Deep Learning-based Side-channel Analysis against AES Inner Rounds

Sudharshan Swaminathan[1], Lukasz Chmielewski[2], Guilherme Perin[1], and Stjepan Picek[1]

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

Abstract. Side-channel attacks (SCA) focus on vulnerabilities caused by insecure implementations and exploit them to deduce useful information about the data being processed or the data itself through leakages obtained from the device. There have been many studies exploiting these side-channel leakages, and most of the state-of-the-art attacks have been shown to work on systems implementing AES. The methodology is usually based on exploiting leakages for the outer rounds, i.e., the first and the last round. In some cases, due to partial countermeasures or the nature of the device itself, it might not be possible to attack the outer round leakages. In this case, the attacker has to resort to attacking the inner rounds.

This work provides a generalization for inner round side-channel attacks on AES and experimentally validates it with non-profiled and profiled attacks. This work *formulates the computation of the hypothesis values of any byte in the intermediate rounds*. The more inner the AES round is, the higher is the attack complexity in terms of the number of bits to be guessed for the hypothesis. We discuss the main limitations for obtaining predictions in inner rounds and, in particular, we compare the performance of Correlation Power Analysis (CPA) against deep learning-based profiled side-channel attacks (DL-SCA). We demonstrate that because trained deep learning models require fewer traces in the attack phase, they also have fewer complexity limitations to attack inner AES rounds than non-profiled attacks such as CPA. This paper is the first to propose deep learning-based profiled attacks on inner rounds of AES under several time and memory constraints to the best of our knowledge.

1 Introduction

In the past twenty years, a large number of academic and industrial research provided methods to attack and protect the Advanced Encryption Standard (AES) implementations. Among these attacks, side-channel analysis (SCA) targets unintentional leakages from software and hardware implementations. The aim can be twofold: from the designer's perspective (the defensive side), a side-channel analysis indicates a potential source of leakages in the implemented algorithm. Additionally, the analysis provides important directions to design countermeasures to mitigate such attacks. On the other hand, an evaluator (offensive side)

is interested in verifying the worst-case security to advise the manufacturer or certify the implementation against specific types of SCAs and applications. Besides different perspectives, one must consider different types of side-channel attacks. One common division is into non-profiled attacks and profiled attacks. New forms of non-profiled and profiled SCA encounter in AES a suitable target to validate proposed methods. In this sense, most works concentrate their effort in attacking the first and last AES (encryption or decryption) rounds, leaving the attacks on inner rounds out of scope.

The reason stems from the attack complexities and assumptions: attacking the outer rounds requires a minimal effort in terms of key guessing and the number of measurements. On the other hand, several design reasons could limit a side-channel attack application on the outer (i.e., first and last) rounds. Countermeasures (as they add costs overheads to the design) could be applied only to these outer rounds, leaving inner rounds unprotected. In this case, the only side-channel attack mitigations are the inherent sources of noise and misalignments. Additionally, for faster encryption or decryption processes, it is common to implement several AES rounds within a single clock cycle, which is a highly adopted mechanism for hardware-based implementations. This limits the side-channel leakages of the AES intermediate bytes that do not coincide with the clock cycles edges.

The past (and not very recent) literature already proposed various differential power analysis (DPA) attacks to inner AES rounds. In [11], the authors described a DPA attack on round 2, requiring the same attack complexity (8 bits) as attacking round 1 and with an overhead in the required number of measurements due to the chosen-input nature of the attack. Lu et al. investigated how many rounds of an AES implementation should be protected to be secure against power analysis attacks [13]. They provided two main conclusions: attack the inner rounds of AES is possible at the cost of increasing the data complexity and that any attack requiring a DPA on more than 32 bits is considered infeasible and is therefore not explored.

In this work, we extend the formulation of [13] and provide a theoretical generalization of such an attack on all the inner rounds of AES-128. This generalization would give the designers a comprehensive understanding of the complexity of the attack at each round and the threat profile that the attacker needs to have to make a successful attack. This work assumes that inner rounds are not protected by specific countermeasures (e.g., first-order masking or multiple rounds within a single clock cycle) but only by inherent noise and misalignment. Afterward, we run both non-profiled attacks (CPA) and profiled attacks (deep learning-based SCA) and show that deep learning-based SCA reaches significantly better attack performance and succeeds in scenarios where CPA does not indicate a successful key recovery.

Our Contributions.

1. We first generalize the computation of hypothesis for any byte in the intermediate rounds for AES-128 in the encryption mode with some predefined conditions in mind and use the same to determine the relative difficulty of

such attacks. Due to the non-linear substitutions in each AES round, targeting any intermediate byte after n *S-boxes* requires an attack complexity of $8 \times n$ bits. The attack complexity in terms of the number of bits represents the bit-length of guessed hypothesis. This introduces significant time and memory overheads to mount such a high complexity attack.
2. To make our analysis more realistic, we introduce potential countermeasures (such as Gaussian noise and misalignment) to power traces collected from an unprotected AES.
3. The training phase of a deep learning-based profiled attack on inner rounds is not affected by the increased attack complexity. Consequently, we show that the attack phase from the deep learning-based approach is a considerable improvement over limitations faced by non-profiled CPA due to the added countermeasures, especially when the attack complexity is higher than 16 bits. In this case, the attacker faces strong time and memory limitations in terms of processed attack traces.
4. In scenarios when CPA cannot succeed due to implicit countermeasures (which is a practical case shown in this paper on encryption round 3), a convolutional neural network-based profiled attack can easily recover the key even with a very limited number of attack traces.
5. As we specify in this paper, the variability in target intermediate values of profiling traces is only limited by the number of possible plaintexts and key combinations. However, repeating some plaintext-key combinations does not negatively impact the profiling phase.

2 Preliminaries

2.1 Correlation Power Analysis (CPA)

CPA is a statistical method used to correlate the side-channel traces with the observed leakage [3]. There, an attacker has to perform numerous encryptions/decryptions and collect the traces. A hypothesis for each key guess can then be obtained by using a leakage model. CPA uses Pearson Correlation for differentiating between the modeled and the actual power traces.

Pearson Correlation increases towards $+1$ in case observations are directly correlated, and towards -1 in case they are inversely correlated. In the case that they are independent of each other, the value is closer to 0. Adopting a divide and conquer approach, we take one key byte and its corresponding hypothesis at a time. So considering n traces, T data points where t corresponds to one data point in time, $p_{i,t}$ would be the measurement or the data point at the ith trace and t time. We use the leakage model to derive a hypothesis power consumption, $h_{i,k}$ denoting the hypothesis for ith trace and for key guess k. The correlation

between the hypothesis and the measured traces is then computed as:

$$r_{k,t} = \frac{\sum_{i=1}^{n}[(h_{i,k} - \bar{h_k})(p_{i,t} - \bar{p_t})]}{\sqrt{\sum_{i=1}^{n}(h_{i,k} - \bar{h_k})^2 \sum_{i=1}^{n}(p_{i,t} - \bar{p_t})^2}}. \quad (1)$$

Here, $r_{k,t}$ gives the correlation value for one key guess and for one data point in time. The same is done for all the selected data points (the number of which would depend on the captured leakage and the attack point) for each key guess. The maximum absolute value of these correlations computed for all data points is then used as the actual correlation value for that key guess. The key guess having the highest absolute correlation value is most probable to be the actual key byte.

2.2 Deep Learning Methodologies

Deep learning-based SCA (DL-SCA) provides an improvement over other profiled attacks such as template attacks [5] in terms of efforts during pre-processing of traces and effectiveness of the attack. Deep learning methodologies take the traces along with their labels in the profiling phase across the selected data points in time, run them through the defined model, and determine the weights according to the defined criteria such as high accuracy and minimal loss. The labels here depend on the leakage function and the key hypotheses. The input layer of the DL model contains the measurements of the traces across the data points in time, and the output layer contains output nodes for each of the classes defined by the leakage model. These trained weights are then used in the attack phase to determine the probabilities of each of the classes given by the intermediate value corresponding to each key guess. The key guess having the highest probability values would indicate the most likely secret key.

In this work, we use convolutional neural networks (CNNs) to conduct the deep learning-based SCA. We employ CNN with VGG-like architecture as it is a prominent model used for SCA, see, e.g., [1,10]. The original model was developed for the purpose of image classification, where the input signal has multiple input dimensions starting from 2. As SCA has only one spatial dimension considering its data points in time, the main difference that VGG-like architectures introduce is how it handles 1-dimension input signal on each of its convolution and pooling operations.

A CNN is a model which is a combination of convolutional layers, pooling layers, and fully-connected layers. The convolutional operation involves a filter bank applied on the input signal across time t (time steps in the case of SCA owing to the 1-dimensional property of the side-channel traces):

$$(\phi * x)(t) = \sum_{a=-\infty}^{\infty} x(t)\phi(t-a), \quad (2)$$

where $\phi \in \mathbb{R}^{i \times o \times s}$ is a filter with i input channels, o output channels, and s filter length. A pooling layer is a non-linear layer that applies down-sampling over the given input on a particular axis using techniques such as average or maximum of multiple values. This is done to reduce the spatial size of the channels, thereby limiting the number of neurons. Fully-connected layers are layers where every input signal can be mapped to an output signal of that layer, i.e., every neuron is connected with all the neurons in the neighborhood layer. This is usually done by taking a dot product between the weight matrix and the input vector. Then, this VGG-like CNN `cnn`, is represented as follows [10]:

$$\texttt{cnn} = \texttt{fc}_{\theta,\texttt{softmax}} \circ \prod_{p=1}^{P} \texttt{fc}_{\theta^P,\texttt{ReLU}} \circ \prod_{q=1}^{Q} (\texttt{pool} \circ \prod_{r=1}^{R_q} \texttt{conv}_{\phi^r,\texttt{ReLU}}), \tag{3}$$

where P, Q represents number of fully-connected layers, `fc`, and convolutional layer blocks respectively. The latter themselves are a combination of pooling layers and individual convolutional layers `conv`, wherein R_q represents the number of convolutional layers `conv` in the qth convolutional block. $\texttt{conv}_{\phi,\sigma}$ and $\texttt{fc}_{\theta,\sigma}$ are convolutional and fully-connected layers respectively and are defined as:

$$\begin{aligned} \texttt{conv}_{\phi,\sigma}(X) &= \sigma(\phi * X), \\ \texttt{fc}_{\theta,\sigma}(x) &= \sigma(\theta^T x). \end{aligned} \tag{4}$$

Here, $X \in \mathbb{R}^{i \times d}$ is the input with i channels and d length and $x \in \mathbb{R}^f$ is the input vector for the fully-connected layers with f dimensions. $\theta \in \mathbb{R}^{f \times h}$ is a projection matrix that applies the weight matrix transforming the f dimensional input to h dimensional output (the bias term has been omitted for simplicity). Finally, σ is an activation function, where `ReLU` is usually used for hidden layers and `softmax` is used for the final output layer, representing the probabilities of each of the defined classes.

2.3 Attack Evaluation Methodology

The most commonly used metric for evaluating the performance of a side-channel attack is key rank. We use the same for evaluating the performance of the attacks carried out in this work. An average key rank (denoted guessing entropy) represents the average number of keys the attacker needs to go through during the attack to reveal the actual key successfully [21]. As seen in the above sections, we obtain a posterior distribution of probabilities for each of our defined classes as the output of the attacks. The key guess contributing the most to the highest probable predicted class across the attack traces is predicted to be the key byte being used. Consequently, the output vector that is obtained during the attack is of the form $\mathbf{k} = [k_0, k_1, k_2, ..., k_{|K|-1}]$, where $|K|$ is the size of the keyspace. These key guesses contained in the vector $\mathbf{k}$ are then ordered in the decreasing order of probability, that is, k_0 is the most probable key guess, also known as the best guess, and $k_{|K|-1}$ is the least probable key guess. We then check the position at which the actual key byte resides in this ordered list, and this position of the actual key byte is termed as the key rank.

3 Related Work

3.1 On Attacking the AES Inner Rounds

The first and the last rounds, being dependent on a relatively small fraction of the key, are more vulnerable and are therefore primary targets of side-channel attacks. As we go into the inner rounds, every intermediate byte would depend on an increasing number of key bytes due to the diffusion properties of AES, thereby increasing the data complexity of the attack. The trade-off, therefore, focuses on protecting the first and the last rounds and leaving other intermediate rounds unprotected or with very simple countermeasures [22,7]. In some cases of hardware implementation, it is also possible that multiple rounds are executed within one clock cycle. This would result in the inner rounds being exposed, i.e., it would then be possible to capture traces corresponding to the inner rounds. In such cases, the hypothesis built for the first round would not correlate to the captured traces, and the attack would not work. *Such cases, along with the hindrance caused by the partial countermeasures, raise the need to look into attacks on unprotected or even partially protected inner rounds and understand the resources that the attacker would need to launch such attacks.*

While Jaffe et al. already described a DPA attack after the SubBytes of round 2 [11], Lu et al. answered an important question about how many rounds of an AES implementation should be protected for it to be secure against power analysis attacks [13]. To this end, they show that it is possible to attack the inner rounds of AES at the cost of increasing the data complexity of the attack. They define the feasibility of an attack by the number of bits required to launch the DPA/CPA and set this threshold at 32 bits. Consequently, any attack requiring a DPA on more than 32 bits is considered infeasible and, as such, not investigated by them.

We extend on the same and formulate a generalization of such an attack on the inner rounds.

3.2 On Machine Learning-based SCA

Many approaches have been developed in SCA, from statistical methods such as CPA/DPA to template attacks and machine learning-based approaches. While the former ones have been studied extensively, attacks based on profiling involving machine learning and deep learning are still developing. Already studies appearing one decade ago showed that machine learning could be used to mount successful side-channel attacks that are also more effective than template attacks [8,9]. Machine learning methods such as SVM have also been used to defeat masked implementations, as shown by Lerman et al. [12]. Extending on the same, Gilmore et al. showed that neural networks could also be used to tackle the masking countermeasure and are more effective than the other machine learning-based approaches [6]. However, these implementations depend on a crucial assumption that the random masks are available to the attacker during the profiling phase, which as mentioned by [6] is an impractical assumption.

As discussed before, most of the practical and efficient countermeasure implementations involve only the outer rounds [22,7]. Therefore, we can bypass these countermeasures if we attack the inner rounds directly, which would also not necessitate having the random masks used by the target implementation.

Deep learning (more precisely, convolutional neural networks and multilayer perceptrons) has been successfully used to attack AES implementations as first shown by Maghrebi et al. [15]. Next, Cagli et al. showed that convolutional neural networks could break implementations protected with the jitter countermeasure, especially if the attack is augmented with synthetic data obtained from data augmentation techniques [4]. Kim et al. discussed the VGG-like architecture that showed good attack performance for several datasets, where some were using masking or hiding countermeasures [10]. Benadjila et al. introduced the ASCAD dataset, which is a dataset used in most of the SCA studies today, and also investigated the hyperparameter tuning to find architectures leading to successful attacks [1]. Picek et al. showed that metrics commonly indicating the performance of machine learning algorithms are not appropriate to assess the SCA performance [17]. Zaid et al. proposed a methodology to design convolutional neural network architectures that have a small number of trainable parameters and that result in efficient attacks [25]. Wouters et al. further discussed the methodology perspective, providing even smaller neural network architectures that perform well [24]. Perin et al. explored how deep learning-based SCA generalizes to previously unseen examples and showed that ensembles of random neural networks could outperform even state-of-the-art neural network architectures [16]. Rijsdijk et al. introduced the reinforcement learning approach for designing neural networks that perform well and are as small as possible [18].

These studies represent only a fraction of works exploring machine learning-based side-channel attacks, but to the best of our knowledge, none of those works consider attacking inner rounds of AES.

4 First-Order Non-Profiled Attacks on AES Inner Rounds

Lu et al. [13] give five general principles for attacking bytes in the inner rounds of AES using first and second-order DPA. These principles consider the attack to be feasible as long as the attack is on less than 32 bits. However, since our aim is to generalize the attack on any intermediate byte and observe the complexity of such an attack, the feasibility of the attack itself is not a factor that we consider here. We focus on the following two principles listed by [13] that are based on the first-order DPA:

1. Attacking from input: any intermediate byte before the MixColumns operation of round 3 can be exploited by conducting a first-order DPA attack and will depend on the part of the plaintext bytes being fixed.
2. Attacking from the output: any intermediate byte resulting from the AddRoundKey operation of round 7 can be exploited to conduct a first-order DPA attack and will depend on some of the ciphertext bytes being fixed.

Note: Although Lu et al. [13] consider any byte after the AddRoundKey operation of round 7, we noticed that it was also possible to attack from output before the AddRoundKey of round 7 while considering single bit DPA attacks.

We now extend over the principle above and start by first attacking in the encryption mode from input at rounds 2, 3, and 4. Next, we attack a byte from the output at round 7. We base these attacks on chosen-plaintext and adaptive chosen-ciphertext attacks and adopt the computation of a byte at rounds 2 and 3 from the work of Lu et al. [13]. Observing the attack on rounds 2 and 3 and then consequently analyzing the same for round 4 helped us to figure out a pattern for generalizing the attack in any intermediate round during encryption. We, therefore, calculate the required number of fixed plaintext bytes and consequently the attack complexity in terms of the number of bits to be guessed while attacking from both input and output for AES encryption mode. Details on how to generalize the attack on any intermediate byte in the inner rounds are provided in Section 4.6. We also observed that while attacks on rounds 2 and 3 were practically feasible using a chosen-plaintexts approach, it was not possible to attack rounds beyond round 4 using this same principle successfully.

4.1 Notations

Before moving onto attacking the inner round bytes, we present the notations used in the following sections.

- Plaintext bytes are denoted by p_i, where i is the index of the byte. Similarly, ciphertext bytes are denoted by c_i.
- The output bytes of an S-box in any round is denoted by v_i^n, where i is the index of the byte and n indicates the round. For example, v_0^1 is the first byte obtained after the S-box in round 1. Similarly, bytes after the MixColumns operation are denoted using u_i^n, while the output bytes of a round, i.e., bytes after the AddRoundKey are denoted by w_i^n.
- The key bytes are denoted by k_i^n and the round key they belong to is denoted by K_n. The initial key would then be $\{k_0^0, k_1^0, ..., k_{15}^0\} \in K_0$, while the last round key would be $\{k_0^{10}, k_1^{10}, ..., k_{15}^{10}\} \in K_{10}$
- S-box in round n is denoted as S_n and we denote its application on an input byte u as $S_n(u)$. The inverse of the S-box is denoted as S_n^{-1}.
- Terms such as γ, δ, θ are used to denote 8-bit constants.

4.2 Attacking a Byte After the S-box at Round 2

In this attack, the goal is to recover K_0, i.e., the first round key in the AES encryption process. To simplify the attack understanding, we start by predicting the first byte immediately after the S-box in round 2. Let this byte be v_0^2, i.e., the first byte in the AES state after S_2. Let the input to S_2 be w_0^1, a resultant byte from the AddRoundKey operation from round 1. This AddRoundKey operation involves XORing a key byte from K_1 (round key from round $i = 1$), say k_0^1, with

the first byte u_0^1 obtained after MixColumns of round 1. This can be written as:

$$\begin{aligned} v_0^2 &= S_2(w_0^1), \\ w_0^1 &= u_0^1 \oplus k_0^1. \end{aligned} \tag{5}$$

The process is illustrated in Figure 1. As mentioned above, u_0^1 is the first byte after the MixColumns operation is applied on 4 bytes of round 1, specifically after ShiftRows. Let the bytes after S_1 be represented as $v_i^1,\ i \in \{0, .., 15\}$, in which case the value of u_0^1 can then be written as:

$$u_0^1 = 02 * v_0^1 \oplus 03 * v_5^1 \oplus 01 * v_{10}^1 \oplus 01 * v_{15}^1, \tag{6}$$

where $*$ represents the field multiplication operation in $GF(2^8)$. Here, we can see that the bytes $(v_5^1, v_{10}^1, v_{15}^1)$ have been used as a consequence of the shuffling caused by ShiftRows. Substituting the value of u_0^1 (and subsequently w_0^1) in Eq. (5) from Eq. (6), we have:

$$v_0^2 = S_2(02 * v_0^1 \oplus 03 * v_5^1 \oplus 01 * v_{10}^1 \oplus 01 * v_{15}^1 \oplus k_0^1). \tag{7}$$

Let us denote $\gamma = 03 * v_5^1 \oplus 01 * v_{10}^1 \oplus 01 * v_{15}^1 \oplus k_0^1$ be a 8-bit constant byte. This constant results from fixing 3 plaintext bytes (p_5, p_{10}, and p_{15}, as shown in Figure 1) across all side-channel measurements. By inserting γ in Eq. (7) we obtain:

$$v_0^2 = S_2(02 * v_0^1 \oplus \gamma). \tag{8}$$

The byte v_0^1 can be written as the result of S_1: $v_0^1 = S_1(p_0 \oplus k_0^0)$. Then Eq. (8) can be rewritten as:

$$v_0^2 = S_2(02 * S_1(p_0 \oplus k_0^0) \oplus \gamma). \tag{9}$$

Having γ as constant, we then need to guess only 16 bits of data, which is (k_0^0, γ) in order to find the key byte k_0^0. This attack on v_0^2 has been diagrammatically represented in Figure 1. Similarly, we can target k_4 by having p_9, p_{14}, and p_3 as constant bytes, and so on. We also note here that if we approach this attack without using chosen plaintexts, we would have to attack/brute force 4 bytes of the key directly instead due to the effect of MixColumns.

4.3 Attacking a Byte After the S-box at Round 3

A similar approach to the one seen for round 2 can be applied for attacking a byte after S-box in round 3 (S_3) as well. Here too, let us take the first byte after S_3 as an example and let this byte be v_0^3. Then the first byte of the input to S_3 is w_0^2, and u_0^2 is the first byte obtained after the MixColumns of round 2. Then we have:

$$v_0^3 = S_3(w_0^2) = S_3(u_0^2 \oplus k_0^2). \tag{10}$$

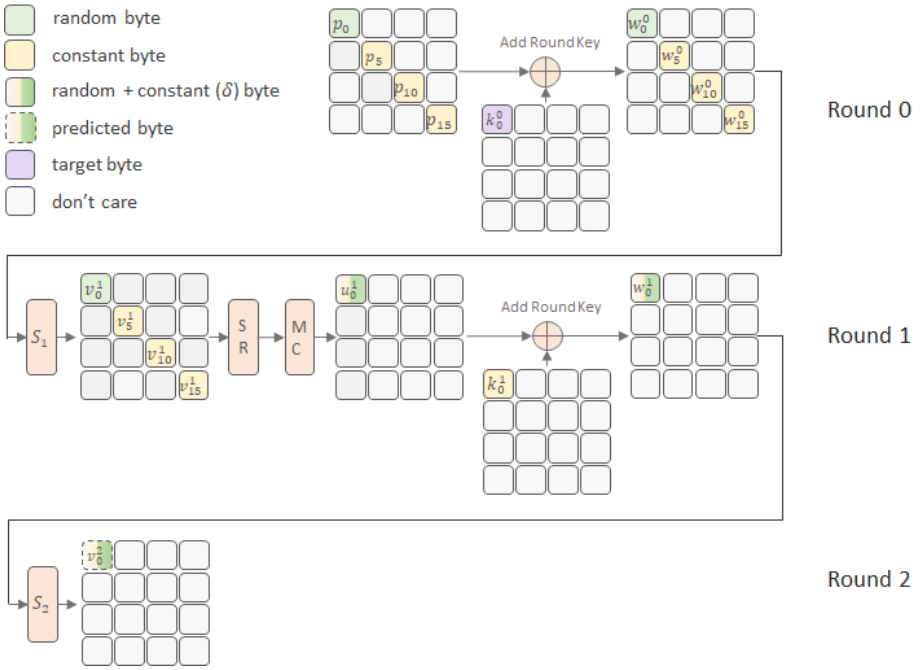

Fig. 1: Attacking k_0^0 by targeting first byte (v_0^2) after S_2.

We follow the approach from Section 4.2, u_0^2 depends on 4 bytes which are input to the MixColumns operation at round 2. So if $v_i^2,\ i \in \{0,..,15\}$ represent the bytes after S_2, we can write the MixColumns operation following the ShiftRows, resulting in u_0^2 as:

$$u_0^2 = 02 * v_0^2 \oplus 03 * v_5^2 \oplus 01 * v_{10}^2 \oplus 01 * v_{15}^2. \tag{11}$$

Substituting the value of u_0^2 in Eq. (10) from Eq. (11), we obtain:

$$v_0^3 = S_3(02 * v_0^2 \oplus 03 * v_5^2 \oplus 01 * v_{10}^2 \oplus 01 * v_{15}^2 \oplus k_0^2). \tag{12}$$

Let us consider $\gamma = 03 * v_5^2 \oplus 01 * v_{10}^2 \oplus 01 * v_{15}^2 \oplus k_0^2$ and substitute this in Eq. (12):

$$v_0^3 = S_3(02 * v_0^2 \oplus \gamma). \tag{13}$$

The bytes $u_i^1,\ i \in \{0,..,15\}$ are the bytes obtained after the MixColumns operation of round 1, then we have:

$$\begin{aligned} v_0^2 &= S_2(u_0^1 \oplus k_1^1), & v_5^2 &= S_2(u_5^1 \oplus k_5^1), \\ v_{10}^2 &= S_2(u_{10}^1 \oplus k_{10}^1), & v_{15}^2 &= S_2(u_{15}^1 \oplus k_{15}^1). \end{aligned} \tag{14}$$

Expanding on u_0^1, we observe an equation similar to Eq. (11), where it depends on 4 bytes $(v_0^1, v_5^1, v_{10}^1, v_{15}^1)$ obtained from the output of S_1:

$$u_0^1 = 02 * v_0^1 \oplus 03 * v_5^1 \oplus 01 * v_{10}^1 \oplus 01 * v_{15}^1. \tag{15}$$

Expanding on this further, each of v_i^1 can be written as a result of S-box on plaintext bytes XORed with the round key bytes of round 0, which can be written as:

$$\begin{aligned} v_0^1 &= S_1(p_0 \oplus k_0^0), \qquad & v_5^1 &= S_1(p_5 \oplus k_5^0), \\ v_{10}^1 &= S_1(p_{10} \oplus k_{10}^0), \qquad & v_{15}^1 &= S_1(p_{15} \oplus k_{15}^0). \end{aligned} \tag{16}$$

From the above, we can conclude that u_0^1 depends on 4 plaintext bytes, meaning that each of $(v_0^2, v_5^2, v_{10}^2, v_{15}^2)$ also depend on 4 plaintext bytes. Similar conclusions can be made for $(u_5^1, u_{10}^1, u_{15}^1)$ as well. Considering $03 * v_5^1 \oplus 01 * v_{10}^1 \oplus 01 * v_{15}^1 \oplus k_0^1 = \delta$ and reformulating Eq. (13), we obtain:

$$v_0^3 = S_3(02 * S_2(u_0 \oplus k_0^1) \oplus \gamma) \implies v = S_3(02 * S_2(02 * v_0^1 \oplus \delta) \oplus \gamma), \tag{17}$$

which can then be rewritten as:

$$v_0^3 = S_3(02 * S_2(02 * S_1(p_0 \oplus k_0^0) \oplus \delta) \oplus \gamma). \tag{18}$$

Here, γ depends on 12 plaintext bytes, and δ depends on three plaintext bytes. In order to keep the values of γ and δ constant, we need to keep 15 plaintext bytes constant. *We then have to guess the entire set (k_0^0, δ, γ) in order to find the key byte k_0^0, giving us a data complexity of 24 bits for attacking one key byte.* This attack has been diagrammatically represented in Figure 2. Keeping only a single byte variable gives us 256 plaintexts. Since in practice, a first-order DPA can break an AES S-box implementation with 30 to 100 traces [13], this attack is consequently a feasible venture that can be undertaken in some scenarios.

4.4 On the Attack Feasibility After the S-box at Round 4

Here, we consider attacking a byte immediately after the S-box in round 4 (S_4). Let this be the first byte v_0^4. Similar to Eq. (10), w_0^3 is a byte obtained after round 3 and u_0^3 is a byte after the MixColumns of round 3. Then with $k_0^3 \in K_3$, we have:

$$\begin{aligned} v_0^4 &= S_4(w_0^3), \\ w_0^3 &= u_0^3 \oplus k_0^3. \end{aligned} \tag{19}$$

The byte u_0^3 results from MixColumns in round 3 and can be written as:

$$u_0^3 = 02 * v_0^3 \oplus 03 * v_5^3 \oplus 01 * v_{10}^3 \oplus 01 * v_{15}^3, \tag{20}$$

where $(v_0^3, v_5^3, v_{10}^3, v_{15}^3)$ are bytes resulting from the S-box operation of this same round 3. Consider $\theta = 03 * v_5^3 \oplus 01 * v_{10}^3 \oplus 01 * v_{15}^3 \oplus k_0^3$. Now, using Eq. (20) and deriving the value of v_0^3 from Eq. (18), the byte v_0^4 can then be written as:

$$v_0^4 = S_4(02 * S_3(02 * S_2(02 * S_1(p_0 \oplus k_0^0) \oplus \delta) \oplus \gamma) \oplus \theta), \tag{21}$$

Here, θ depends on $(v_5^3, v_{10}^3, v_{15}^3)$. From Eq. (18), it can be observed that each of these bytes depend on the set (δ, γ, p_i), where p_i is some plaintext byte not

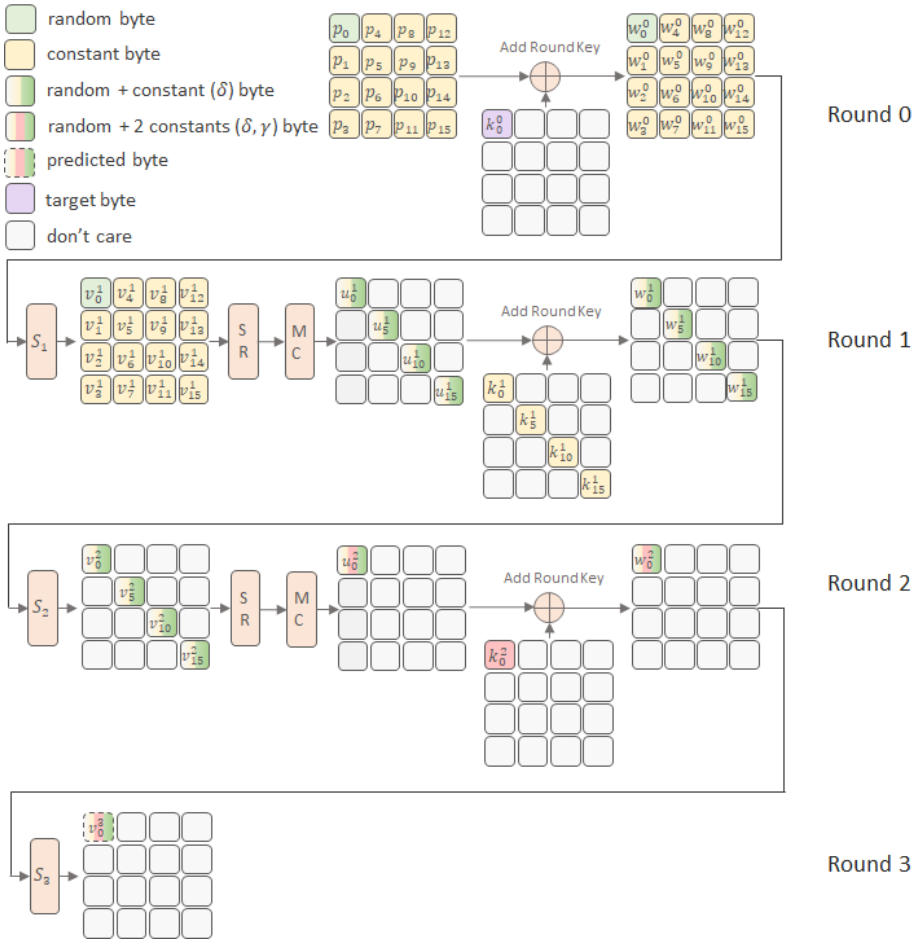

Fig. 2: Attacking k_0^0 by targeting first byte (v_0^3) after S_3.

included in either δ or γ. Combining the plaintext bytes that this set depends on, it can be concluded that $(v_5^3, v_{10}^3, v_{15}^3)$ depend on 16 bytes of plaintext each. Thus, θ effectively depends on all 16 plaintext bytes. This way, implementing an attack to recover k_0^0 by predicting v_0^4 requires fixing the 16 plaintexts for each side-channel measurement. Also, we would have to guess the variables of the set $(k_0^0, \delta, \gamma, \theta)$ in this case, that is, the attack would have to guess 32 bits in order to find one key byte. Therefore, this turns this statistical DPA attack infeasible in practice. On the other hand, a profiled attack can still vary k_0^0 (and keeping all remaining key bytes from K_0 fixed), which allows collecting profiling traces with at most 256 different intermediate values for v_0^4. Although the profiling phase allows larger variability, the attack phase is still restricted to a single plaintext-key combination.

4.5 Attacking a Byte Before AddRoundKey at Round 7

In this case, we formulate an attack on round 7 from the output in encryption mode, which would require an adaptive chosen-ciphertext attack. The process is similar to that noticed in the case of encryption. Attacking the byte u_0^7 we have:

$$u_0^7 = k_0^7 \oplus S_8^{-1}(v_0^8), \tag{22}$$

where v_0^8 is a byte from after S_8 and $k_0^7 \in K_7$. The byte v_0^8 affects 4 bytes of the resultant state after the MixColumns of round 8.

The value v_0^8 can be expressed as follows:

$$v_0^8 = 0\mathrm{e} * u_0^8 \oplus 0\mathrm{b} * u_1^8 \oplus 0\mathrm{d} * u_2^8 \oplus 09 * u_3^8, \tag{23}$$

where $(u_0^8, u_1^8, u_2^8, u_3^8)$ are bytes from the state after the MixColumns operation of round 8. These 4 bytes can then be written in terms of another 4 bytes from after S_9. That is, for $(v_0^9, v_1^9, v_2^9, v_3^9)$ being bytes after S_9 and $k_0^8, k_1^8, k_2^8, k_3^8$ being bytes of K_8, we have:

$$\begin{aligned} u_0^8 &= S_9^{-1}(v_0^9) \oplus k_0^8, \\ u_1^8 &= S_9^{-1}(v_1^9) \oplus k_1^8, \\ u_2^8 &= S_9^{-1}(v_2^9) \oplus k_2^8, \\ u_3^8 &= S_9^{-1}(v_3^9) \oplus k_3^8. \end{aligned} \tag{24}$$

Consider $0\mathrm{b} * u_1^8 \oplus 0\mathrm{d} * u_2^8 \oplus 09 * u_3^8 \oplus k_0^8 = \gamma$. Plugging the value of u_0^8 into Eq. (23), and subsequently, the value of v_0^8 into Eq. (22), we obtain:

$$u_0^7 = k_0^7 \oplus S_8^{-1}(0\mathrm{e} * S_9^{-1}(v_0^9) \oplus \gamma). \tag{25}$$

Expanding v_0^9, which affects 4 bytes after MixColumns of round 9, we get:

$$v_0^9 = 0\mathrm{e} * u_0^9 \oplus 0\mathrm{b} * u_1^9 \oplus 0\mathrm{d} * u_2^9 \oplus 09 * u_3^9, \tag{26}$$

where $u_0^9, u_1^9, u_2^9, u_3^9$ are the first 4 bytes from after the MixColumns operation of round 9. Each of these bytes go through the S-box and ShiftRows of round 10 and the last AddRoundKey before giving out ciphertext bytes. Therefore, u_i^9 can be represented as:

$$\begin{aligned} u_0^9 &= S_{10}^{-1}(c_0 \oplus k_0^{10}) \oplus k_0^9, & u_1^9 &= S_{10}^{-1}(c_{13} \oplus k_{13}^{10}) \oplus k_1^9, \\ u_2^9 &= S_{10}^{-1}(c_{10} \oplus k_{10}^{10}) \oplus k_2^9, & u_3^9 &= S_{10}^{-1}(c_7 \oplus k_7^{10}) \oplus k_3^9, \end{aligned} \tag{27}$$

where $(c_0, c_7, c_{10}, c_{13})$ are ciphertext bytes. Considering $0\mathrm{b} * u_1^9 \oplus 0\mathrm{d} * u_2^9 \oplus 09 * u_3^9 \oplus k_0^9 = \delta$, we can rewrite Eq. (25) as:

$$u_0^7 = k_0^7 \oplus S_8^{-1}(0\mathrm{e} * S_9^{-1}(0\mathrm{e} * S_{10}^{-1}(c_0 \oplus k_0^{10}) \oplus \delta) \oplus \gamma). \tag{28}$$

The term δ depends on the bytes u_1^9, u_2^9, u_3^9, which in turn depend on one ciphertext byte each, as seen above. γ depends on (u_1^8, u_2^8, u_3^8) which in turn depend on

(v_1^9, v_2^9, v_3^9) that are similar to v_0^9. We can observe from Eq. (26) that v_0^9 would be affected by four ciphertext bytes, which would actually be the case with v_1^9, v_2^9, and v_3^9 as well. We can thus conclude that γ would depend on 12 ciphertext bytes.

A statistical attack on the S-box in this case, such as DPA, would therefore include an attack on 32 bits of the set $(k_0^7, k_0^{10}, \delta, \gamma)$ and requiring 15 ciphertext bytes to be constant. An improvement can be achieved here by performing a bitwise attack such as a single-bit DPA as indicated in [13]. Here, k_0^7, being XORed, would not affect the magnitude of the difference but would only affect the sign. Performing a single-bit DPA attack and taking the absolute of the difference would therefore cancel out the influence of k_0^7. A similar observation can be made for CPA attacks as well. This would bring the attack complexity down to 24 bits as then we would have to attack only $(k_0^{10}, \delta, \gamma)$.

4.6 Generalization of the Attack on the Inner Rounds

We can use the individual attacks on the inner rounds in the previous section to derive a generalized view of the attack complexity and requirements while attacking any intermediate byte.

Generalizing for Attacks from Input in the Encryption Mode Taking into consideration Eq. (18) and Eq. (21), we can generalize any byte i after S-box in round j, v_i^j into the form:

$$v_i^j = S_j(m_1 * S_{j-1}(m_2 * \ ... \ * S_2(m_{j-1} * S_1(p_n \oplus k_n^0) \oplus \theta_1) \oplus \theta_2) ... \oplus \theta_{j-1}), \quad (29)$$

where p_n is a plaintext byte that directly affects v_i^j and $k_n^0 \in K_0$ is the initial key byte that is XORed with it. Every θ_j requires $3 \times 4^{j-1}$ bytes of plaintext to be constant. We then have to attack the set $(k_i^0, \theta_1, ..., \theta_{i-1})$, which results in attacking or guessing $8i$ bits in order to obtain 1 key byte, k_i^0. Table 1 gives the number of constant plaintext bytes required for the largest θ in an equation, which is θ_{j-1} for round j and the number of bits to attack for finding 1 byte of the key. This table also provide the maximum amount of plaintext-key options for the attack on round i. Note that when attacking, e.g., round 4 and 5, an attacker would have to target a single plaintext-key combination, transforming the attack process into a simple power analysis (SPA).

Generalizing for Attacks from Output in the Encryption Mode Similar to the attack from input, we can generalize the attack from output from the one given in Eq. (28). Attacking a byte before the AddRoundKey of $(10 - j)$th round, v_i^{10-j}, we have:

$$v_i^{(10-j)} = k_m^{(10-j)} \oplus S^{-1}_{(10-j)+1}(m'_1 * S^{-1}_{(10-j)+2}(m'_2 * \ ... \\ * S_9^{-1}(m'_{j-1} * S_{10}^{-1}(c_n \oplus k_n^{10}) \oplus \theta_1) \oplus \theta_2) ... \oplus \theta_{j-1}), \quad (30)$$

Round i	No. of fixed plaintext bytes (effectively) to attack k_0^0 and to predict $S_i(w_0^{i-1})$	No. of bits to be guessed in order to attack k_0^0 and to predict $S_i(w_0^{i-1})$	Plaintext × Key options to attack k_0^0 and to predict $S_i(w_0^{i-1})$ (p=Profile, a=Attack)
1	0	8	p=$2^{128} \times 2^8$, a=$2^{128} \times 1$
2	3	16	p=$2^{104} \times 2^8$, a=$2^{104} \times 1$
3	15	24	p=$2^8 \times 2^8$, a=$2^8 \times 1$
4	48 (16)	32	p=1×2^8, a=1×1
5	192 (16)	40	p=1×2^8, a=1×1

Table 1: Number of constant plaintext bytes required and number of bits to attack for 1 key byte for first 5 rounds of AES-128. This table also shows the maximum amount of different combinations of plaintext and key for profile and attack phases in each target round.

where $k_m^{(10-j)} \in K_{(10-j)}$. c_n is a ciphertext byte and $k_n^{10} \in K_{10}$. Every θ_j requires $3 \times 4^{j-1}$ constant ciphertext bytes. As mentioned in Section 4.5, if we carry out the attack in a bitwise manner, the attack set would consist of $(k_0, \theta_1, \theta_2, ..., \theta_{j-1})$ giving us $8i$ bits to attack in order to obtain the key byte k_0. Key generation being invertible, we can work our way upwards to obtain the original key bytes K_0 from K_{10}.

5 Experimental Results

Next, we discuss our experimental results where we provide results for attacks from input only. We plan to address attacks from the output in encryption mode in future works. Section 5.1 describes the setup we use to acquire the power traces, while Section 5.2 gives an insight into the deep learning architecture that we use to perform the attacks. Sections 5.3 and 5.4 exhibit the attacks where we compare the performance of deep learning against CPA on the acquired traces, both before and after introducing countermeasures such as (Gaussian) noise and misalignment. More specifically, we observe the effect of Gaussian noise for the attack on round 2 while using both misalignment and Gaussian noise for round 3 traces.

5.1 Setup

We use a general setup for capturing the power traces for all of our experiments. The traces contain power measurements collected from a Pinata development board [3] based on a 32-bit STM32F4 microcontroller with an ARM-based architecture, running at the clock frequency of 168 MHz. We acquired power traces from a standard unprotected AES-128 look-up table implementation running on

[3] Pinata Board: `https://www.riscure.com/product/pinata-training-target/`

the target device. The setup consisted of a Riscure current probe[4], a Lecroy Waverunner 610Zi oscilloscope, and a computer to communicate with the equipment and store the acquired traces. The power traces were measured at a sampling frequency of 1GS/sec and consisted of 220 000 samples. We perform power acquisitions specifically for rounds 2 and 3 and use the chosen plaintext strategy for the attacks as was discussed in Section 4 and Table 1.

For round 2, we need four acquisitions to attack all the key bytes since it is possible to attack 4 bytes at once. We collect 10 000 traces per acquisition, with 20% of the traces having a fixed key which is also the target key. We use Gaussian noise as a test against countermeasure while attacking both rounds 2 and 3. The mean and the standard deviation of the original traces dataset have been used to generate the Gaussian noise that is added to each trace. That is, the new traces with the noise were computed as follows,

$$X^* = X + \mathcal{N}(\mu_x, \sigma_x^2), \tag{31}$$

where $\mathcal{N}(\mu_x, \sigma_x^2)$ is the Gaussian distribution formed using the mean μ_x and the variance σ_x^2 of the original traces X itself. For round 3, we have to perform 16 acquisitions for attacking all key bytes since only one key byte can be attacked at a time. We collect 3 000 traces per acquisition for round 3, with all the traces having the fixed target key. The traces collected were misaligned during the time of acquisition, and we use this misalignment for an additional countermeasure test in this case. That is, we first align the traces and perform the attacks, followed by attacking the original dataset to compare the results in the presence of misalignment. We employ a standard pattern-based approach to do the alignment.

5.2 The Deep Learning Model Architecture

CNN architectures analogous to VGG [20] have been shown to give good results in the field of DL-SCA [1,10] and is, therefore, one of the widely adopted models. We particularly use the benchmarked model architecture CNN_{best}, which has been proven to outperform other models such as VGG-16 and MLP_{best} as shown by Benadjila et al. [1].

The architecture CNN_{best} contains five convolutional blocks, to begin with, where each block is made up of 1 convolutional layer and one average pooling layer. Each convolutional layer has filters for each block as (64, 128, 256, 512, 512), the kernel size as 11 (effectively indicating *same* padding), and uses ReLU as the activation function. The convolutional blocks are followed by two fully connected layers, each containing 4 096 units. Finally, the output layer uses Softmax and gives the probabilities for all the classes, which in our case would be the probabilities for each of the 9 Hamming Weight classes. The model uses categorical cross-entropy as the loss function, which is the most prominent of the loss function used in such case scenarios as has been mentioned in Section 2.2.

[4] Current probe: `https://www.riscure.com/product/current-probe`

For hyperparameter tuning, CNN_{best} works with the RMSprop backpropagation optimizer, a learning rate of 10^{-5}, and trains for 75 or 100 epochs for a batch size of 200. While we do not change the optimizer and the learning rate, Benadjila et al. [1] also showed CNN_{best} has an equally good performance with 50 epochs as well. We observed that while 50 epochs give better results for round 3, 100 epochs worked better while attacking a byte at round 2. Further, we also noticed better performance in the attack phase (w.r.t. the number of traces taken to guess the correct key byte) when using a smaller batch size, which is then fixed to be 64 in our experiments. Accordingly, the input layer then has the shape of $(2\,960 \times 64)$ where 2 960 is the number of PoIs (or features) selected. Table 2 shows the benchmarked values used for CNN_{best} and the values that we consider for this work.

We also test randomized CNN architectures with up to 4 convolutional layers each having the kernel size ranging from 10 to 20 and a stride of either 5 or 10, followed by 3 dense layers each having up to 1 000 neurons and a layer weight initializer randomly picked from (`random_uniform`, `glorot_uniform`, `he_uniform`). The activation function for all layers was randomly selected from (`relu`, `selu`, `elu`, and `tanh`). We observed that most of these random architectures also showed good results in breaking the inner rounds.

Hyperparameters	Benchmarked Choice	Our Setup
Training Hyperparameters		
Epochs	up to 100	50 (R3)/100(R2)
Batch size	200	64
Architecture Hyperparameters		
Blocks	5	5
CONV layers	1	1
Filters	64	64
Kernel size	11	11
FC layers	2	2
ACT function	ReLU	ReLU
Pooling layer	Average	Average
Padding	With zeros	With zeros

Table 2: Summary of the benchmarked values of the hyperparameters and the values used in our work.

5.3 Attacking a Byte After Round 2 S-box

To attack a byte after the S-box of round 2, each target byte needs three plaintext bytes to be fixed in the target dataset, thereby allowing us to target four key bytes with each acquisition of power traces. For example, in order to target key bytes (0, 4, 8, 12), we need to have the other 12 plaintext bytes fixed. Therefore, trace set acquisition is made accordingly, where these 4 bytes of the plaintext

are randomly defined, and the others remain fixed. An attack to find all the 16 key bytes would therefore require four such acquisitions in total.

We chose to attack the 0^{th} key byte for showcasing our results. Using Eq. (29), we compute the hypothesis for attacking key byte 0 as follows,

$$hyp = HW[S(02 * S(p_0 \oplus k_0) \oplus \delta)], \tag{32}$$

where $\delta = 03 * S(p_5 \oplus k_5) \oplus 01 * S(p_{10} \oplus k_{10}) \oplus 01 * S(p_{15} \oplus k_{15})$. As can be seen here, we need to keep the plaintext bytes (5, 10, 15) fixed in order to make the attack possible, and the hypothesis *hyp* itself depends on only p_0 and k_0 of the input trace. For DL-SCA, we label the traces during the profiling phase using the hypothesis and then guess the bytes (k_0, δ) during the attack phase. We set the hyperparameters as discussed in Section 5.2. Training and validation are done for 7 500 and 500 traces, respectively, and on variable keys that do not consist of the target key bytes while having the constant plaintext bytes as `0x00` for simplicity. The attack is performed on a set of 2 000 traces with a fixed key. In the case of DL-SCA, we observe that the attack yields the key after 238 traces, as shown in Figure 3 when the rank becomes 0. We generalize the term to *rank* here since we are guessing another byte apart from the key byte itself, and therefore, it is of the order 10^4 denoting roughly the 65 536 possibilities while guessing 16 bits (2^{16} possibilities). We can then deduce that the attack takes 238 traces to start recognizing the correct trend from profiling, thereby leading to correct guesses thereafter, which we can see from the drop of the rank to 0.

We then launch CPA on a set of 2 000 traces with a fixed key derived from the same dataset used above. We first compute the hypothesis for all the 2^{16} guesses and as given in Eq. (32). The correlation is then computed for all the guesses per trace, and the guess with the highest value is chosen to be the most likely guess as in any CPA attack. This experiment is then repeated 100 times for each batch of shuffled traces, and the highest correlation value is then averaged out, resulting in an average rank for each batch. The results of this attack are shown in Figure 3. The average rank achieved by CPA is six after 2 000 traces. As we notice a decreasing trend in the average ranks, we believe that CPA would eventually find the key if given more traces during the attack.

We now add noise to the power traces as described in Section 5.1 and observe the performance of both scenarios again. With added Gaussian noise, DL-SCA still finds the key after 139 traces as seen in Figure 4, while CPA does not find the key even after 2 000 traces despite the downward trend that we see in Figure 4. The average rank given by CPA, in this case, is 352 after 2 000 traces while it attempts to guess 16 bits of information.

5.4 Attacking a Byte After Round 3 S-box

Round 3 requires the attacker to get a separate trace set acquisition process per target key byte. In this work, we specifically target the first key byte k_0. We then compute the hypothesis as follows,

$$hyp = HW[S(02 * S(02 * S(p_0 \oplus k_0) \oplus \delta) \oplus \gamma)], \tag{33}$$

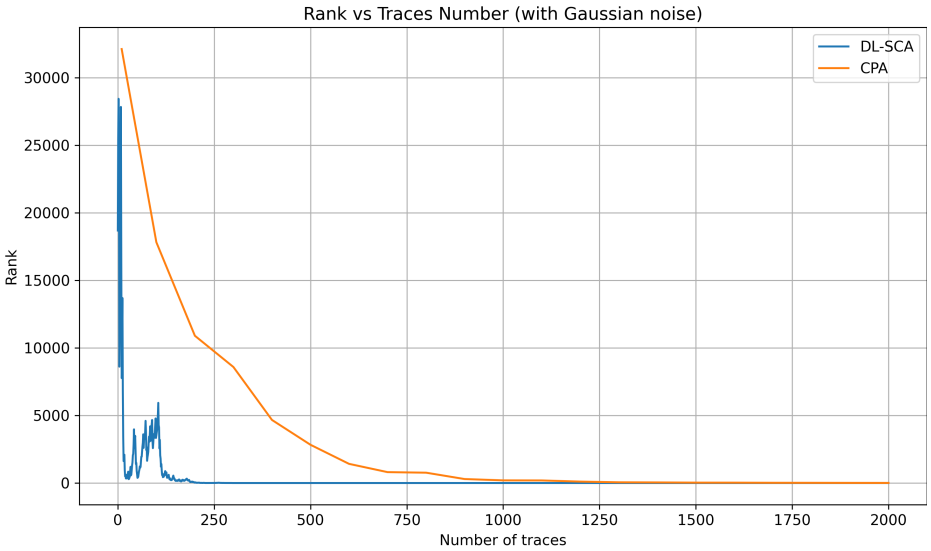

Fig. 3: DL-SCA and CPA for key byte 0 after S-box on encryption round 2.

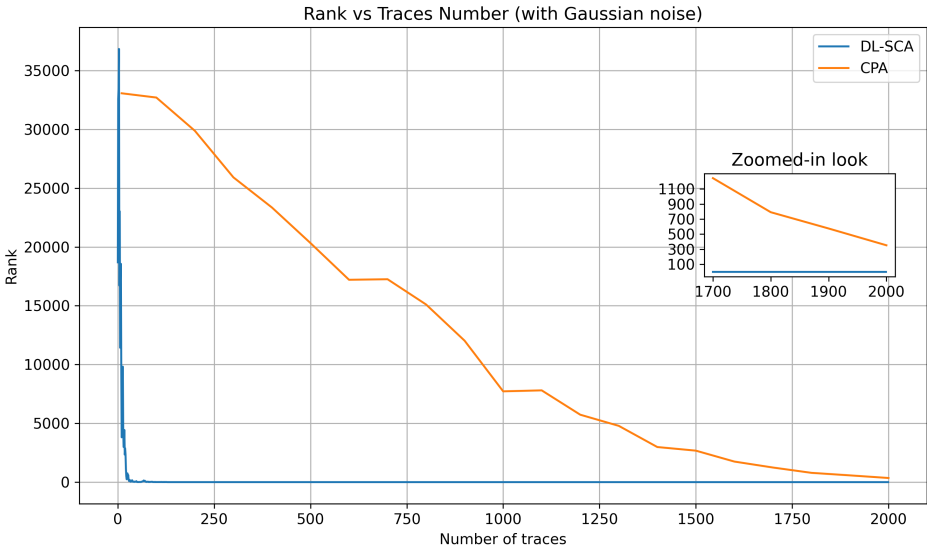

Fig. 4: DL-SCA and CPA after adding Gaussian Noise for key byte 0 after S-box on encryption round 2.

where hyp is the 8-bit hypothesis computed for one input trace while p_0 and k_0 are the first bytes of plaintext and key for that input trace, respectively. Since this depends on p_0, we gather the acquisition set with the first byte as variable and the rest of the bytes as constant, which we set as `0x00` for simplicity. As

discussed in Section 5.1, we first perform the attacks on aligned traces followed by attacks on the misaligned ones. For DL-SCA on the aligned set of traces, since we have only 3 000 traces collected per acquisition in our dataset, we use the first 2 000 traces for the profiling phase, the following 500 for validation and attack the next 500 traces. The model used is as described in Section 5.2. As done for round 2, the label for each trace is computed using Eq. (33) for profiling, where (δ, γ) can be set to any constant including `0x00`. During the attack we attempt to guess 3 bytes (k_0, δ, γ). On performing the attack in this case, we successfully attain the key byte k_0 along with the correct values of δ and γ after 11 traces. The result is shown in Figure 5 (here too, we generalize the term to *rank* since we are guessing 3 bytes in total). Similar to the result seen for round 2, the rank is of the order 10^6, indicating the 2^{24} possible guesses (16 million possibilities) for 24 bits of data. The attack takes just 11 traces to start recognizing the trend and guessing the correct key.

For CPA, we compute the hypothesis and subsequently the correlation for all the 2^{24} guesses, similar to what was done for round 2. The result of this attack is then shown in Figure 5. The correct key converges towards the highest correlation value as expected from a successful CPA attack, and the correct key is obtained after 50 traces and again at 110 traces. Here, we restrict the computation of key ranks to only 1 experiment instead of 100 as done in the case of round 2. Therefore, the results for CPA on round 3 are given as a proof-of-concept for the attack. This is because of the CPU-intensive operations done while brute-forcing 24 bits on a standard personal computer. The experiments were done using Intel Core i9 8-core processor and 16GB RAM. Computation of hypothesis for 500 traces takes approximately 27 minutes, followed by an average of 9 minutes for computing the key rank for each batch of traces. With an increment of 10 traces per batch, completing 1 experiment for all the batches ranging from 10 to 500 traces (50 batches) takes approximately 7.35 hours. Multi-processing can be used to speed the experiments, but storing of 2^{24} possibilities for each trace is memory intensive, thereby making the use of multiple processes more expensive (in terms of speed-memory trade-off) for a standard personal computer.

We now use the misaligned traces to compare the performance of DL-SCA and CPA in the presence of such an (implicit) countermeasure. We use the same DL model along with the hyperparameters and the samples of the traces to perform DL-SCA on the misaligned traces. The attack reveals the key after ten traces. We realize intuitively that a CPA attack will be difficult to perform on misaligned traces. This is indeed proven by the results as well, which can be seen from its erratic nature. The results for DL-SCA and CPA on misaligned traces are shown in Figure 6.

We further compare the performance of DL-SCA with CPA by adding Gaussian noise to the misaligned traces. The results can be seen in Figure 7. While DL-SCA finds the key after 34 traces, CPA is unable to do so even after going through our entire attack set of 500 traces.

While DL-SCA successfully finds the key in all the above cases, CPA is successful only in the case when the traces are aligned. The effectiveness of

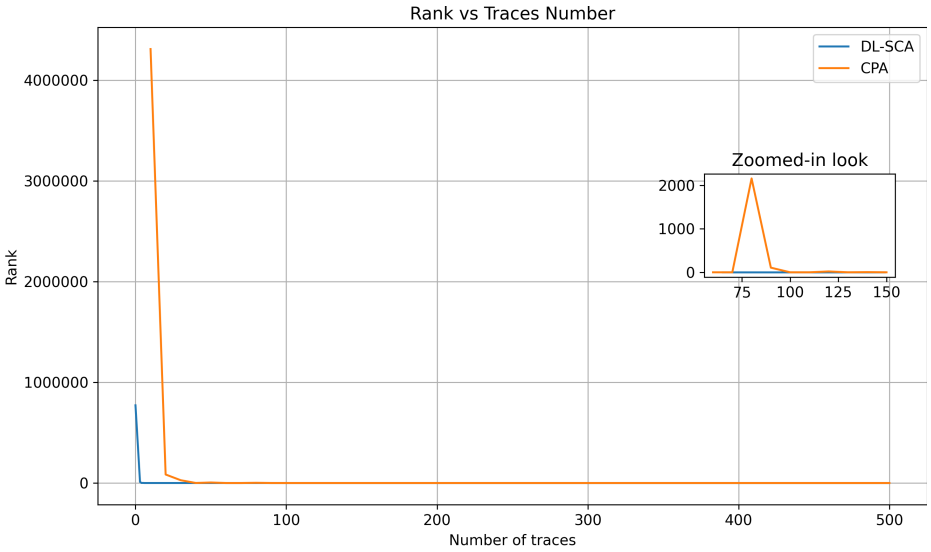

Fig. 5: DL-SCA and CPA on aligned traces for key byte 0 after S-box on encryption round 3.

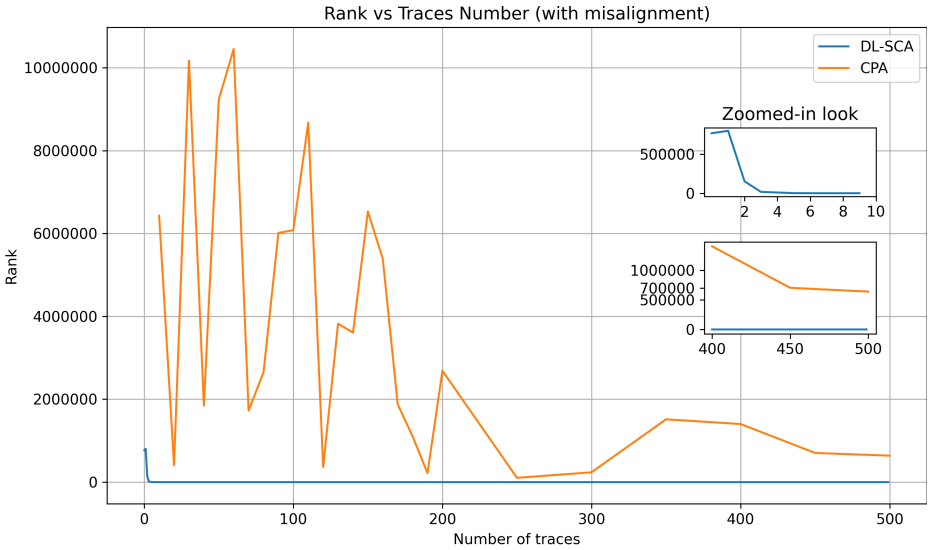

Fig. 6: DL-SCA and CPA on misaligned traces for key byte 0 after S-box on encryption round 3.

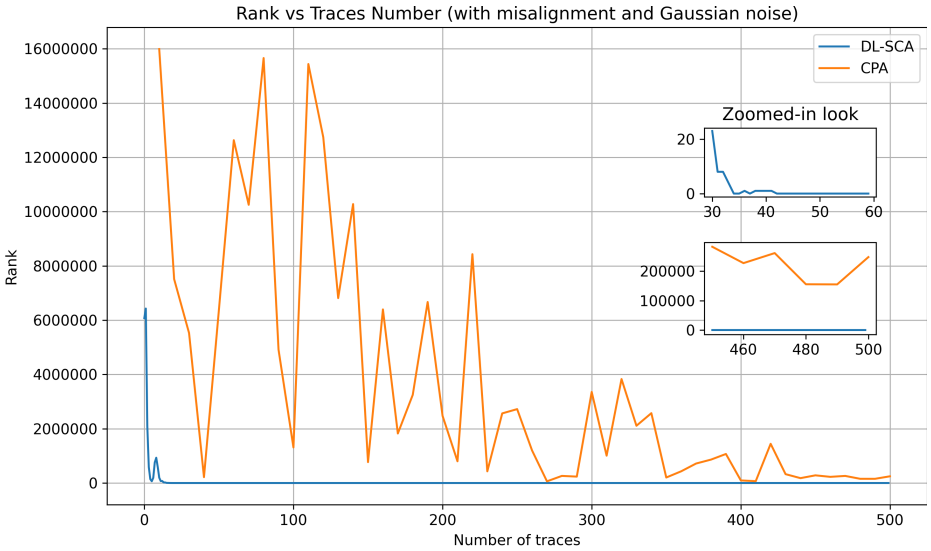

Fig. 7: DL-SCA and CPA on misaligned traces after adding Gaussian noise for key byte 0 after S-box on encryption round 3.

DL-SCA is further proven when attacking misaligned traces where it succeeds with as few as ten traces, a case where CPA was unsuccessful. *We can therefore conclude that DL-SCA certainly outperforms CPA by a tremendous margin when attacking the inner rounds.*

5.5 Attacking a Byte After Round 4 S-box

To attack the byte after the round 4 S-box, we need to guess 32 bits comprising the set of $(k_0, \delta, \gamma, \theta)$, as can also be seen from Eq. (21) and Table 1. Although attacking 32 bits is still feasible, the usage of the aforementioned three constants implies that all the 16 bytes of plaintext and the key need to be fixed for this particular attack to work. However, profiling using the same plaintexts and the same key would result in the same labels and consequently would result in the overfitting of the model.

Another case scenario would involve profiling using different plaintext but a constant key. This would mean calculating the exact values of δ, γ, and θ, which in turn leads to a properly trained model. However, the assumption in the attack phase while computing the four target bytes is that these 4 bytes are constant during the profiling as well and, by extension, should ideally have different Hamming Weights as labels than what was computed. As an example, two plaintexts having the same first byte should have the same label and, therefore, similar traces. However, since we are using different plaintexts for each trace during profiling, the training factor that the constants bring in is totally eliminated. This effectively means that the training phase and the attacking phase are car-

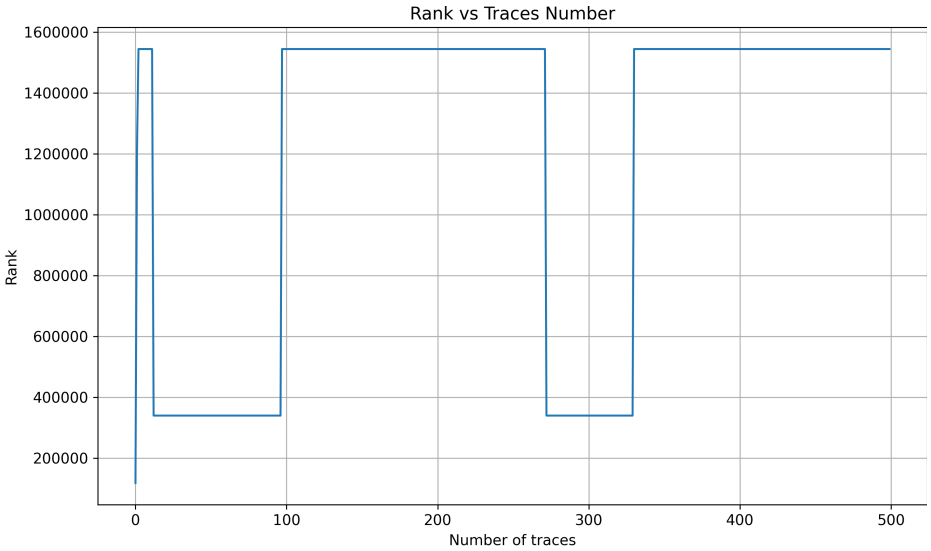

Fig. 8: DL-SCA on round 4 S-box with different plaintexts being used while training and constant plaintext used for attack. A fixed key was used both for profiling and for attack.

ried out on data that are completely different from each other, thereby rendering the attack unsuccessful. The results for the same are shown in Figure 8, and it can be observed that the rank never converges to a correct guess and does not show a decreasing trend either. A similar result was also seen while using the same plaintext but different keys. This is because the values of δ, γ, and θ not only depend on the plaintext but also on the keys and the subsequent round keys. *As of now, we conclude that an attack on any byte after the round 4 S-box is infeasible within the boundaries considered by our work.*

6 Conclusions and Future Work

Profiled and non-profiled side-channel attacks on inner AES rounds face several limitations. In this work, we proposed general formulations to attack any intermediate byte in AES encryption mode. Results indicated that attacks on rounds 2 and 3 are practical besides the increased complexity in the hypothesis guessing (16 and 24 bits, respectively). We demonstrated in practice that because profiled attacks are less restricted from fixed plaintext limitations in the profiling phase, DL-SCA can easily succeed in recovering the key in scenarios without or with (noise and misalignment) countermeasures. On the other hand, non-profiled attacks, such as CPA, becomes highly constrained by time and memory limitations as a consequence of increased complexity to guess intermediates from inner rounds. As mentioned by several related works, for several targets, DL-SCA shows easier key recovery in comparison to non-profiled attacks

if the profiling phase is done appropriately. Therefore, as shown in this paper, DL-SCA becomes a strong candidate to attack (not properly protected) inner rounds from AES.

Besides that, we also observed that results presented in Section 5 have certain limitations. Most notably, the presented approach fails at attacking further than round 3. Therefore, the most interesting open question is whether it is possible to attack rounds between 4 and 6. We believe that this goal should be achievable using deep learning. The first, more straightforward approach would be to attack both S-box input and output using multi-label deep learning [14]. We envision that in this approach, attacking the Hamming Weight of both intermediates would be the most efficient. By targeting these two intermediate states at once, the attack would be able to recover the key in a similar way to [23,2,19]. Note that this method can be applied without requiring access to input and output for AES [5].

The second approach would be to attack a combination of S-box input and output. For example, we believe that it might be sufficient to use an XOR of S-box input and output as a label. The traces might not be directly leaking that XOR value, but we envision that the neural network should be able to combine S-box input and output leakages and classify the XORed value correctly, in a similar way to which neural networks were shown to combine leakages in masked AES traces [12]. We leave further investigating of the aforementioned ideas, as well as practical experiments for attacks in decryption mode, as future works.

References

1. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ascad database. ANSSI, France & CEA, LETI, MINATEC Campus, France. Online verfügbar unter https://eprint. iacr. org/2018/053. pdf, zuletzt geprüft am **22**, 2018 (2018)
2. Bouder, H.L., Lashermes, R., Linge, Y., Thomas, G., Zie, J.: A multi-round side channel attack on AES using belief propagation. In: Cuppens, F., Wang, L., Cuppens-Boulahia, N., Tawbi, N., García-Alfaro, J. (eds.) Foundations and Practice of Security - 9th International Symposium, FPS 2016, Québec City, QC, Canada, October 24-25, 2016, Revised Selected Papers. Lecture Notes in Computer Science, vol. 10128, pp. 199–213. Springer (2016). https://doi.org/10.1007/978-3-319-51966-1_13, `https://doi.org/10.1007/978-3-319-51966-1_13`
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004). https://doi.org/10.1007/978-3-540-28632-5_2, `https://doi.org/10.1007/978-3-540-28632-5_2`

[5] Observe that similar results might be achievable using template attacks, but our choice is deep learning as it has been shown to outperform template attacks multiple times.

4. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 45–68. Springer (2017)
5. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: CHES. LNCS, vol. 2523, pp. 13–28. Springer (August 2002), San Francisco Bay (Redwood City), USA
6. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of aes. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 106–111. IEEE (2015)
7. Herbst, C., Oswald, E., Mangard, S.: An aes smart card implementation resistant to power analysis attacks. In: International conference on applied cryptography and network security. pp. 239–252. Springer (2006)
8. Heuser, A., Zohner, M.: Intelligent machine homicide. In: International Workshop on Constructive Side-Channel Analysis and Secure Design. pp. 249–264. Springer (2012)
9. Hospodar, G., Gierlichs, B., De Mulder, E., Verbauwhede, I., Vandewalle, J.: Machine learning in side-channel analysis: a first study. Journal of Cryptographic Engineering **1**(4), 293 (2011)
10. Kim, J., Picek, S., Heuser, A., Bhasin, S., Hanjalic, A.: Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 148–179 (2019)
11. Kocher, P.C., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011). https://doi.org/10.1007/s13389-011-0006-y, https://doi.org/10.1007/s13389-011-0006-y
12. Lerman, L., Bontempi, G., Markowitch, O.: A machine learning approach against a masked aes. Journal of Cryptographic Engineering **5**(2), 123–139 (2015)
13. Lu, J., Pan, J., den Hartog, J.: Principles on the security of AES against first and second-order differential power analysis. In: Zhou, J., Yung, M. (eds.) Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6123, pp. 168–185 (2010). https://doi.org/10.1007/978-3-642-13708-2_11, https://doi.org/10.1007/978-3-642-13708-2_11
14. Maghrebi, H.: Deep learning based side-channel attack: a new profiling methodology based on multi-label classification. IACR Cryptol. ePrint Arch. **2020**, 436 (2020), https://eprint.iacr.org/2020/436
15. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: International Conference on Security, Privacy, and Applied Cryptography Engineering. pp. 3–26. Springer (2016)
16. Perin, G., Chmielewski, L., Picek, S.: Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(4), 337–364 (Aug 2020). https://doi.org/10.13154/tches.v2020.i4.337-364, https://tches.iacr.org/index.php/TCHES/article/view/8686
17. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 209–237 (Nov 2018). https://doi.org/10.13154/tches.v2019.i1.209-237, https://tches.iacr.org/index.php/TCHES/article/view/7339
18. Rijsdijk, J., Wu, L., Perin, G., Picek, S.: Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. IACR Trans-

actions on Cryptographic Hardware and Embedded Systems **2021**(3), 677–707 (Jul 2021). https://doi.org/10.46586/tches.v2021.i3.677-707, https://tches.iacr.org/index.php/TCHES/article/view/8989

19. Saha, S., Bag, A., Roy, D.B., Patranabis, S., Mukhopadhyay, D.: Fault template attacks on block ciphers exploiting fault propagation. In: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings. Lecture Notes in Computer Science, vol. 12105. Springer (2020). https://doi.org/10.1007/978-3-030-45721-1_22
20. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
21. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009. pp. 443–461. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
22. Tillich, S., Herbst, C., Mangard, S.: Protecting aes software implementations on 32-bit processors against power analysis. In: International Conference on Applied Cryptography and Network Security. pp. 141–157. Springer (2007)
23. Veyrat-Charvillon, N., Gérard, B., Standaert, F.X.: Soft analytical side-channel attacks. In: Sarkar, P., Iwata, T. (eds.) Advances in Cryptology – ASIACRYPT 2014. pp. 282–296. Springer Berlin Heidelberg, Berlin, Heidelberg (2014)
24. Wouters, L., Arribas, V., Gierlichs, B., Preneel, B.: Revisiting a methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(3), 147–168 (Jun 2020). https://doi.org/10.13154/tches.v2020.i3.147-168, https://tches.iacr.org/index.php/TCHES/article/view/8586
25. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1), 1–36 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.1-36, https://tches.iacr.org/index.php/TCHES/article/view/8391