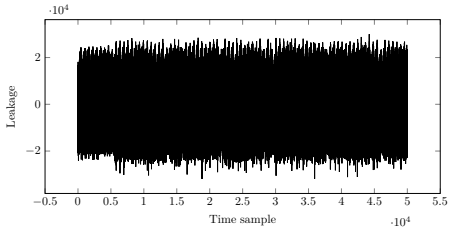
Dropout rates and regularization played a crucial role in maintaining model



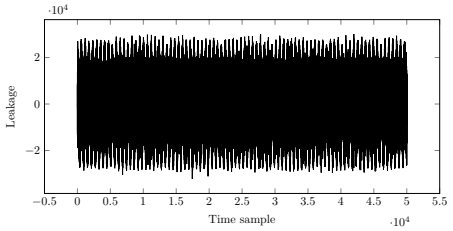
(a) Differential trace with jitter of ±5 samples.



(b) Differential trace with jitter of ±10 samples.



(c) Differential trace with jitter of ±15 samples.



(d) Differential trace with jitter of ±20 samples.

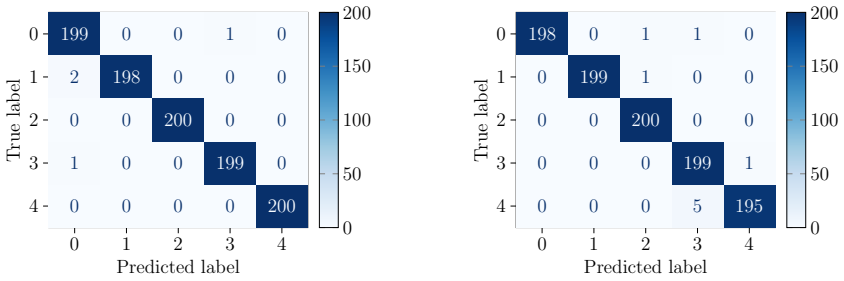
Figure 9: Comparison of differential traces with artificial random jitter obtained by shifting the traces from the DS4 dataset (average of $1,024$ traces).

Table 4: CNN architectures for different jitter on DS4 in a fixed key scenario.

Hyperparameters	**Tested Values**	**Jitter 5**	**Jitter 10**	**Jitter 15**	**Jitter 20**
Convolutional groups	[1, 2, 3, 4, 5]	5	5	5	5
Convolutional units	[4, 8, 10, 12, 14, 16]	16	12	14	14
Dense layers	[1, 2, 3, 4]	1	1	4	4
Dense units	[128, 256, 512, 1,024]	1,024	256	128	256
Convolutional pool size	[1, 2, 3, 4, 5, 6]	6	4	6	5
Convolutional strides	[1, 2, 3, 4]	4	2	3	3
Convolutional kernel size	[2, 8, 10, 11, 12, 15]	15	11	15	11
Convolutional regularization	[Yes, No]	No	Yes	Yes	Yes
Dense regularization	[Yes, No]	No	Yes	Yes	Yes
Dropout	[0.2, 0.4, 0.6, 0.8]	0.8	0.2	0.4	0.4
Optimizer	[Adam, RMSprop]	RMSprop	RMSprop	RMSprop	RMSprop
Activation function	[ReLU, SELU]	SELU	ReLU	ReLU	ReLU
Accuracy	**Train f1-score**	0.975	1.0	0.946	0.707
	Valid f1-score	0.999	0.998	0.907	0.673
	Test f1-score	0.996	0.991	0.974	0.668

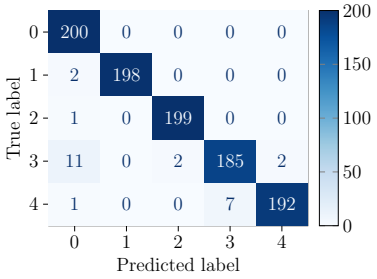
performance. Higher dropout rates (0.8) were optimal for jitter 5, while lower rates ($0.2 - 0.4$) were better for higher jitter levels. Both convolutional and dense regularizations were beneficial under higher jitter conditions (they were not needed for jitter 5), emphasizing the need for regularization to combat overfitting and maintain model robustness.

Optimizer *RMSprop* was consistently the best optimizer across all jitter levels. Regarding activation functions, SELU was effective for jitter 5, while ReLU

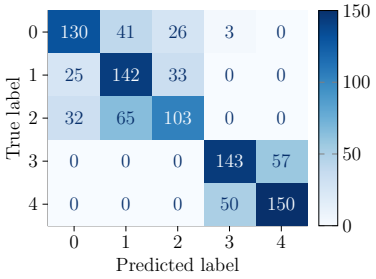


(a) Jitter 5.

(b) Jitter 10.



(c) Jitter 15.



(d) Jitter 20.

Figure 10: Confusion matrices for test scenario: jitter impact – fixed key.

was more suitable for higher jitter levels.

The models exhibited varying performance metrics across different jitter levels. For jitter 5, the train f1-score was 0.975, validation f1-score was 0.999, and test f1-score was 0.996, indicating high accuracy. However, as jitter levels increased, performance metrics generally declined. For jitter 20, the train f1-score dropped to 0.707, with corresponding decreases in validation (0.673) and test f1-score (0.668). This trend underscores the challenge of maintaining high performance under increased jitter conditions.

For example, in Figure 10a, the model shows high accuracy, with only four misclassifications. In Figure 10b, the misclassifications increase slightly, to nine misclassifications. In Figure 10c, the model maintains high accuracy, but there are more errors compared to jitters 5 and 10, resulting in 26 misclassifications in total. Finally, in Figure 10d, there is a significant drop in accuracy with 332 misclassifications, making this scenario unfit for a successful attack. In summary, as the jitter value increases, the classification accuracy decreases, with the most noticeable decline at a jitter value of 20.

In summary, the escalation of jitter-induced perturbations is manifest in the differential traces, as depicted by the increasing noise and irregularities from jitter values of 5 to 20. This observation underscores the critical importance of mitigating jitter to preserve high classification accuracy when deploying CNNs for such analytical purposes. The results indicate that elevated jitter levels significantly impair the model's performance, emphasizing the necessity for robust

jitter control methodologies.

4.4 Test Scenario: Noise Impact – Random Key

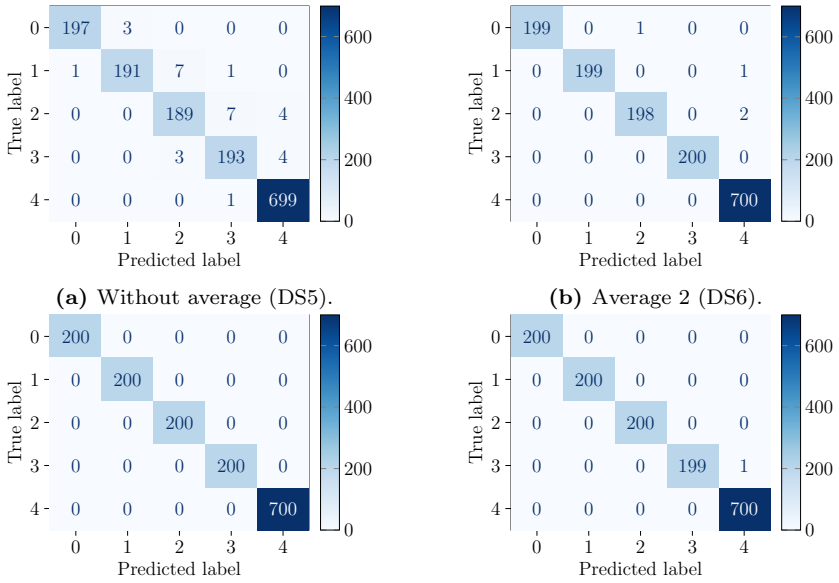
In the previous subsections, high prediction accuracies using CNNs were achieved. In this part we shift to a different scenario – instead of using a fixed key for all the encryptions, we use a random key for each plaintext pair. The initial dataset again contained an equal number of power traces for all the classes. Similarly to the previous case, a high detection accuracy was attained. To further improve accuracy, the NN model was fine-tuned with an additional dataset to recognize undesirable collisions, specifically, collisions across multiple columns falling into class 4. Specifically, we extended the dataset by 250 differential power traces for each combination of active columns (the extended dataset contains a total of 3500 differential power traces for class 4). After this fine-tuning, a further increase in classification accuracy was observed. Table 5 lists the best-tuned CNN parameter values for different averaging sizes. We again used the same averaging values ($0, 2, 4$, and 1024) and titled the resulting datasets DS5, DS6, DS7 and DS8, respectively.

Table 5: CNN architecture for the training dataset in a random key scenario.

Hyperparameters	**Tested Values**	**DS5** **0 average**	**DS6** **2 average**	**DS7** **4 average**	**DS8** **1024 average**
Convolutional groups	[1, 2, 3, 4, 5]	5	5	4	3
Convolutional units	[4, 8, 10, 12, 14, 16]	14	16	12	16
Dense layers	[1, 2, 3, 4]	1	1	3	3
Dense units	[128, 256, 512, 1,024]	512	1,024	128	256
Convolutional pool size	[1, 2, 3, 4, 5, 6]	5	6	3	1
Convolutional strides	[1, 2, 3, 4]	2	4	2	4
Convolutional kernel size	[2, 8, 10, 11, 12, 15]	12	15	15	8
Convolutional regularization	[Yes, No]	Yes	No	No	No
Dense regularization	[Yes, No]	No	Yes	No	No
Dropout	[0.2, 0.4, 0.6, 0.8]	0.6	0.2	0.6	0.4
Optimizer	[Adam, RMSprop]	RMSprop	RMSprop	RMSprop	Adam
Activation function	[ReLU, SELU]	ReLU	SELU	SELU	SELU
Accuracy	**Train f1-score**	0.971	1.0	0.947	0.986
	Valid f1-score	0.960	0.992	0.998	0.994
	Test f1-score	0.979	0.997	1.0	0.999

Table 5 presents the optimal configurations for our proposed models on these datasets. The number of convolutional units and groups differs between $12-16$. For instance, 16 convolutional units were optimal for both 2 and $1,024$ averages, while 14 and 12 units were better for 0 and 4 averages, respectively. The number of 5 convolutional groups was optimal for 0 and 2 averages, 4 groups for 4 averages, and 3 groups for 1024 averages. This suggests that higher averaging levels might benefit from fewer convolutional groups.

The number of dense layers and units also significantly influenced performance. A single dense layer was sufficient for 0 and 2 averages, whereas 3 dense layers were necessary for 4 and 1024 averages. Dense units showed variability, with higher values (1024) for 2 averages and lower values ($128-512$) for others.



(a) Without average (DS5). **(b)** Average 2 (DS6).

(c) Average 4 (DS7). **(d)** Average 1024 (DS8).

Figure 11: Confusion matrices on test scenario: noise impact – random key.

This indicates that increasing the number of dense layers and adjusting dense units are essential for handling higher levels of averaging.

Dropout rates and regularization strategies were critical for maintaining model accuracy. Higher dropout rates (0.6) were effective for 0 and 4 averages, while lower rates ($0.2 - 0.4$) were better for 2 and 1024 averages. Convolutional regularization was only beneficial for 0 averages, and dense regularization was advantageous for 2 averages. This highlights the need for tailored regularization approaches to optimize model performance based on the averaging level.

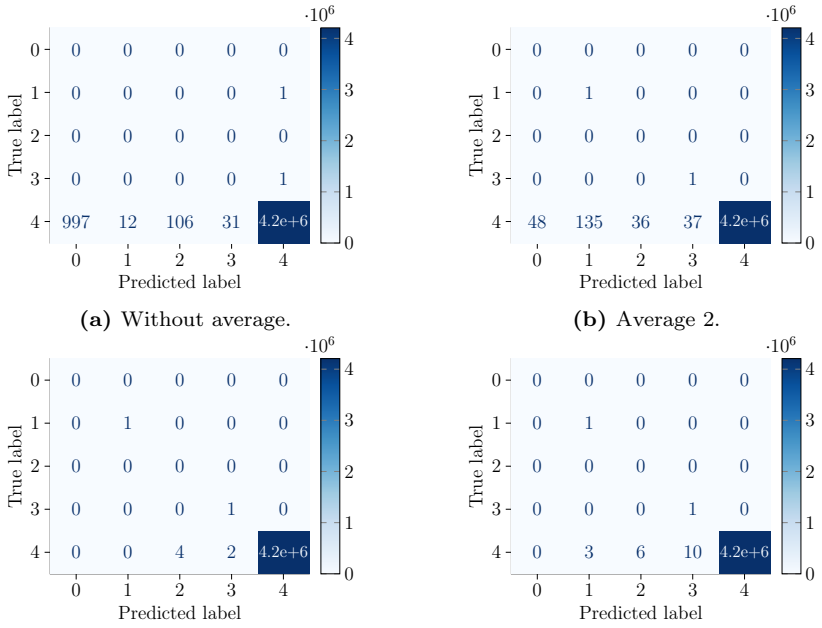
The choice of optimizer and activation function was pivotal across different averaging levels. *RMSprop* was consistently the best optimizer for 0, 2 and 4 averages, while *Adam* was better for 1024 averages. Regarding activation functions, ReLU was effective for 0 averages, whereas SELU was more suitable for higher averaging levels. This underscores the importance of selecting appropriate optimization and activation strategies tailored to the specific averaging level.

The models exhibited high-performance metrics across different averaging levels. For 2 averages, the train f1-score reached 1.0, with a validation f1-score of 0.992 and a test f1-score of 0.997. Similarly, 4 and 1024 averages also showed excellent performance, with test f1-scores of 1.0 and 0.999, respectively. The metrics for 0 averages were slightly lower but still high, with a train f1-score of 0.971, validation f1-score of 0.960, and test f1-score of 0.979. These metrics demonstrate the models' robustness and accuracy across varying averaging levels.

The confusion matrix in Figure 11 indicates highly accurate models with minimal misclassifications. While the DS5 without averaging still contained 30 false negatives, the averaged datasets exhibited strong performance with only four misclassifications for DS6, one misclassification for DS8, and no misclassification for DS3.

4.5 Real-World Attack Scenario

In this section, a dataset was created consisting of power traces from known plaintexts without the knowledge of the secret key. The plaintexts were generated by random modifications to the main diagonal. According to the original article [BBH+20], a collision occurs in approximately $2^{11.5}$ plaintexts. Again, we created four datasets with different averaging levels ($0, 2, 4$ and 1024). Each generated dataset contained 2901 power traces. Utilizing all the combinations of these traces results in the creation of 4206450 differential traces (slightly more than 2^{22}). Analyzing the dataset, we found out that in total, 2 collisions were present, in classes 1 and 3. In these generated differential power traces, CNNs trained with a random key dataset (see Section 4.4) were used to identify these collisions. To enhance the prediction accuracy for the dataset, we introduced a threshold mechanism, effectively reducing the number of false negative predictions. Figure 12 shows the confusion matrices for this dataset. In the following, we will discuss the results.



(a) Without average. **(b)** Average 2.

(c) Average 4. **(d)** Average 1024.

Figure 12: Confusion matrices for real-world attack scenario.

In the case of the dataset without averaging, the CNN exhibits the lowest accuracy, as expected. While it might seem that the majority of differential traces are classified correctly, the dataset is highly imbalanced with just 2 valid collisions. Two valid collisions present in the dataset were not identified.

In the dataset for average 2, the model improves on the classification accuracy. Misclassifications are slightly reduced compared to no averaging and what is important, the valid collisions were correctly classified in this case. In the dataset for average 4, the model demonstrates the highest accuracy for this scenario. Only 6 differential traces were misclassified, and the valid collisions were correctly classified, thus allowing to mount a key recovery attack for the remaining key candidates. In the dataset for average 1024, the model also achieves a very high accuracy, albeit lower than in the previous case. In this case, 19 differential traces were incorrectly classified, and again, the valid collisions were identified.

Overall, the results indicate that the CNN is highly effective in classifying differential power traces, with a marked decrease in misclassification rates as the average increases. This demonstrates the robustness of the CNN in identifying valid collisions even when varying levels of averaging are applied.

DL-SITM Full Key Recovery

In this part, we attempt to answer the question: *How expensive is it to convert the observed leakage to full key-recovery?*

For each AES state diagonal, examining $2^{11.5}$ plaintexts yields 2^{22} differential pairs. With a 2^{-22} probability of a good differential, we expect at least one suitable pair per diagonal with $1 - 1/e \approx 63.21\%$ probability. This gives an overall success rate of $\approx 15.96\%$ for all four diagonals. Each collision reduces its diagonal's key space to $\approx 2^8$ candidates. Let N be the total key candidates (across both true and false positive collisions) and t the number of 4-tuples of valid collisions satisfying all four state diagonals. The median number of trials before finding the key is exactly $N/2$ when $t = 1$, and decreases as a function of N when t increases. Overall success probability can be increased to 50 percent simply by increasing the number of plaintexts to $\approx 2^{11.96}$ per diagonal.

In our experiments with 4-fold averaged traces, we identified 8 potential collisions across all columns using $2^{11.5+2} = 2^{13.5}$ plaintexts, yielding $1538 \approx 2^{10.59}$ key candidates for the tested diagonal. Assuming all four diagonals behave similarly (a simplifying assumption that should mildly underestimate attack costs), this results in approximately 2^{43} remaining candidates for the full 16-byte key. The median cost of finding the key is at most half the number of remaining candidates, and will be lower if $t > 1$. Given AES-NI acceleration, testing one key should cost ≈ 200 clock cycles including some overhead [Gue12], so a $\approx 2^{43}$ workload can be done in less than a day on a 12-core PC.

5 Discussion

Worse Performance in a 1024-fold Averaging Regime Compared to 4-fold Averaging. It might seem surprising that 4-fold averaging does better in our tests than 1024-fold averaging. However, in the following we list plausible reasons for this:

- *Removal of features by averaging:* CNN models rely on fine-grained, and often non-linear, features. There is no reason to suppose that these would survive averaging. This is in contrast to classical template models, which model the expected signal at the points of interest explicitly as some base signal plus noise.
- *Model generalization:* Overfitting may be easier on very smooth data (1024-fold averaging). Related to this, any steps in a training pipeline that artificially add some form of noise (in our case, maybe dropout) will arguably take smoothed data farther out of distribution than original data.

Potential Improvements of the Classifiers. Several ways to potentially improve the attack exist. Misclassifications could be reduced through multi-stage classifiers or by combining information from multiple related leakage targets similar to what SASCA-like attacks do [VCGS14]. Jitter resilience might be enhanced by processing trace pairs directly or alignment of traces by pre-processing. Directly processing pairs of traces would also allow to combine predictions optimally instead of averaging trace data. We leave these possible improvements to future research.

6 Conclusion

In this paper, we improve on the practical realization of the SITM attack proposed in [BBH+20]. By utilizing deep learning models, we can automate the attack steps, removing the need to visually inspect over 4M differential traces. The presented DL-SITM attack methodology is capable of handling higher levels of noise, reducing the need to average the traces from 1000 to 4. Apart from that, it can analyze traces with jitter of up to ± 15 samples.

References

[AKH20] Eslam G AbdAllah, Yu Rang Kuang, and Changcheng Huang. Advanced encryption standard new instructions (aes-ni) analysis: Security, performance, and power consumption. In *Proceedings of the 2020 12th International Conference on Computer and Automation Engineering*, pages 167–172, 2020.

[Bak22] Anubhab Baksi. Classical and physical security of symmetric key cryptographic algorithms. Springer, 2022. https://link.springer.com/book/10.1007/978-981-16-6522-6.

[BBH+20] Shivam Bhasin, Jakub Breier, Xiaolu Hou, Dirmanto Jap, Romain Poussier, and Siang Meng Sim. Sitm: See-in-the-middle side-channel assisted middle round differential cryptanalysis on spn block ciphers. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 95–122, 2020.

[BCH+22] Lejla Batina, Lukasz Chmielewski, Björn Haase, Niels Samwel, and Peter Schwabe. Sok. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 557–589, 2022.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6*, pages 16–29. Springer, 2004.

[BJB18] Jakub Breier, Dirmanto Jap, and Shivam Bhasin. Scadpa: Side-channel assisted differential-plaintext attack on bit permutation based ciphers. In *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1129–1134. IEEE, 2018.

[BJHB20] Jakub Breier, Dirmanto Jap, Xiaolu Hou, and Shivam Bhasin. On side channel vulnerabilities of bit permutations in cryptographic algorithms. *IEEE Transactions on Information Forensics and Security*, 15:1072–1085, 2020.

[BKL+07] Andrey Bogdanov, Lars R Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew JB Robshaw, Yannick Seurin, and Charlotte Vikkelsoe. Present: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems-CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings 9*, pages 450–466. Springer, 2007.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ascad database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.

[CA16] M Emre Celebi and Kemal Aydin. *Unsupervised learning algorithms*, volume 9. Springer, 2016.

[CBB21] Andrea Caforio, Fatih Balli, and Subhadeep Banik. Complete practical side-channel-assisted reverse engineering of aes-like ciphers. In *International Conference on Smart Card Research and Advanced Applications*, pages 97–117. Springer, 2021.

[CRR03] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems-CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers 4*, pages 13–28. Springer, 2003.

[DR99] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[FLD12] Yunsi Fei, Qiasi Luo, and A.Adam Ding. A statistical model for DPA with novel algorithmic confusion analysis. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems – CHES 2012*, volume 7428 of *Lecture Notes in Computer Science*, pages 233–250. Springer Berlin Heidelberg, 2012.

[Gue12] Shay Gueron. Intel advanced encryption standard (aes) new instructions set. Technical Report 323641-001, Intel Corporation, September 2012. Revision 3.01.

[HB24] Xiaolu Hou and Jakub Breier. *Cryptography and Embedded Systems Security*. Springer Nature Switzerland, 2024.

[HBB21] Xiaolu Hou, Jakub Breier, and Shivam Bhasin. Dnfa: Differential no-fault analysis of bit permutation based ciphers assisted by side-channel. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 182–187. IEEE, 2021.

[HBB22] Xiaolu Hou, Jakub Breier, and Shivam Bhasin. Sbcma: Semi-blind combined middle-round attack on bit-permutation ciphers with application to aead schemes. *IEEE Transactions on Information Forensics and Security*, 17:3677–3690, 2022.

[HTF+09] Trevor Hastie, Robert Tibshirani, Jerome Friedman, Trevor Hastie, Robert Tibshirani, and Jerome Friedman. Overview of supervised learning. *The elements of statistical learning: Data mining, inference, and prediction*, pages 9–41, 2009.

[KDB+22] Satyam Kumar, Vishnu Asutosh Dasu, Anubhab Baksi, Santanu Sarkar, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Side channel attack on stream ciphers: a three-step approach to state/key recovery. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(2):166–191, 2022.

[KJJ99] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology—CRYPTO'99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings 19*, pages 388–397. Springer, 1999.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 348–375, 2020.

[MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007.

[MWM21] Thorben Moos, Felix Wegener, and Amir Moradi. Dl-la: Deep learning leakage assessment: A modern roadmap for sca evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 552–598, 2021.

[PDJ+19] Sikhar Patranabis, Nilanjan Datta, Dirmanto Jap, Jakub Breier, Shivam Bhasin, and Debdeep Mukhopadhyay. Scadfa: Combined sca+ dfa attacks on block ciphers with practical validations. *IEEE Transactions on Computers*, 68(10):1498–1510, 2019.

[PK22] Jonghyun Park and Jongsung Kim. See-in-the-middle attacks on blockciphers aria and default. In *International Conference on Information Security and Cryptology*, pages 3–16. Springer, 2022.

[PKK21] Jonghyun Park, Hangi Kim, and Jongsung Kim. Improved see-in-the-middle attacks on aes. In *International Conference on Information Security and Cryptology*, pages 271–279. Springer, 2021.

[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.

[PSK+18] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In *Security, Privacy, and Applied Cryptography Engineering: 8th International Conference, SPACE 2018, Kanpur, India, December 15-19, 2018, Proceedings 8*, pages 157–176. Springer, 2018.

[SL09] Marina Sokolova and Guy Lapalme. A systematic analysis of performance measures for classification tasks. *Information processing & management*, 45(4):427–437, 2009.

[SM23] Marvin Staib and Amir Moradi. Deep learning side-channel collision attack. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 422–444, 2023.

[SMY09] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28*, pages 443–461. Springer, 2009.

[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.

[VCGS14] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20*, pages 282–296. Springer, 2014.

[WHJ+21a] Yoo-Seung Won, Dong-Guk Han, Dirmanto Jap, Shivam Bhasin, and Jong-Yeon Park. Non-profiled side-channel attack based on deep learning using picture trace. *IEEE Access*, 9:22480–22492, 2021.

[WHJ+21b] Yoo-Seung Won, Xiaolu Hou, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Back to the basics: Seamless integration of side-channel pre-processing in deep neural networks. *IEEE Transactions on Information Forensics and Security*, 16:3215–3227, 2021.

[WPP23] Lichao Wu, Guilherme Perin, and Stjepan Picek. Hiding in plain sight: Non-profiling deep learning-based side-channel analysis with plaintext/ciphertext. *Cryptology ePrint Archive*, 2023.

[ZBHV20] 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*, pages 1–36, 2020.