Side-channel Attacks on Inner Rounds of AES and PRESENT

Sudharshan Kumar S.

A deeper look into the inner rounds of SPN based block ciphers and how this vision can help us attack the intermediate bytes using Deep Learning.

TU Delft

Side-channel Attacks on Inner Rounds of AES and PRESENT

by

Sudharshan Kumar S.

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Tuesday August 31, 2021 at 03:30 PM.

Student number:	5148340	
Thesis committee:		
Chair:	Prof. dr. ir. R. L. Inald Lagendijk,	EEMCS, TU Delft
Supervisor:	Dr. Stjepan Picek,	EEMCS, TU Delft
Committee Member:	Dr. Julián Urbano,	EEMCS, TU Delft
Daily Supervisors:	Dr. Guilherme Perin,	EEMCS, TU Delft
	Łukasz Chmielewski,	Radboud University

TU Delft

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 PRESENT, and experimentally validates the same for AES with non-profiled and profiled attacks. We formulate the computation of the hypothesis values of any byte in the intermediate rounds of both AES and PRESENT. The more inner the 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.

Preface

Before you lies my Master thesis 'Side-channel Attacks on Inner Rounds of AES and PRESENT' wherein we attempt to attack the inner rounds of these block ciphers using Deep Learning methods to extract the key. This thesis has been brought to you by a good number of hours on Google Scholar, a significant consumption of coffee and/or Red bull, a lot of Zoom meetings (a side-cheer to my internet connection), Coursera, Reddit and Youtube breaks, and finally a pinch of blessing from the almighty beings.

Firstly, I would sincerely like to thank my supervisors, Stjepan, Guilherme and Lukasz for their immense support and guidance throughout the course of my thesis. Their patience and flexibility meant a lot, especially when all of us were struggling with the 'new normal'.

Being an international student away from home also takes a toll on you. As was for me and so many others, this up-the-hill journey was made easier and so much more tolerable when shared with friends, both here and back home. A special shoutout to Shipra and Aditya who stood by me along the way. Lastly, but definitely not the least, an epitome of unconditional support, my parents, to whom my thanks does not even begin to express the immense amount of gratitude I have for them, for they are the reason I am what I am today, an M.Sc. Graduate.

Sudharshan Kumar S.
Delft, August 2021

Contents

1 Introduction

Block ciphers constitute a building block for almost every cryptosystem being used today. From TLS connections to smart cards, various standards such as FIPS [32] have been set as guidelines for using these cryptosystems in such a way that provides optimum security as per the application. It is therefore imperative to make sure that these ciphers are implemented in the most secure way possible so that none of the sensitive information it protects falls into the wrong hands. Through history we have seen vulnerabilities that arise not due to weakness of the cryptosystem itself, rather because of an insecure implementation. Such a weak implementation causes the system to leak information that an attacker can guess/deduce through proper observation and analysis. For example, attacks such as BEAST [40] realises a vulnerability in the way in which SSL-3.0/TLS-1.0 implements the AES-CBC cipher that lets the attacker reveal(guess) encrypted content, while the construction of the cipher itself is not the problem.

Attacks where unintended leakage caused by improper implementation is exploited to gather sensitive information are called Side-channel Attacks. Side-channel attacks have especially proven effective in terms of security of embedded systems which use ciphers such as AES or even lightweight ciphers such as PRESENT. AES is a symmetric cipher based on Substitution-Permutation Network (SPN) that has been standardised by NIST [31] in 2001 and adopted as an ISO [14] international standard in 2005. It has gone through rigorous tests and has proven to be one of the most secure ciphers and is thus widely used, from TLS/SSL in the TCP/IP network stack to secure storage systems such as Hardware Security Modules (HSM). PRESENT [3] is also a symmetric cipher, also based on SPN, but designed specifically as a lightweight cipher for embedded devices and devices that need to work in resource constrained environments such as RFID tags and sensor networks. Fulfilling the requirement of a lightweight cipher, PRESENT computes over less amount of data with a 64 bit block size and a key size of 80/128 bits, has a smaller algorithm size and the capability to run purely on hardware making them fast and efficient.

As mentioned, side-channel attacks are realised through unintended information leakages that can be observed and measured by a malicious actor. The type of leakages range from power consumption [23] and electromagnetic emissions to even light emissions light-emissions. We focus on power-based leakages for the purpose of this work, which is the information that can be extracted based on the power consumed by the device while performing an encryption/decryption operation. The observation of the leakages is usually

done through localized power measurements of the processor or microcontroller. For instance, a shunt resistor is placed between the voltage supply and the device in question, and the voltage across the resistor can be measured by an oscilloscope. This voltage would then be proportional to the current drawn by the device while performing the required computation. These measurements obtained from the oscilloscope are then be converted to power traces on which further analysis takes place. Measuring the electromagnetic field around the device can also serve as a source for obtaining power traces, as the intensity of the field would also be proportional to the power consumption of the device. Tool kits such as ChipWhisperer's [12] also provide ready-made boards that can be used to record power traces from target devices.

The power traces measured are then analysed to gather more information about the input that might have been used. In the case of devices performing encryption/decryption operations using AES/PRESENT, side-channel attacks focus on extracting the key used by the device. Many methods have been introduced for the same, such as Differential Power Analysis (DPA) [23], Correlation Power Analysis (CPA) [5] and Template attacks [9]. The analysis performed by these methods is based on the differences in the power traces shown by the device corresponding to different inputs. These difference are realised with the help of leakage models, such as Hamming Distance, Hamming Weight or Identity leakage models. These leakage models help to predict the behaviour of the circuit depending on hypothesis or a guess of the input value, following the idea that a transition of each bit in an output register is correlated with the observed power traces. The methods then compute their best hypothesis/guess to reveal the most probable key used by the device.

While methods such as CPA and DPA are stochastic-based approaches and are categorized as Non-profiled attacks, methods such as Template attacks are categorized as a Profiled attack. As the name suggests, Profiled attacks assumes that the attacker has a device similar to the target device and can "profile" its behaviour thereby having a better chance to crack the target with less number of traces during attack, while Non-profiled attacks has no "training" step and is directly launched on the target. The success of these methods has attracted much research in the area focusing on breaking a targeted implementation and its consequences in the real world. Deep-learning models, such as Convolutional Neural Networks (CNN), that come under the category of Profiled attacks have also been shown to be very effective in breaking implementation of these cryptosystems [28].

AES and PRESENT, being SPN based ciphers, have multiple rounds. One can see (even intuitively) that it would be easier to attack on the first and the last rounds where the data being worked on is closer to the input/output that the attacker might already have the knowledge of. For example, the output of the S-box in the first round of AES would depend on only 1 plaintext byte and 1 key byte making the hypothesis easier to calculate. In this work, we propose to take a deeper look into side-channel attacks on AES and also consider PRESENT [3]. While a variety of methods have been introduced to attack different implementations of AES and PRESENT, the attack points/vectors have almost been the same, that is the first and last rounds of these ciphers. These approaches work using the following data: (1) the power traces corresponding to the first/last rounds of the encryption operation, (2) the hypothesis that is computed to correlate to the traces based on the S-box output of the first/last round. However, computation of the hypothesis becomes more complex in the inner rounds as more key bytes start affecting the input as is the property of SPN ciphers. There can be instances where the traces for the first round are not available

or cannot be processed for mounting an attack. We therefore, study the attack on the inner rounds of AES and PRESENT and attempt to determine the feasibility of these attacks and the increased attack complexity thereof. Our hypothesis is then formulated around generalizing such attacks and checking the possibility of using existent DL-SCA methodologies to execute the same. We would also like to compare the performance of DL-SCA with classical methods such as CPA and analyse their capabilities of handling the increased complexity of attacking the inner rounds.

The rest of the document is structured as follows: Chapter 2 gives the background required for the performed attacks and methodologies. Chapter 3 discusses the previous studies related to our work wherein we also describe our hypothesis in detail. Chapter 4 dives deep for attacking the inner rounds of AES and then extends the same for PRESENT. Chapter 5 explains the experimental results based on our hypothesis using Deep Learning models and its comparison with CPA. Chapter 6 discusses future work and concludes the report.

2

Background

An attack exploiting any unintended leakage from the device to deduce information is categorized as a side-channel attack, as already seen in Chapter 1. We can categorize them into Non-profiled side-channel attacks as CPA and DPA, and Profiled side-channel attacks such as Template attacks and Deep-learning based attacks. As this work focuses on Power-based Side-channel Attacks on AES and PRESENT, we shall first start by briefly discussing the working of these cryptosystems in Section 2.1. We will particularly be focusing on the AES-128 and PRESENT-80 variants in this work. Section 2.2 then goes through the basics of Power-based side-channel attacks, briefly discussing power consumption in circuits, and how observing this power consumption can help us retrieve information about the inputs to the device. We then explain various power-based analysis methods used for the duration of this work in Section 2.3 and Section 2.4.

2.1. Cryptosystem Basics

This section briefly explains the working of AES-128 and PRESENT-80. The notation used throughout this work is also consistent with the notation used for these cryptosystems in the following sections.

2.1.1. AES

We use AES-128[32] in this work which encrypts 128-bit plaintext blocks and has 10 rounds. The confusion layer of AES is taken care of by the Byte Substitution layer or S-box. The diffusion is done with the help of Shift Rows and the Mix-columns operations. All the mathematical operations in AES is done in $GF(2^8)$. Since this work makes use of AES-128 only in the encryption mode, we cover only the process of encryption here.

AES works on bytes, that is, the 128-bit plaintext is split into 16 bytes and put into a 4×4 array which we call the state matrix. Every element in the state matrix can be referenced by the notation $((0,0),(0,1),....,(3,3))$ where the (i,j)th element is for the ith row and jth column. Every byte goes through the following in each round.

1. **Substitute bytes** - S-box is applied to every byte of the input 4×4 array.

2. **Shift rows** - The output of the S-box is given to Shift rows where each row of the array is shifted by i bytes where i is the index of the row.

3. **Mix-columns** - The Mix-columns operation multiplies each column of the state matrix with a fixed matrix to give the resultant columns of the new state matrix.

4. **Add round key** - The resultant state is not XOR'd with the round key.

The Mix-columns operation is omitted in last round. The algorithm covering all the AES rounds is given in Algorithm 1. Here, the initial state matrix is termed as X_0, S represents S-box on the state matrix, SR represents Shift Rows and MC represents Mix-columns. K_0 is the initial key and K_i is the round key for each of the i subsequent rounds. The diagrammatic representation of the round operations as given in [33] is shown in Figure 2.1.

Algorithm 1: The AES algorithm through all the 10 rounds

Result: Ciphertext C
Plaintext P: State Matrix X_0;
$X \leftarrow X_0 \oplus K_0$;
for *rounds i: 1 to 9* **do**
  $X \leftarrow S[X]$;
  $X \leftarrow SR[X]$;
  $X \leftarrow MC[X]$;
  $X \leftarrow X \oplus K_i$;
end
$X \leftarrow S[X]$; $X \leftarrow SR[X]$; $C \leftarrow X \oplus K_{10}$;

2.1.2. PRESENT

The working of PRESENT [3] is similar to that of AES because of its SPN structure. There are primarily 2 variants introduced by the authors, the 80-bit variant and the 128-bit variant. While the former deals with 80-bit keys and the latter with 128-bit keys, both work on 64-bit data blocks. We will be discussing the 80-bit variant in this report.

The plaintext block length is 64 bits and the operations is done on chunks of 4-bits each, that is, unlike AES, PRESENT deals with nibbles instead of bytes. The confusion layer is implemented using 4-bit S-boxes and the diffusion layer is implemented with the help of a bitwise permutation operation on the input state called the pLayer. The algorithm for all the 31 rounds is given in Algorithm 2. Here too, the initial state matrix is represented by X_0, the round key for each round i by K_i, the S-box operation by S and the pLayer operation by $pLayer$. A diagrammatic representation of 2 rounds of PRESENT as given in [3] is shown in Figure 2.2.

2.2. Power Side-channels

Power-based attacks analyses the power traces recorded from the target device. To give an example how such an attack can look like in a real life setup, let us consider the following example. Let us consider a POS device performing an encryption of the pin entered by the user before sending it over the network and an attacker eavesdropping on the network and intercepting the encrypted pins. A typical encryption operation is CPU intensive, that is, the device would consume more power than normal while encrypting the pin. The attacker here turns to observing the power being consumed by the device at various time intervals

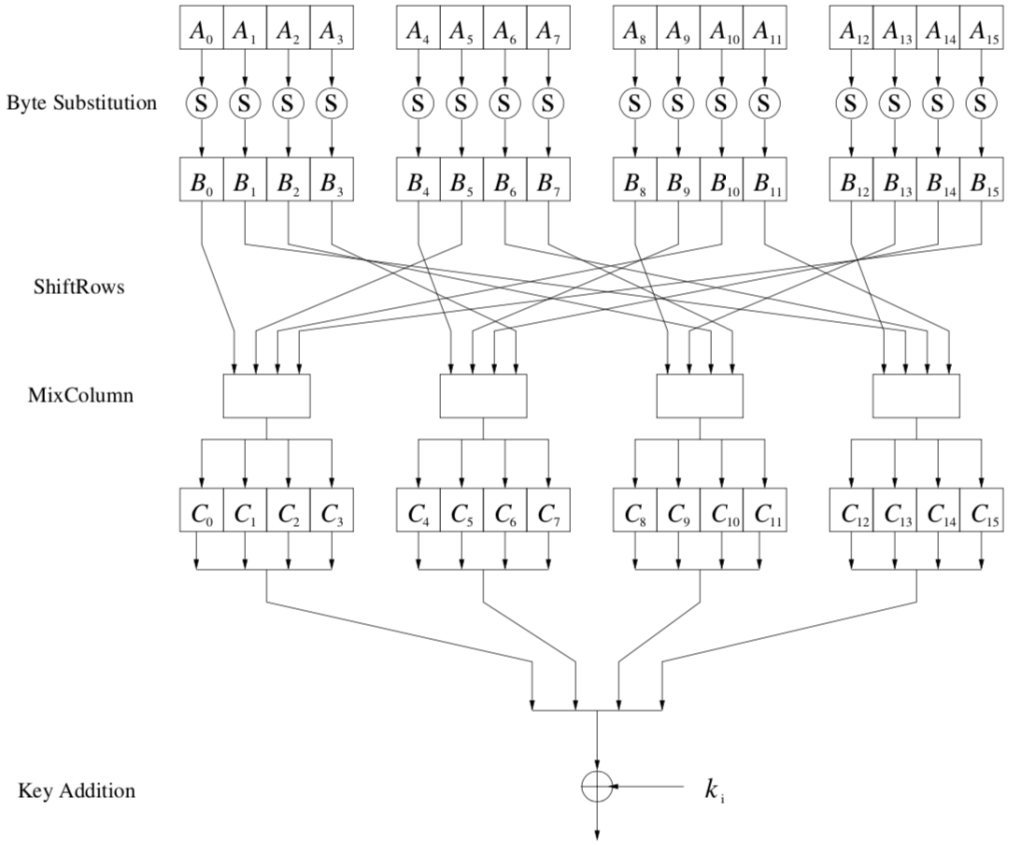

Figure 2.1: AES round structure for rounds 1-9 in AES-128 [33].

Algorithm 2: The PRESENT algorithm through all the 31 rounds

Result: Ciphertext C
GenerateRoundKeys(): $(K_0, .., K_{31})$;
Plaintext P: State Matrix X_0;
for *rounds i: 1 to 31* **do**
  $X \leftarrow X \oplus K_i$;
  $X \leftarrow S[X]$;
  $X \leftarrow pLayer[X]$;
end
$C \leftarrow X \oplus K_{32}$;

which can be done by various means such as measuring the electromagnetic waves being emitted from the device or the heat being generated. This attacker can then extract such time frames wherein the power being consumed is higher that the usual stay-alive measurements. Using any of the various aforementioned analysis methods, it is possible to deduce the key being used for encryption by the POS system from the recorded traces. Once in hold of the key, the attacker can use it to decrypt the intercepted pins and compromise the targeted users.

Depending on the computation that a CPU performs and also on the input it takes, it

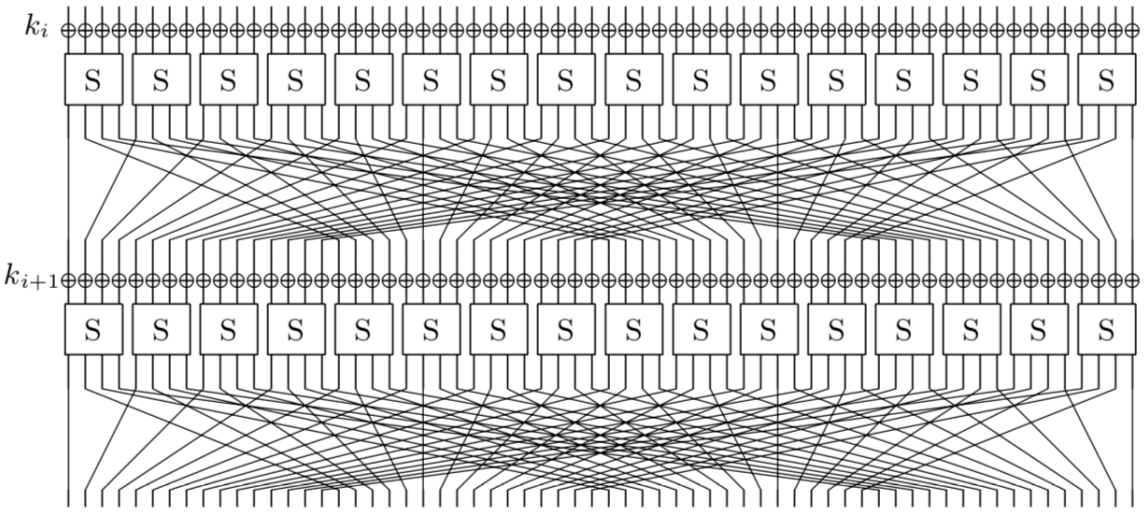

Figure 2.2: Two rounds of PRESENT [3].

consumes different amounts of power. This depends on the parts of the circuit that are being activated for a particular calculation and a particular input. On a hardware level, any computation is done using logic gates which are in turn realized using CMOS transistors. For example, a CMOS NOT gate is shown in Figure 2.3.

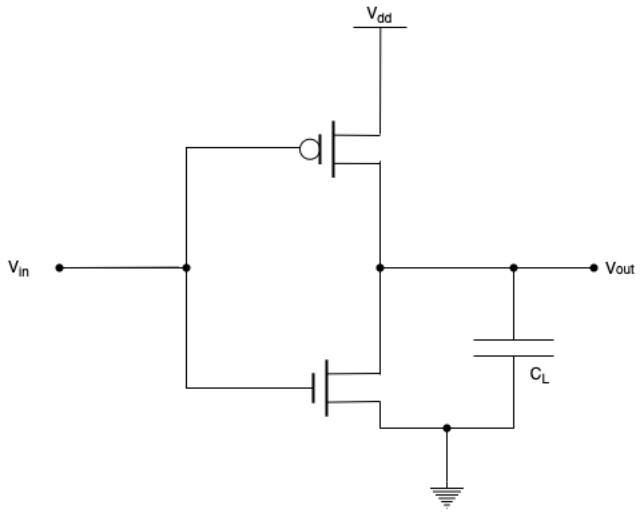

Figure 2.3: A CMOS NOT gate

Here, when the input $V_{in} = 0$, The PMOS gate is closed and output voltage V_{out} is connected to the supply voltage V_{DD}. When the input is high, the NMOS now connects the output voltage to the ground and the PMOS disconnects it from the supply voltage. In such circuits, there is always a difference in voltage levels and such a CMOS transistor will always drain voltage losing energy in the form of static power consumption.

There is also some loss of energy when switching states leading to a dynamic power consumption. When V_{in} switches from 1 to 0, the switch is not entirely perfect and there is a moment when both PMOS and NMOS form a closed circuit. In this case, a small amount

of supply voltage leaks to the ground. There is also a change in the voltage levels and a small amount of this energy is stored by the conductor itself in the form of load capacitance. When there is transition of V_{in} from 0 to 1, the PMOS opens the circuit and the NMOS short circuits. The energy stored in the form of load capacitance then drains to the ground through the short circuit. Therefore, we can observe a dynamic power consumption in any logical circuit depending on transitions that it goes through which in turn depend on the input being processed. Table 2.1 shows the power consumptions related to each type of transition for a NOT gate implemented with a CMOS transistor.

Transition of Input Signal	Leakage	Energy Consumption
$0 \rightarrow 0$	Static leakage	Low
$0 \rightarrow 1$	Static leakage + Short circuit + Charge dissipation	High
$1 \rightarrow 0$	Static leakage + Short circuit	Medium
$1 \rightarrow 1$	Static leakage	Low

Table 2.1: Charge consumption depending on input transitions in a CMOS NOT gate

Considering the leakage models, the Hamming Distance (HD) model correlates the power traces to the number of bits that were transitioned from 1 to 0 and vice-versa in a state register. The Hamming Weight (HW) model on the other hand considers only 0 to 1 transitions, thereby giving it more significance. As seen in Table 2.1, this transition has a higher energy consumption than others and therefore, the Hamming Weight model would be more effective for analysing the power traces.

2.3. Direct/Non-Profiled Side-channel Attacks

Non-Profiled techniques or Direct Attacks are directly used on the traces gathered from the target device. Any statistical methods for determining the keys would come under this category. A general structure for statistical SCA models is shown in Figure 2.4. We shall go through SPA, DPA and CPA in this section.

2.3.1. Simple Power Analysis (SPA)

SPA [22] is based on visual inspection of the power and helps to observe simple patterns in the power consumption which can give an insight into the type of computation being done by the device. In the case of symmetric ciphers, SPA does not help in directly finding the key, but is used by the attacker to gain other information about the device such as the cipher being used or the length of the key. For example, Figure 2.5 shows a raw power trace gathered from a target device. An SPA on this trace clearly shows 10 peaks / patterns indicating that the device could be computing the AES-128 cipher containing 10 rounds.

In some cases, such as those involving asymmetric ciphers such as RSA implemented with Square and Multiply algorithm, SPA can be used to find the private key for the corresponding power trace.

2.3.2. Differential Power Analysis (DPA)

DPA [23] is a more advanced form of power analysis and has the capability to find the key bytes being used through statistical methods. As mentioned previously, the main concept

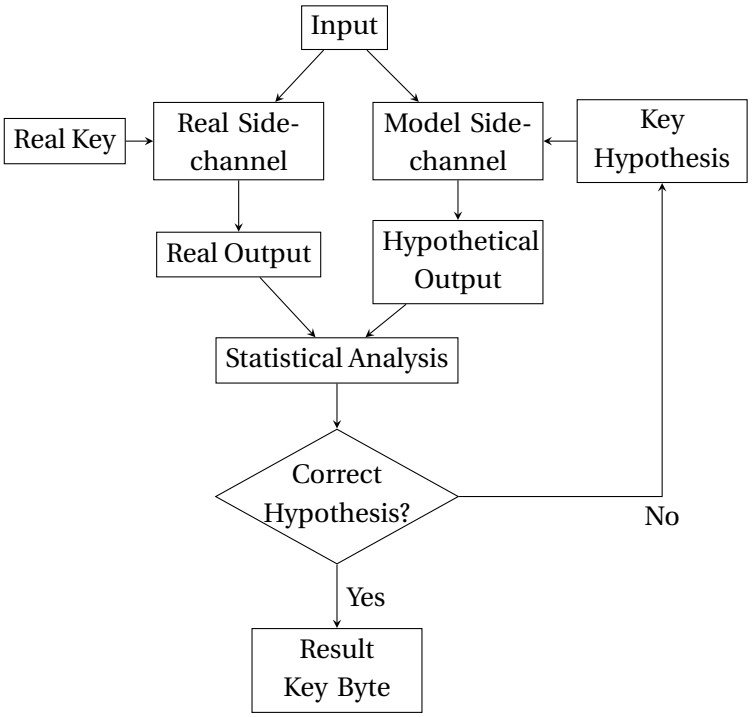

Figure 2.4: General representation of the statistical approach to SCA

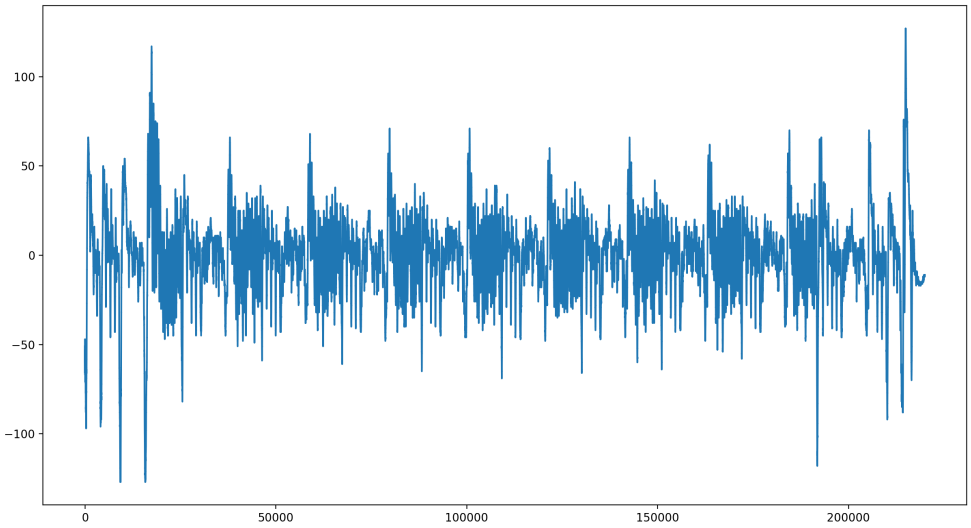

Figure 2.5: Power trace captured from one of the target devices

behind this method is that there is a correlation between the power consumption and the key byte being processed by the device. Considering one bit, say the least significant bit (LSB), of the data being processed, 2 power traces at a particular point in time will be similar if the transition occurring in the circuit is similar, that is either $0 \rightarrow 1$ indicating a higher power consumption or $1 \rightarrow 0$ which would comparatively have a lower power consumption.

The attacker then does the following steps. They perform encryption/ decryption of n

plaintexts, obtain those many power traces. For each of the power trace, a hypothesis key byte is chosen from the 256 possibilities, that is, adopting a divide and conquer approach, the guessing and hypothesis construction is done for 1 key byte at a time. A hypothesis of the leaked LSB is then obtained using a leakage model such as Hamming Weight or Hamming Distance. The average of power traces with the leaked bit being 1 is then obtained and is then subtracted against the average of power traces that have a leaked bit of 0. In general, if the ith target bit (in this case, the LSB) corresponding to the ith trace is d_i and the voltage measured for the ith encryption at time t is $p_{i,t}$, then for 1 hypothesis key byte, the difference of averages can be represented as,

$$\Delta p_t = \frac{\sum_{i=1}^{n} d_i \,.\, p_{i,t}}{\sum_{i=1}^{n} d_i} - \frac{\sum_{i=1}^{n} (1-d_i) \,.\, p_{i,t}}{\sum_{i=1}^{n} (1-d_i)}. \tag{2.1}$$

This difference Δp_t between the 2 averages is computed for all the 256 possibilities/hypotheses and for the entire time interval of the power trace. The key byte having the highest difference of averages is most probable to be the correct key byte. This follows the idea that the correct key byte would give the correct hypothesis thereby indicating the correct power transition and therefore leading to the maximum difference of averages.

2.3.3. Correlation Power Analysis (CPA)

CPA [5] is also a statistical method that is used to correlate the power traces with the observed leakage. Similar to the methodology followed in DPA, an attacker has to perform numerous encryptions/decryptions and collect the power traces. Using a leakage model, a hypothesis for each key guess can then be obtained. Unlike DPA that uses difference of averages for differentiating between the modeled and the actual power traces, CPA uses Pearson Correlation. Pearson Correlation between 2 datasets X, Y can be defined as follows:

$$\rho_{X,Y} = \frac{cov(X,Y)}{\sigma_x \sigma_y} = \frac{E[(X-\mu_X)(Y-\mu_Y)]}{\sqrt{E[(X-\mu_X)^2]E[(Y-\mu_Y)])^2]}}. \tag{2.2}$$

Pearson Correlation increases towards +1 in case X and Y 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 similar divide and conquer approach as done in the case of DPA, we take 1 key byte and its corresponding hypothesis at a time. So considering n traces, T data points where t corresponds to 1 data point in time, $p_{i,t}$ would be the voltage 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 follows:

$$r_{k,t} = \frac{\sum_{i=1}^{n} [(h_{i,k} - \bar{h_k})(p_{i,t} - \bar{p_t})]}{\sqrt{\sum_{i=1}^{n} (h_{i,k} - \bar{h_k})^2 \sum_{i=1}^{n} (p_{i,t} - \bar{p_t})^2}}. \tag{2.3}$$

Here $r_{k,t}$ would give the correlation value for one key guess and for a 1 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 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.

2.4. Profiled Side-channel Attacks

Profiled side-channel attacks take into consideration that the attacker has a copy of the target device. The attack is then split into 2 phases: the profile phase and the attack phase. In the Profile phase, the attacker profiles or trains his attack on the copy of the target device in his position. This can also be done with the help of Machine Learning and Deep Learning algorithms. The output of the profile phase is a classification model (usually based on probability distribution of different key guesses) which is then used in the attack phase on the actual target device to extract the secret key.

2.4.1. Template attacks

Template attacks introduced in [9], is based on classification of the power traces into classes defined by the leakage model. This classification is done by computing probability distribution of each of the classes using Bayes' theorem. For example, in the case of Hamming Weight, there will be 9 classes, $0-8$. In the profiling phase, a template is then built by deriving this probability distribution for the defined classes using traces from a copy of the device and known keys. During the attack phase, the traces from the target device is then compared against the template and the key guess and its corresponding hypothesis is then categorized as per the computed distributions. The probabilities for each key guess is then summed over all the traces captured during the attack phase and key guess with the highest probability is determined to be the actual key. In order to have a successful attack the template needs to have enough samples for each of the defined classes. This would also determine the effective number of traces that would be required in the profile phase to make the attack successful.

2.4.2. Deep Learning Methodologies

Deep Learning based SCA (DL-SCA) provides an improvement over Template attacks 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, runs it through the defined model, and determines 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 voltage 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 the most probable actual key. Convolutional Neural Network (CNN) and Multi-Layer Perceptron (MLP) models have proven effective for side-channel attacks [24, 47] and more efficient than non-profiling SCA [28].

Neural Network In general, a layer of the neural network consists of neurons which are defined by its weight w and bias b. The output of this neuron depends on the activation function, σ, applied to it and can be written as

$$y = \sigma(\sum_{i=1}^{n} w_i x_i + b), \tag{2.4}$$

where $x_i, i \in [1, n]$ are all the input values to the neuron and w_i corresponds to the weight assigned to each of these inputs. w_i is then adjusted for each neuron depending on the desired result (which is defined with labels) and other criteria such as loss in which cases algorithms such as gradient descent is then applied. The activation function σ is a non-linear function applied on the output of the transformation done with the weights. Rectified Linear Unit (ReLU) or Softmax are common examples of such activation functions and are also used for this work. These can be formalized as follows:

$$\begin{aligned} \sigma_{\texttt{ReLU}}(z) &= max(0, z), \\ \sigma_{\texttt{softmax}}(z)_j &= \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}, \text{ for } j = 1, ..., K \text{ and } \mathbf{z} = (z_1, ..., z_K) \in \mathbb{R}^K, \end{aligned} \tag{2.5}$$

where $\sigma_{\texttt{softmax}} : \mathbb{R}^K \rightarrow [0, 1]^K$ with K being the dimensionality of the output layer. In the case of CNN, each trace and its data points form the input layer forms a 1-dimensional input signal which is fed to a series of the aforementioned layers to classify the traces using probability of the defined classes. We use CNN with VGG-like architecture in our work as it is the most prominent model used for SCA to the best of our knowledge. The original model was developed for the purpose of image classification where the input signal has multiple input dimensions starting from 2. But as SCA has only 1 spatial dimension considering its data points in time, the main difference that VGG-like architectures introduce is the way in which 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 being applied on the input signal across time t (time steps in the case of SCA owing to the 1-dimensional property of the power traces), which as shown in [21], can be represented as,

$$(\phi * x)(t) = \sum_{a=-\infty}^{\infty} x(a)\phi(t - a), \tag{2.6}$$

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 so as 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 a output signal of that layer, that is, 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. This VGG-like CNN `cnn`, is represented in [21] as follows,

$$\texttt{cnn} = \texttt{fc}_{\theta,\texttt{softmax}} \circ \prod_{p=1}^{P} \texttt{fc}_{\theta^p,\texttt{ReLU}} \circ \prod_{q=1}^{Q} (\texttt{pool} \circ \prod_{r=1}^{R_q} \texttt{conv}_{\phi^r,\texttt{ReLU}}), \tag{2.7}$$

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 follows [21],

$$\begin{aligned} \texttt{conv}_{\phi,\sigma}(X) &= \sigma(\phi * X), \\ \texttt{fc}_{\theta,\sigma}(x) &= \sigma(\theta^T x), \end{aligned} \tag{2.8}$$

where $X \in \mathbb{R}^{i \times d}$ 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). It is also worth noticing that the definition of `fc` here is equivalent to the definition of the output neuron given in Eq. (2.4). σ is an activation function which can either be `ReLU` or `softmax` as shown in Eq. (2.5). As seen in Eq. (2.7), `ReLU` is usually used for hidden layers and `softmax` is used for the final output layer which represents the probabilities of each of the defined classes. An example of this CNN structure can also be pictorially represented as given in Figure 2.6, as adopted from [34].

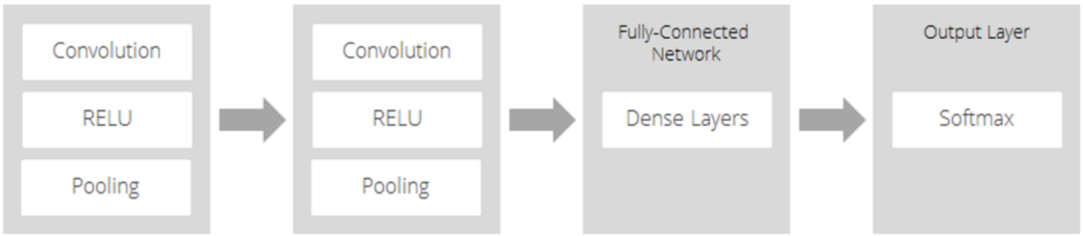

Figure 2.6: An example of CNN architecture used in DL-SCA [34].

Training the Neural Network and tuning its Hyper-parameters The training phase iteratively applies the Gradient Descent algorithm in order to minimize the loss which is defined by quantifying the classification error encountered while classifying over the training/profiling set. This loss is computed with the help of a loss function such as cross-entropy loss, a classical loss function which is generally used for classification problems[16] such as the one we are trying to solve as well. Cross-entropy loss being smooth and decomposable adjusts well while optimizing standard gradient-based methods and is therefore a default choice for many of the DL-SCA models[2]. The parameters that define the architecture and subsequently the training of the model are called Hyper-parameters. These Hyper-parameters generally consist of Batch size, Epochs, Optimizers and Learning rates. We discuss these Hyper-parameters as follows.

- **Batch size** - Batch size defines the number of inputs that are processed at once by the model during training. Mini-batch learning is usually preferred[16] where a small

batch of inputs is used at a time to learn the model. Several efficiency/accuracy factors such as parallelization using GPUs, stability of the model, etc. are used to determine the size of the mini-batch used during the training phase[16].

- **Epochs** - When the training algorithm such as a Stochastic Gradient Descent goes through the entire dataset at least once, it is called an epoch. While less number of epochs (iterations) can lead to under-fitting (the model is not able to capture the required trends in the training data), more iterations can cause over-fitting (the model is too specific to the training data and does not generalize its results). The number of epochs is therefore an important parameter to tune while training the model.

- **Optimizers and Learning rates** - Several extensions and variants of the Stochastic Gradient Descent have been proposed in the context of deep learning. There are variants of the Stochastic Gradient Descent model that have been proposed in the context of Deep Learning and these are called Optimizers. They aim to adapt the learning rate, otherwise known as the step size, of the Gradient Descent during the training phase.

These Hyper-parameters are tuned by performing ad-hoc tests wherein factors such as accuracy and efficiency of the model while training are taken into consideration. That is, a base model with its Hyper-parameters is chosen. Then the impact of each of the Hyper-parameters is studied and modified in a way such that the resultant model gives the best results during training and validation, for example, in terms of accuracy.

2.5. 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 [44]. 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

This chapter discusses the previous work that has been done in the area with respect to this thesis. Section 3.1 goes through the problem that we aim to solve by discussing the most prominent SCA methodologies and their attack surfaces as given in literature. Keeping in mind other relevant work, Section 3.2 then repostulates our hypothesis about the problem statement and how our contributions help to solve it.

3.1. SCA on Outer Rounds

In this section we first discuss the most prominent attack surface for SCA attacks for both software and hardware implementations. The focus would primarily be on AES and the same can be extended for PRESENT as well. We then go through the countermeasures recommended in the literature that would render these attack surfaces useless while using typical off-the-shelf SCA methodologies.

3.1.1. Attack surface

The variations of a known data, such as plaintext or ciphertext fed to the target device, form the basis of attacks based on statistical analysis. In order to reveal the secret, which in most cases is the key, the attacker chooses a leakage model depending on the architecture of the device/implementation in question. Hamming Weight and Hamming Distance are most commonly considered leakage models where the value of either of them would correlate to some measurable quantity such as power traces. The correlation can then be used by the attacker to perform guesses with plausible candidate keys (or key bytes). Each guessed key byte is then ranked on the basis of its confidence defined using metrics such as absolute difference (DPA) correlation (CPA), or even probability in case of Template attacks as shown in [42].

For software implementations, each operation such as SubBytes, AddRoundKey etc. is implemented in one entire clock cycle. Here, the SubBytes (S-box) operation, being the first non-linear operation encountered during encryption, becomes an obvious choice for an attack surface in terms of leakage model correlating with the power traces. For example, the hamming weight of the S-box applied on the byte guesses in the first round can be correlated with the power traces and consequently the guessed byte having the maximum correlation value turns out to be the correct key byte if the process is done right.

As for the hardware implementations, it is possible to combine multiple operations in a single clock cycle. For example, an FPGA can perform an entire cipher round (encryption and decryption) in one clock cycle [30]. This gives us 2 venues of attack for consideration. If an intermediate state (the state after every round) and the ciphertext is stored in the same register [10], then the guesses can be done for the last round wherein the model is implemented over inverse of the S-box. The difference between the ciphertext and the state just before the last round determines the power traces and consequently the correlations. This might not work in every situation since it is possible that the ciphertext and the previous-to-last state are not stored in the same register rendering the leakage model useless [11]. However, in such cases the device would be storing the intermediate states in the same register after each clock cycle/round and a leakage model using the value of this state register would provide a potential attack vector. That is, a leakage over just the S-box would not be sufficient and our model would have to include the Mix-columns computations as well in order to take into account the computation of an entire round. This can be observed in microcontrollers such as STM32F4, STM32L4, etc. The works shown in [11, 29, 30] demonstrate a single-bit CPA/DPA using the Hamming distance model over the state register carrying out the side-channel attack successfully on AES-128 and AES-256.

3.1.2. Preliminary countermeasures and their drawbacks

In the light of the above attacks, a variety of countermeasures were introduced for AES primarily based on techniques such as Masking and Hiding. Masking follows the principle of adding random masks to the intermediate values thereby affecting the correlation between the power traces (or the measurable quantity) and the data being processed [1, 8]. While Hiding involves blinding the correlations by adding randomised noise into it with techniques such as Software balancing, Power consumption randomisation, etc. [13]. Achieving these countermeasures successfully, however, has been a challenge in terms of cost. Masking through all the rounds of AES adds significant cost over memory and execution [7, 17, 45]. A similar conclusion can also be drawn for Hiding countermeasure which would need expensive circuits to be implemented in hardware. Therefore, a security/performance trade-off is required in order to implement a relatively secure system which is efficient at the same time.

As can be seen from Section 3.1.1, the first and last rounds of operation are primary targets to statistical attacks. These rounds are more vulnerable as any intermediate value from these rounds depend on a relatively small fraction of the key. 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 [17, 45].

In some cases of hardware implementation, it is also possible that multiple rounds are executed within 1 clock cycle. This would result in the inner rounds being exposed, that is, 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 [23], 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 [26]. 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.

3.1.3. Deep Learning

Many approaches have been developed in the field of SCA, from statistical methods such as CPA/DPA to template attacks, ML-SCA and DL-SCA. While the former ones have been studied extensively, attacks based on profiling involving machine learning and deep learning are still developing. Already studies have shown that machine learning could be used to mount successful side-channel attacks that are also more effective than template attacks [18, 19]. Machine learning methods such as SVM has also been used to counter masked implementations as shown by Lerman et al. [25]. Extending on the same, Gilmore et al. show that neural nets can also be used to tackle the masking countermeasure and are more effective and efficient at it than the Machine learning-based approaches [15]. This emphasizes on the power of neural networks to correctly identify the masking applied to the rounds of AES and effectively removing their effect, thereby allowing the attacker to successfully proceed with an approach that would work even for unmasked implementations. However, these implementations depend on one crucial assumption that the random masks are available to the attacker during the profile phase, which as mentioned by [15] is an impractical assumption. As mentioned before, most of the practical and efficient countermeasure implementations involve only the outer rounds [17, 45]. Therefore, we can bypass these countermeasures if we attack the inner rounds directly which would also not require us to be in the possession of the random masks being used by the target implementation.

We can extend the neural network implementations to profile the intermediate bytes in the inner rounds thereby executing a DL-SCA on the inner rounds. Deep Learning (more precisely, Convolutional Neural Networks (CNN) and Multi Layer Perceptrons (MLP)) has indeed been successfully used to attack AES implementations as first shown by Maghrebi et al. [28]. 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 [6]. Kim et al. discussed the VGG-like architecture that showed good attack performance for several datasets, where some were using masking or hiding countermeasures [20]. 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 [2]. Picek et al. showed that metrics commonly indicating the performance of machine learning algorithms are not appropriate to assess the SCA performance [36]. 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 [49]. Wouters et al. further discussed the methodology perspective, providing even smaller neural network architectures that perform well [48]. 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 [35]. Rijsdijk et al. introduced the reinforcement learning approach for designing neural networks that perform well and are as small as possible [39]. 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.

3.2. SCA on Inner Rounds: Our Hypothesis and Contributions

We would like to determine to what extent can we attack the inner rounds of AES and PRESENT and the factors affecting such attacks, and also the most efficient attack methodology for the inner rounds. Our initial hypothesis here was based on just the feasibility of attacking the inner rounds. As this has already been shown by Jiqiang Lu et al. [26], we are now interested to know if it is possible to generalize the attacks on inner rounds in order to understand the data complexity and the techniques that would be needed to attack an intermediate byte at any round. We would then like to check if a similar attack and its generalisation can be done for the inner rounds of PRESENT as well. We note here, to the best of our knowledge, that an attack on the inner rounds of PRESENT has not been performed yet. The generalisation 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. As mentioned in Section 3.1.3, Deep Learning provides us with an interesting prospective for a successful attack and we would like to showcase these Deep Learning-based approaches would be an improvement over methodologies using DPA/CPA. To this end, our two-fold hypothesis can be summarised as follows,

1. Is it possible to perform side-channel attacks on the inner rounds of AES and PRESENT?
   (a) If possible, how can the hypothesis be computed for these inner rounds?
   (b) What would be the factors determining the feasibility of such an attack?
   (c) Would it be possible to generalise such an attack and its determining factors for any of the inner rounds if these ciphers?

2. If the above attack is possible, can Deep Learning methods be used to attack these inner rounds and extract the key? How effective would these attacks be as compared to the classical approaches such as CPA.

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. We therefore attempt to answer the above questions through the following 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 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. A similar generalization is done for PRESENT-80 as well, in the encryption mode. Similar to what was observed in the case of AES, targeting any intermediate byte after n *S-boxes* requires an attack complexity of $4 \times (n+1)$ in this case.

3. We experimentally validate the generalization for AES with non-profiled and profiled attacks. To make our analysis more realistic, we introduce potential countermeasures (such as Gaussian noise and misalignment) to power traces collected from an unprotected AES.

4. 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.

5. 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.

6. As we specify in this work, 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.

We shall see that it is possible to generalize the computation of any intermediate byte in AES-128 and PRESENT-80 and that the complexity of the attack increases with the depth of the round when attacking from input during encryption. We shall also see that apart from being more effective, Deep Learning also handles the increasing data complexity of the attacks in a much better way that the classical CPA approach.

4

Attacking the Inner Rounds

In this chapter, we dig into the method for finding the value of a byte in any of the intermediate rounds of AES-128 and PRESENT-80. We shall first explore the process and the methodology of doing so in the context of AES-128 in Section 4.1, building upon which we formulate the attack on the inner rounds of PRESENT-80 in Section 4.2.

4.1. Attacking the Inner Rounds of AES-128

Lu et al. [26] 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 [26] 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. [26] 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. [26]. 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 4.1.6. 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.

4.1.1. 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.

4.1.2. 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:

$$\begin{aligned} v_0^2 &= S_2(w_0^1), \\ w_0^1 &= u_0^1 \oplus k_0^1. \end{aligned} \tag{4.1}$$

The process is illustrated in Figure 4.1. 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:

$$u_0^1 = 02 * v_0^1 \oplus 03 * v_5^1 \oplus 01 * v_{10}^1 \oplus 01 * v_{15}^1, \tag{4.2}$$

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. (4.1) from Eq. (4.2), we have:

$$v_0^2 = S_2(02 * v_0^1 \oplus 03 * v_5^1 \oplus 01 * v_{10}^1 \oplus 01 * v_{15}^1 \oplus k_0^1). \tag{4.3}$$

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 4.1) across all side-channel measurements. By inserting γ in Eq. (4.3) we obtain:

$$v_0^2 = S_2(02 * v_0^1 \oplus \gamma). \tag{4.4}$$

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. (4.4) can be rewritten as:

$$v_0^2 = S_2(02 * S_1(p_0 \oplus k_0^0) \oplus \gamma). \tag{4.5}$$

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 4.1. 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.

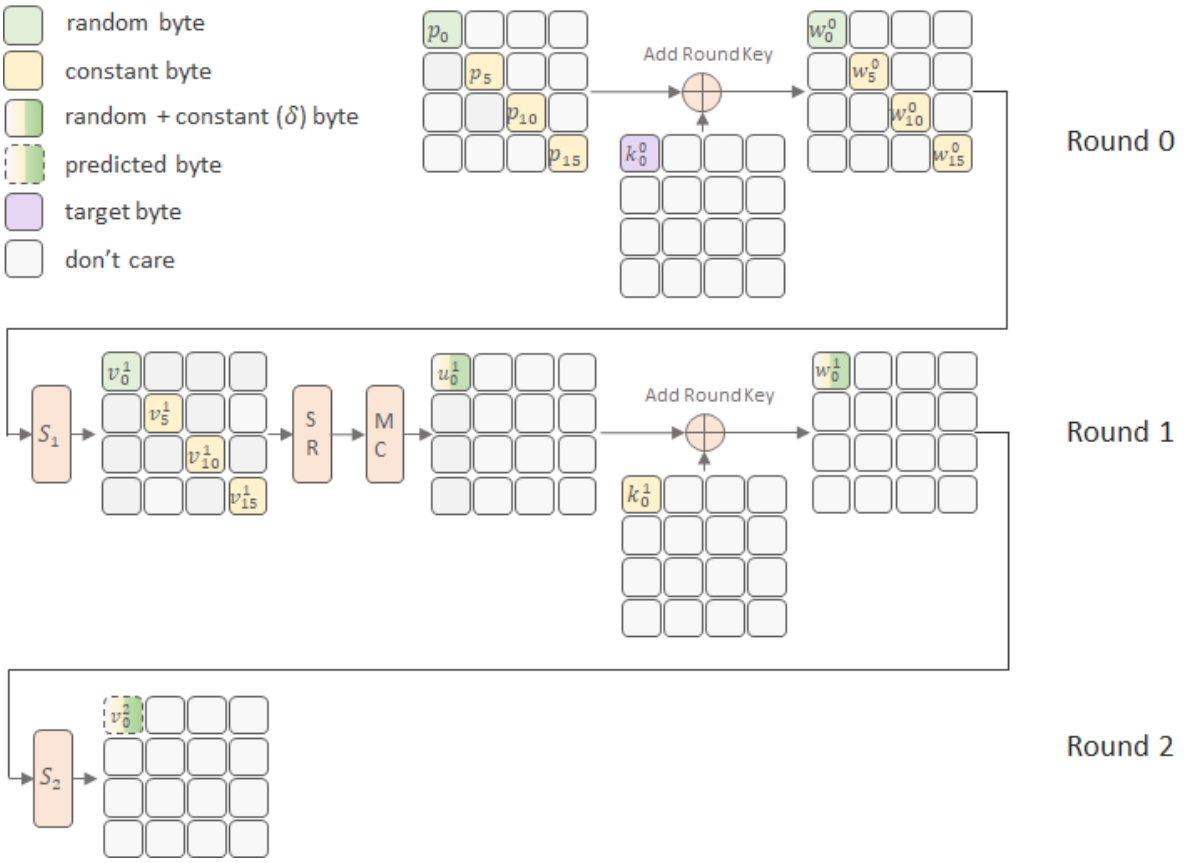

Figure 4.1: Attacking k_0^0 by targeting first byte (v_0^2) after S_2.

4.1.3. 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:

$$v_0^3 = S_3(w_0^2) = S_3(u_0^2 \oplus k_0^2). \tag{4.6}$$

We follow the approach from Section 4.1.2, 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:

$$u_0^2 = 02 * v_0^2 \oplus 03 * v_5^2 \oplus 01 * v_{10}^2 \oplus 01 * v_{15}^2. \tag{4.7}$$

Substituting the value of u_0^2 in Eq. (4.6) from Eq. (4.7), we obtain:

$$v_0^3 = S_3(02 * v_0^2 \oplus 03 * v_5^2 \oplus 01 * v_{10}^2 \oplus 01 * v_{15}^2 \oplus k_0^2). \tag{4.8}$$

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. (4.8):

$$v_0^3 = S_3(02 * v_0^2 \oplus \gamma). \tag{4.9}$$

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

$$\begin{aligned} v_0^2 &= S_2(u_0^1 \oplus k_1^1), \qquad v_5^2 = S_2(u_5^1 \oplus k_5^1), \\ v_{10}^2 &= S_2(u_{10}^1 \oplus k_{10}^1), \qquad v_{15}^2 = S_2(u_{15}^1 \oplus k_{15}^1). \end{aligned} \tag{4.10}$$

Expanding on u_0^1, we observe an equation similar to Eq. (4.7), 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:

$$u_0^1 = 02 * v_0^1 \oplus 03 * v_5^1 \oplus 01 * v_{10}^1 \oplus 01 * v_{15}^1. \tag{4.11}$$

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:

$$\begin{aligned} v_0^1 &= S_1(p_0 \oplus k_0^0), \qquad v_5^1 = S_1(p_5 \oplus k_5^0), \\ v_{10}^1 &= S_1(p_{10} \oplus k_{10}^0), \qquad v_{15}^1 = S_1(p_{15} \oplus k_{15}^0). \end{aligned} \tag{4.12}$$

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. (4.9), we obtain:

$$v_0^3 = S_3(02 * S_2(u_0 \oplus k_0^1) \oplus \gamma) \implies v = S_3(02 * S_2(02 * v_0^1 \oplus \delta) \oplus \gamma), \tag{4.13}$$

which can then be rewritten as:

$$v_0^3 = S_3(02 * S_2(02 * S_1(p_0 \oplus k_0^0) \oplus \delta) \oplus \gamma). \tag{4.14}$$

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 4.2. 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 [26], this attack is consequently a feasible venture that can be undertaken in some scenarios.

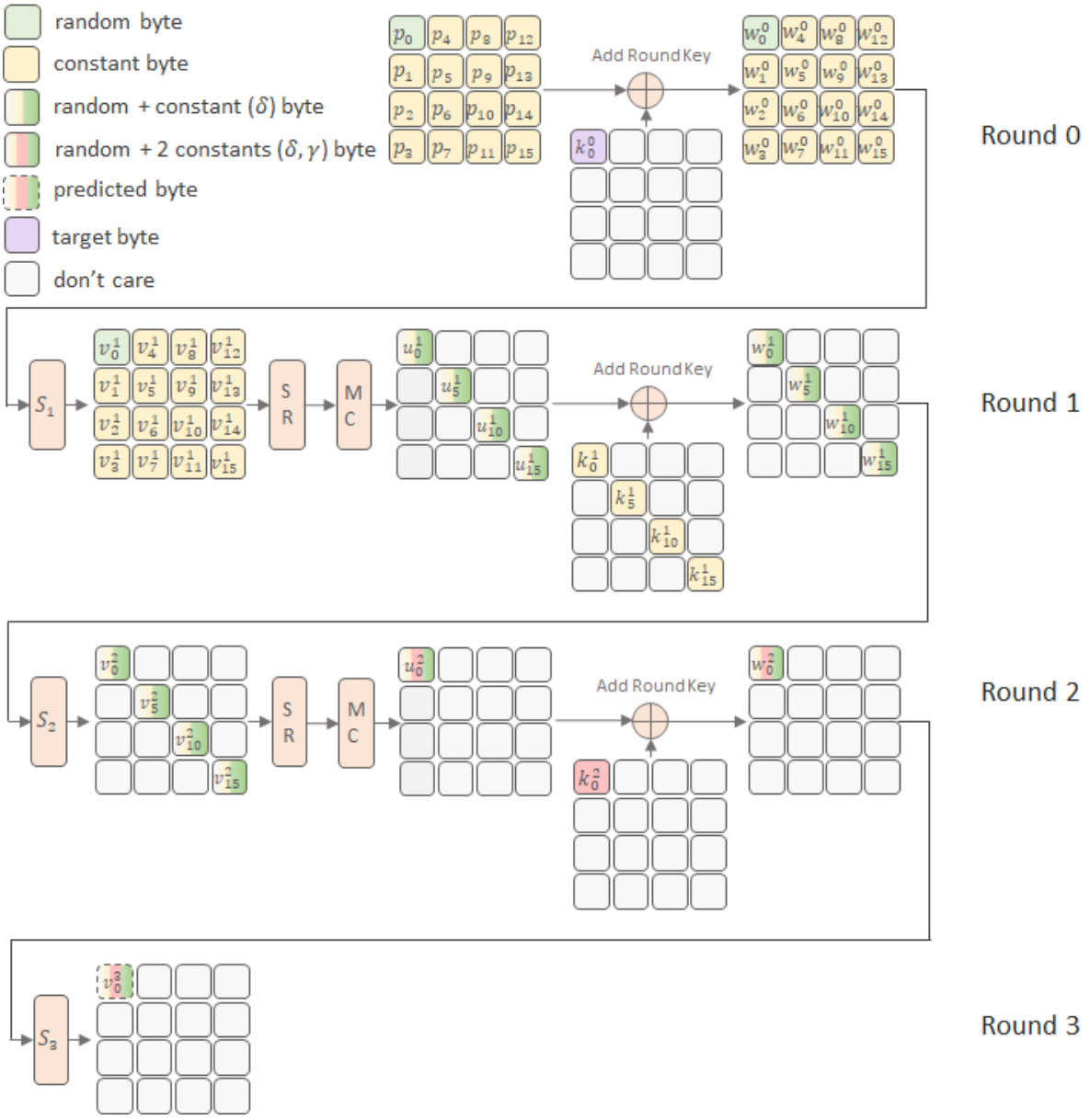

Figure 4.2: Attacking k_0^0 by targeting first byte (v_0^3) after S_3.

4.1.4. 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. (4.6), 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:

$$\begin{aligned} v_0^4 &= S_4(w_0^3), \\ w_0^3 &= u_0^3 \oplus k_0^3. \end{aligned} \tag{4.15}$$

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

$$u_0^3 = 02 * v_0^3 \oplus 03 * v_5^3 \oplus 01 * v_{10}^3 \oplus 01 * v_{15}^3, \tag{4.16}$$

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. (4.16) and deriving the value

of v_0^3 from Eq. (4.14), the byte v_0^4 can then be written as:

$$v_0^4 = S_4(02 * S_3(02 * S_2(02 * S_1(p_0 \oplus k_0^0) \oplus \delta) \oplus \gamma) \oplus \theta), \tag{4.17}$$

Here, θ depends on $(v_5^3, v_{10}^3, v_{15}^3)$. From Eq. (4.14), 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 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.

4.1.5. 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:

$$u_0^7 = k_0^7 \oplus S_8^{-1}(v_0^8), \tag{4.18}$$

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:

$$v_0^8 = 0e * u_0^8 \oplus 0b * u_1^8 \oplus 0d * u_2^8 \oplus 09 * u_3^8, \tag{4.19}$$

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:

$$\begin{aligned} u_0^8 &= S_9^{-1}(v_0^9) \oplus k_0^8, \\ u_1^8 &= S_9^{-1}(v_1^9) \oplus k_1^8, \\ u_2^8 &= S_9^{-1}(v_2^9) \oplus k_2^8, \\ u_3^8 &= S_9^{-1}(v_3^9) \oplus k_3^8. \end{aligned} \tag{4.20}$$

Consider $0b * u_1^8 \oplus 0d * u_2^8 \oplus 09 * u_3^8 \oplus k_0^8 = \gamma$. Plugging the value of u_0^8 into Eq. (4.19), and subsequently, the value of v_0^8 into Eq. (4.18), we obtain:

$$u_0^7 = k_0^7 \oplus S_8^{-1}(0e * S_9^{-1}(v_0^9) \oplus \gamma). \tag{4.21}$$

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

$$v_0^9 = 0e * u_0^9 \oplus 0b * u_1^9 \oplus 0d * u_2^9 \oplus 09 * u_3^9, \tag{4.22}$$

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:

$$\begin{aligned} u_0^9 &= S_{10}^{-1}(c_0 \oplus k_0^{10}) \oplus k_0^9, & u_1^9 &= S_{10}^{-1}(c_{13} \oplus k_{13}^{10}) \oplus k_1^9, \\ u_2^9 &= S_{10}^{-1}(c_{10} \oplus k_{10}^{10}) \oplus k_2^9, & u_3^9 &= S_{10}^{-1}(c_7 \oplus k_7^{10}) \oplus k_3^9, \end{aligned} \tag{4.23}$$

where $(c_0, c_7, c_{10}, c_{13})$ are ciphertext bytes. Considering $0b * u_1^9 \oplus 0d * u_2^9 \oplus 09 * u_3^9 \oplus k_0^9 = \delta$, we can rewrite Eq. (4.21) as:

$$u_0^7 = k_0^7 \oplus S_8^{-1}(0e * S_9^{-1}(0e * S_{10}^{-1}(c_0 \oplus k_0^{10}) \oplus \delta) \oplus \gamma). \tag{4.24}$$

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. (4.22) 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 [26]. 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 $(k_0^{10}, \delta, \gamma)$.

4.1.6. 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. (4.14) and Eq. (4.17), we can generalize any byte i after S-box in round j, v_i^j into the form:

$$v_i^j = S_j(m_1 * S_{j-1}(m_2 * \ \ldots\ * S_2(m_{j-1} * S_1(p_n \oplus k_n^0) \oplus \theta_1) \oplus \theta_2) \ldots \oplus \theta_{j-1}), \tag{4.25}$$

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 $(k_i^0, \theta_1, \ldots, \theta_{j-1})$, which results in attacking or guessing $8j$ bits in order to obtain 1 key byte, k_i^0. Table 4.1 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).

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 4.1: 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.

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. (4.24). Attacking a byte before the AddRoundKey of $(10-j)$th round, v_i^{10-j}, we have:

$$v_i^{(10-j)} = k_m^{(10-j)} \oplus S^{-1}_{(10-j)+1}(m'_1 * S^{-1}_{(10-j)+2}(m'_2 * \ ... \\ * S_9^{-1}(m'_{j-1} * S_{10}^{-1}(c_n \oplus k_n^{10}) \oplus \theta_1) \oplus \theta_2) ... \oplus \theta_{j-1}), \tag{4.26}$$

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 4.1.5, 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}.

4.2. Attacking the Inner Rounds of PRESENT-80

As PRESENT works on 4-bit S-boxes, we would target nibbles in the attack. The key chunk to attack would also be represented as nibbles instead of bytes. Being designed for resource constrained environments, PRESENT is usually implemented as *encryption-only* for applications that demand the most efficient use of space. As it is possible to compute the encryption sub-keys *on-the-fly*, implementing PRESENT as encryption-only will result in an ultra-lightweight solution [3]. Due to this preferred design choice by the authors of [3] for implementing PRESENT as encryption-only, we focus on attacking only the encryption operation of PRESENT.

We first introduce the notations that are used in the following sections in Section 4.2.1, followed by some preliminaries required for the attack in Section 4.2.2. We then study the attack after the computation of S-box in round 2,3 and 4 in Sections 4.2.3, 4.2.4, and 4.2.5 respectively, and finally derive the generalization of the attack for a byte in any intermediate round of PRESENT in Section 4.2.7.

4.2.1. Notations and Preliminaries

As done for AES, we first introduce the notations that would be used in this section.

- Plaintext nibbles are denoted by p_i, where i is the index of the nibble.

- A nibble after the AddRoundKey operation is denoted by w_i^n, where i is the index of the nibble and n indicates the round, while the output nibble of an S-box in any round is denoted by v_i^n. For example, v_0^1 is the first nibble obtained after the S-box in round 1. Similarly, nibbles after the pLayer operation are denoted using u_i^n which are also the input to the next round.
- The key nibbles are denoted by k_i^n and the round key they belong to is denoted by K_n. The first round key would then be $\{k_0^1, k_1^1, ..., k_{15}^1\} \in K_1$, while the last round key would be $\{k_0^{31}, k_1^{31}, ..., k_{15}^{31}\} \in K_{31}$
- S-box in round n is denoted as S_n and we denote its application on an input nibble u as $S_n(u)$.
- Terms such as γ, δ, θ are used to denote 4-bit constants.

4.2.2. Preliminaries

Before diving into the actual attack, we go through some preliminary computations that would be required during the attack on the inner rounds.

Mathematical representation of pLayer and its inverse pLayer can be computed in the form of a linear equation.

$$P(i) = \lfloor \frac{i}{4} \rfloor + (16 \times (i \ mod \ 4)), \tag{4.27}$$

where i is the index of the target bit, such that, $i \in [0, 63]$. Similarly, the inverse operation of pLayer can also be represented as a linear equation.

$$P^{-1}(i) = \lfloor \frac{i}{16} \rfloor + (4 \times (i \ mod \ 16)), \tag{4.28}$$

where i is the index of the target bit and $i \in [0, 63]$.

Computing pLayer output from candidate source nibbles In PRESENT, pLayer works in a way that it permutes and rearranges the bits from different nibbles into one nibble in the resultant layer. For example, in every round of PRESENT, the first 4 nibbles, n_0, n_1, n_2, n_3, are taken and the first bit from each of them is extracted that are then combined together to make first nibble of the following round. This is represented in a diagrammatic manner in Figure 4.3.

In practice, this is implemented as any regular bit permutation in hardware [3]. We therefore need to simulate the permutation operation so as to compute the resultant value of a particular nibble. We take the example shown in the Figure 4.3. The first bit from each nibble can be extracted by multiplying each nibble with $0x8$, that is, a logical AND between the nibble and 2^3.

$$\begin{aligned} b_0 &= n_0 \wedge 2^3, \qquad b_1 = n_1 \wedge 2^3, \\ b_2 &= n_2 \wedge 2^3, \qquad b_3 = n_3 \wedge 2^3. \end{aligned} \tag{4.29}$$

The required bits are positioned in the MSB of the nibbles b_0, b_1, b_2, b_3. We now have to shift these required bits to positions such that they then correspond to the same positions

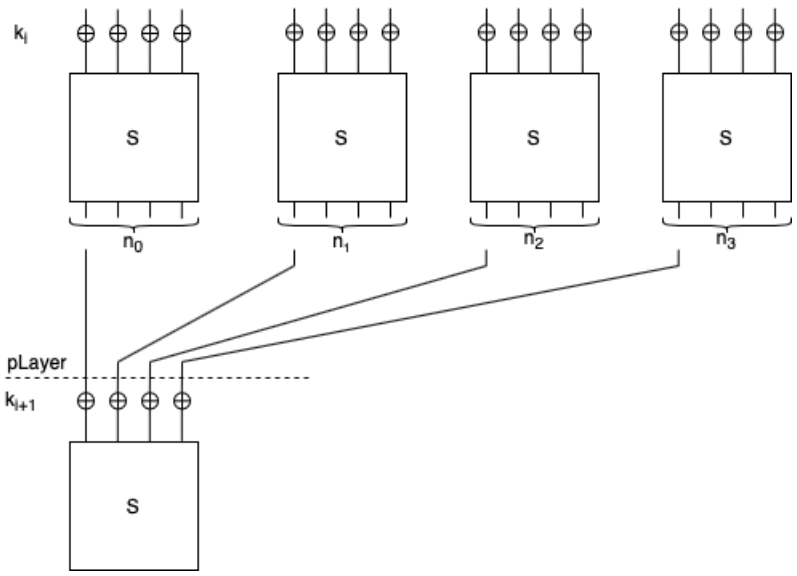

Figure 4.3: Combining the bits from 4 nibbles into 1 resultant nibble

in the target nibble as well. That is, we right shift b_0 by 0 bits, b_1 by 1 bit, b_2 by 2 bits and b_3 by 3 bits. This can be represented as,

$$\begin{aligned} b_0 &= b_0 \gg 0 = (n_0 \wedge 2^3) \gg 0, \\ b_1 &= b_1 \gg 1 = (n_1 \wedge 2^3) \gg 1, \\ b_2 &= b_2 \gg 2 = (n_2 \wedge 2^3) \gg 2, \\ b_3 &= b_3 \gg 3 = (n_3 \wedge 2^3) \gg 3. \end{aligned} \tag{4.30}$$

The resultant nibble v can then be written as,

$$\begin{aligned} v &= b_0 \oplus b_1 \oplus b_2 \oplus b_3 \\ &= ((n_0 \wedge 2^3) \gg 0) \oplus ((n_1 \wedge 2^3) \gg 1) \oplus ((n_2 \wedge 2^3) \gg 2) \oplus ((n_3 \wedge 2^3) \gg 3). \end{aligned} \tag{4.31}$$

The logical AND operation used here can be termed as multiplication operation in $GF(2)$. Depending on the index of the bit to be extracted from the source nibble, this multiplication can be done with any element in the set $(2^3, 2^2, 2^1, 2^0)$. The right shift can also be expressed as a division operation, that is, a right shift in the range $(0,1,2,3)$ would correspond to division by $(2^0, 2^1, 2^2, 2^3)$. We combine these 2 operations under the notation $\bullet$ and an appropriate operand m_i depending on the location of the nibble being attacked. In this case, if we have a nibble n, we can write the above operations as,

$$\begin{aligned} n \bullet m_0 &= (n \wedge 2^3) \gg 0, \quad & n \bullet m_1 &= (n \wedge 2^3) \gg 1, \\ n \bullet m_2 &= (n \wedge 2^3) \gg 2, \quad & n \bullet m_3 &= (n \wedge 2^3) \gg 3, \end{aligned} \tag{4.32}$$

So the Eq. (4.31) can be reformulated as,

$$v = (n_0 \bullet m_0) \oplus (n_1 \bullet m_1) \oplus (n_2 \bullet m_2) \oplus (n_3 \bullet m_3). \tag{4.33}$$

Computing the source nibbles Each nibble at the beginning of each round depends on 4 source nibbles that are input to the pLayer of the previous round. In order to compute the value of a resultant nibble after a pLayer, we first have to find the index of these source nibbles. For example, the first nibble in every round input is extracted from the bits of the first 4 nibbles after the S-box of the previous round. If j is the index of the nibble in question, then the index of the source nibbles i_0, i_1, i_2, i_3 can be computed as,

$$\begin{aligned} i_0 &= (P^{-1}(j*4+0)/4), \\ i_1 &= (P^{-1}(j*4+1)/4), \\ i_2 &= (P^{-1}(j*4+2)/4), \\ i_3 &= (P^{-1}(j*4+3)/4), \end{aligned} \tag{4.34}$$

where P^{-1} is the linear function given in Eq. (4.28).

4.2.3. Attacking a Nibble After S-box at Round 2

Let the target nibble be the 0^{th} nibble after the S-box of round 2, v_0^2,

$$v_0^2 = S_2(w_0^2) = S_2(u_0^1 \oplus k_0^2), \tag{4.35}$$

where the value u_0^1 is the input nibble to round 2 and $k_0^2 \in K_2$. u_0^1 is derived from 4 nibbles resulting from the S-box of the previous round, the index of which can be computed using Eq. (4.34). For the 0^{th} nibble, we have $j = 0$, in which case the index of the source nibbles will be,

$$\begin{aligned} i_0 = (P^{-1}(j*4+0)/4) &\implies (P^{-1}(0)/4) = 0, \\ i_1 = (P^{-1}(j*4+1)/4) &\implies (P^{-1}(1)/4) = 1, \\ i_2 = (P^{-1}(j*4+2)/4) &\implies (P^{-1}(2)/4) = 2, \\ i_3 = (P^{-1}(j*4+3)/4) &\implies (P^{-1}(3)/4) = 3. \end{aligned} \tag{4.36}$$

The value u_0^1 is then derived from the first four nibbles after S-box at round 1. These four nibbles are,

$$\begin{aligned} v_0^1 &= S_1(p_0 \oplus k_0^1), \quad & v_1^1 &= S_1(p_1 \oplus k_1^1), \\ v_2^1 &= S_1(p_2 \oplus k_2^1), \quad & v_3^1 &= S_1(p_3 \oplus k_3^1), \end{aligned} \tag{4.37}$$

where p_0, p_1, p_2, p_3 are the first four plaintext nibbles We then can write $u_0^1 = b_0 \oplus b_1 \oplus b_2 \oplus b_3$ such that,

$$\begin{aligned} b_0 &= (v_0^1 \wedge 2^3) \gg 0, \quad & b_1 &= (v_1^1 \wedge 2^3) \gg 1, \\ b_2 &= (v_2^1 \wedge 2^3) \gg 2, \quad & b_3 &= (v_3^1 \wedge 2^3) \gg 3. \end{aligned} \tag{4.38}$$

Combining this with Eq. (4.31) and Eq. (4.33), we can write,

$$u_0^1 = (S_1(p_0 \oplus k_0^1) \wedge 2^3 \gg 0) \oplus (S_1(p_1 \oplus k_1^1) \wedge 2^3 \gg 1) \\ \oplus (S_1(p_2 \oplus k_2^1) \wedge 2^3 \gg 2) \oplus (S_1(p_3 \oplus k_3^1) \wedge 2^3 \gg 3), \tag{4.39}$$

which can then be rewritten as,

$$u_0^1 = (S_1(p_0 \oplus k_0^1) \bullet m_0) \oplus (S_1(p_1 \oplus k_1^1) \bullet m_1) \oplus (S_1(p_2 \oplus k_2^1) \bullet m_2) \oplus (S_1(p_3 \oplus k_3^1) \bullet m_3). \tag{4.40}$$

Plugging this value of u_0^1 into Eq. (4.35) we obtain,

$$v_0^2 = S_2((S_1(p_0 \oplus k_0^1) \bullet m_0) \oplus (S_1(p_1 \oplus k_1^1) \bullet m_1) \oplus (S_1(p_2 \oplus k_2^1) \bullet m_2) \\ \oplus (S_1(p_3 \oplus k_3^1) \bullet m_3) \oplus k_0^2). \tag{4.41}$$

Now we use the same method as was used in the case of AES. Depending on how many key bits we want to extract, we have to keep some plaintext nibbles constant during the attack. Let's say that we are interested in extracting 1 byte of key, which here could be k_0^1 and k_1^1. In which case, we consider $[(S(p_2 \oplus k_2^1) \bullet m_2) \oplus (S(p_3 \oplus k_3^1) \bullet m_3) \oplus k_0^2] = \gamma$ and we can write v_0^2 as,

$$v_0^2 = S_2((S_1(p_0 \oplus k_0^1) \bullet m_0) \oplus (S_1(p_1 \oplus k_1^1) \bullet m_1) \oplus \gamma). \tag{4.42}$$

Keeping γ constant, we can perform an SCA on the set of 12 bits, (k_0^1, k_1^1, γ) thereby giving us a byte of the key (or 2 key nibbles) comprising k_0^1 and k_1^1. γ here clearly depends on 2 plaintext nibbles p_2 and p_3 which need to be kept constant for a successful attack. In order to reach the generalised form for computing the hypothesis in more inner rounds, we expand further on the above.

4.2.4. Attacking a Nibble After S-box at Round 3

Let's start with a similar assumption of attacking the 0^{th} nibble at round 3, represented by v_0^3.

$$v_0^3 = S_3(w_0^3) = S_3(u_0^2 \oplus k_0^3), \tag{4.43}$$

where u_0^2 is the input nibble to round 3 and $k_0^3 \in K_3$. The value u_0^2 can then be represented using 4 nibbles which were input to the previous round, that is, round 2. This can be written as,

$$u_0^2 = (S_2(u_0^1 \oplus k_0^2) \bullet m_0) \oplus (S_2(u_1^1 \oplus k_1^2) \bullet m_1) \\ \oplus (S_2(u_2^1 \oplus k_2^2) m_2) \oplus (S_2(u_3^1 \oplus k_3^2) \bullet m_3), \tag{4.44}$$

where $u_0^1, u_1^1, u_2^1, u_3^1$ are the first 4 input nibbles at round 2 and $(k_0^2, k_1^2, k_2^2, k_3^2) \in K_2$. Considering $[(S(u_1^1 \oplus k_1^2) \bullet m_1) \oplus (S(u_2^1 \oplus k_2^2) \bullet m_2) \oplus (S(u_3^1 \oplus k_3^2) \bullet m_3) \oplus k_0] = \theta$ and substituting the value of u_0^2 in Eq. (4.43),

$$v_0^3 = S_3((S_2(u_0^1 \oplus k_0^2) \bullet m_0) \oplus \theta), \tag{4.45}$$

the value u_0^1 being an input nibble to round 2 can be derived from equation 4.40. The above equation can then be written as,

$$v_0^3 = S_3((S_2((S_1(p_0 \oplus k_0^1) \bullet m_0) \oplus (S_1(p_1 \oplus k_1^1) \bullet m_1) \\ \oplus (S_1(p_2 \oplus k_2^1) \bullet m_2) \oplus (S_1(p_3 \oplus k_3^1) \bullet m_3) \oplus k_0^2) \bullet m_0) \oplus \theta), \tag{4.46}$$

where p_0, p_1, p_2, p_3 are plaintext nibbles and $(k_0^1, k_1^1, k_2^1, k_3^1) \in K_1$. Similar to what we did in round 2, we can pick 2 key nibbles to attack thereby requiring the other 2 corresponding plaintext nibbles to be constant. So, if we consider $[(S(p_2 \oplus k_2^1) \bullet m_2) \oplus (S(p_3 \oplus k_3^1) \bullet m_3) \oplus k_0^2] = \gamma$, we have,

$$v_0^3 = S_3((S_2((S_1(p_0 \oplus k_0^1) \bullet m_0) \oplus (S_1(p_1 \oplus k_1^1) \bullet m_1) \oplus \gamma) \bullet m_0) \oplus \theta), \tag{4.47}$$

where γ depends on 2 plaintext nibbles, while θ depends on u_1^1, u_2^1, u_3^1. As seen from Eq. (4.40), any u_i^1 will depend on 4 plaintext nibbles. Therefore, θ depends on a total of 12 plaintext nibbles. In order to attack v_0^3 so as to retrieve 1 key byte from $(k_0^1, k_1^1, \gamma, \theta)$, we need to have 15 plaintext nibbles as constants. Statistical attacks on the S-box here such as DPA thus might not be successful since this would only give us a leeway of only 4 bits or 16 possible plaintext for obtaining the traces. However, it would be possible perform a profiled attack with a good success rate in this case. For example, with enough samples for each of the Hamming Weight classes, it is possible to train an efficient CNN model and perform the attack successfully for guessing all the 16 target bits.

At this point, it is possible to increase the number of bits to attack while reducing the number of plaintext bits to keep constant. If we consider $(S(p_3 \oplus k_3^1) \bullet m_7) \oplus k_0^2 = \gamma$, then we have to attack 20 bits, that is, $(k_0^1, k_1^1, k_2^1, \gamma, \theta)$ but this time with 14 plaintext nibbles constant. This immediately gives us 8 bits to vary during the attack wherein we have 256 possible plaintexts for obtaining traces, which would increase the success rate of attacks giving us more samples to work with, as opposed to the success rate one would have while attacking only 16 possible plaintexts. This would even improve the efficiency of statistical methods such as DPA to a significant extent.

4.2.5. On the Attack Feasibility After S-box at Round 4

Let's consider v_0^4 is the nibble to be attacked which as given in the previous sections, can be written as,

$$v_0^4 = S_4(u_0^3 \oplus k_0^4), \tag{4.48}$$

where u_0^3 is the input nibble to round 4. u_0^3 can in turn be represented using the input nibbles to round 3, as shown in the previous section.

$$u_0^3 = (S_3(u_0^2 \oplus k_0^3) \bullet m_0) \oplus (S_3(u_1^2 \oplus k_1^3) \bullet m_1) \\ \oplus (S_3(u_2^2 \oplus k_2^3) \bullet m_2) \oplus (S_3(u_3^2 \oplus k_3^3) \bullet m_3). \tag{4.49}$$

Using a similar expansion as the ones given in Eq. (4.45), we have,

$$\theta_2 = (S_3(u_1^2 \oplus k_1^3) \bullet m_1) \oplus (S_3(u_2^2 \oplus k_2^3) \bullet m_2) \oplus (S_3(u_3^2 \oplus k_3^3) \bullet m_3) \oplus k_0^4, \tag{4.50}$$

$$\theta_1 = (S_2(u_1^1 \oplus k_1^2) \bullet m_1) \oplus (S_2(u_2^1 \oplus k_2^2) \bullet m_2) \oplus (S_2(u_3^2 \oplus k_3^2) \bullet m_3) \oplus k_0^3, \tag{4.51}$$

where $u_0^1, u_1^1, u_2^2, u_3^3$ are the input nibbles to round 2 which affect the first nibble of round 3 u_0^2. Retracing this to the plaintext nibbles, similar to Eq. (4.46), we have,

$$v_0^4 = S_4((S_3((S_2((S_1(p_0 \oplus k_0^1)) \oplus (S_1(p_1 \oplus k_1^1) \bullet m_1) \oplus (S_1(p_2 \oplus k_2^1) \bullet m_2) \oplus (S_1(p_3 \oplus k_3^1) \bullet m_3) \oplus k_0^2) \bullet m_0) \oplus \theta_1) \bullet m_0) \oplus \theta_2). \tag{4.52}$$

As before, while attacking a key byte, we consider $(S(p_2 \oplus k_2^1) \bullet m_2) \oplus (S(p_3 \oplus k_3^1) \bullet m_3) \oplus k_0^2) = \gamma$, and we can rewrite the above as,

$$v_0^4 = S_4((S_3((S_2((S_1(p_0 \oplus k_0^1) \bullet m_0) \oplus (S_1(p_1 \oplus k_1^1) \bullet m_1) \oplus \gamma) \bullet m_0) \oplus \theta_1) \bullet m_0) \oplus \theta_2). \tag{4.53}$$

The values γ and θ_1 as seen in the attack on round 2 and round 3 S-box would depend on 2 plaintext nibbles and 12 plaintext nibbles, respectively. θ_2 depends on u_1^2, u_2^2, u_3^2 which are nibbles at the input of round 3. From Eq. (4.47) we can say that a nibble at round 3 would depend on a set of bits determined by the 2 plaintext nibbles and the 2 constants which in turn depend on 15 plaintext nibbles in total. We can thus infer that any nibble at round 3 input would depend on all the 16 plaintext nibbles. Therefore, θ_2 depends on a total of 48 plaintext nibbles, which effectively is the entire plaintext.

A DPA attack on the round 4 S-box here would pertain to attacking a set of 20 bits, that is, $(k_0^1, k_1^1, \gamma, \theta_1, \theta_2)$. But such an attack would not be feasible as these bits depend on all of the plaintext. In this case, we even cannot decrease the number of constant bits by agreeing to attack more bits, as was done while attacking the nibble at round 3.

4.2.6. Finding the Remaining Key Bits

The above methodologies can be used to extract the first 64 bits of the first round key, which would be the most significant 64 bits of the actual key. In order to find the remaining 16 bits of the key we would have to leverage the obtained first round key while attacking the nibbles in the inner rounds, particularly the second round. As per the key schedule, the second round key would be computed from the first round key and the key register is updated as follows,

1. $[k_{79}k_{78}..k_1k_0] = [k_{18}k_{17}..k_{20}k_{19}]$
2. $[k_{79}k_{78}k_{77}k_{76}] = S[k_{79}k_{78}k_{77}k_{76}]$
3. $[k_{19}k_{18}k_{17}k_{16}k_{15}] = [k_{19}k_{18}k_{17}k_{16}k_{15}] \oplus$ `round_counter`,

where k_{79} is the most significant key bit, while k_0 is the least significant. We already have the most significant 64 bits of the key from the first round key nibbles $[k_0^1, ..., k_{15}^1]$, which in terms of bits in the key register is represented as $[k_{79}k_{78}..k_{17}k_{16}]$. These key bits can then be used to find the output nibbles of the first round, which we denote using u_i^1.

We need to target the remaining key bits $[k_{15}k_{14}..k_1k_0]$, which in the second round would correspond to the bits $[k_{76}k_{75}..k_{62}k_{61}]$ in the key register. These bits can be obtained by guessing the key nibbles $[k_1^2, k_2^2, k_3^2, k_4^2]$ of the second round key. We can then hypothesize the S-box output of the second round, for instance as follows,

$$v_0^2 = S_2(u_0^1 \oplus k_0^2), \tag{4.54}$$

where v_0^2 is the first nibble obtained after S_2. Since we already have u_i^1 , this attack then becomes similar to attacking the first round and the target hypothesis then depends on only the second round key wherein the data complexity of the attack remains 2^8. In this way, we can then guess the aforementioned targeted nibbles of the second round key, revert the key schedule operation and find their corresponding input key bits thereby giving us the remaining 16 bits of the user input key.

4.2.7. Generalization of the Attack on the Inner Rounds

In the above attacks, we use m_0 for representing $(\wedge 2^3 \gg 0)$, m_1 for representing $(\wedge 2^3 \gg 1)$, and so on. However, these notations can be considered generic and we can represent m_i with continuously incrementing index, but with its value belonging to a finite set defined by the position of the target nibble itself.

On the basis of the above equations such as Eq. (4.47) and Eq. (4.53), we can generalise computing the value of the nibble v_i^j as the i^{th} nibble obtained after S-box at round j as,

$$\begin{aligned} v_i^j = S_j(S_{j-1}(S_{j-2}(...S_2(S_1(p_n \oplus k_n^1) \bullet m_1 \oplus S_1(p_{n+1} \oplus k_{n+1}^1) \bullet m_2 \oplus \gamma) \\ \bullet m_3 \oplus \theta_1) \bullet m_4 \oplus \theta_2)... \bullet m_j \oplus \theta_{j-2}), \end{aligned} \tag{4.55}$$

where p_n, p_{n+1} are consecutive plaintext nibbles and k_n^1, k_{n+1}^1 are corresponding key nibbles belonging to 1st round key. j itself depends on which plaintext nibbles affect v_i^j which can be derived from the function P^{-1}. γ depends on 2 plaintext nibbles, and any θ_j depends on 3×4^j plaintext nibbles. The number of bits to attack in order to find 1 key byte would be $4(j+1)$. Table 4.2 gives the number of constant plaintext bytes required for the largest θ in an equation, which is θ_{j-2} for round j and the number of bits to attack for finding 1 byte of the key.

4.3. On the Complexity of Attacking the Inner Rounds

Due to the diffusion of an SPN structure, attacking a byte in the inner rounds is more difficult to attack in the case of both AES and PRESENT. We confirm the same in our work as seen in Sections 4.1.6 and 4.2.7, with respect to the effort that needs to be put in by the attacker. As per Table 4.1, round 2 requires 3 plaintext bytes to be fixed in order to derive 1 key byte, while round 3 requires 15 plaintext bytes to be fixed to extract the same amount of data. Attacks on round 2, requiring only 3 plaintext bytes need to be fixed per key byte, can be combined by grouping the constant bytes in order to attack multiple key bytes at one

Round i	No. of fixed plaintext nibbles (effectively) to attack k_0^1, k_1^1 and predict $S_i(w_0^i)$	No. of bits to be guessed in order to attack k_0^1, k_1^1 and predict $S_i(w_0^i)$	Plaintext × Key options to attack k_0^1, k_1^1 and to predict $S_i(w_0^1)$ (p=Profile, a=Attack)
1	0	8	p=$2^{64} \times 2^8$, a=$2^{64} \times 1$
2	2	12	p=$2^{60} \times 2^8$, a=$2^{60} \times 1$
3	14	16	p=$2^8 \times 2^8$, a=$2^8 \times 1$
4	48 (16)	20	p=1×2^8, a=1×1
5	192 (16)	24	p=1×2^8, a=1×1

Table 4.2: Number of constant plaintext nibbles required and number of bits to attack for 1 key byte for first 5 rounds of PRESENT-80. Attacking the S-box at the first round need not have any nibble fixed while round 2 is influenced by γ (and not by any θ_j), which needs 2 plaintext nibbles fixed.

go. For example, in order to attack key byte 0, we need to have bytes 5, 10, 15 fixed in the plaintext, for key byte 4, bytes 3, 9, 14 need to be constant, and so on. As long as the bytes that need to be constant form a disjoint set, we can combine the constant plaintext bytes to target multiple key bytes using a single acquisition of traces. Such an attack in round 2 is summarised in Table 4.3.

Target Key Bytes	**Plaintext bytes that need to be fixed**
(0, 4, 8, 12)	(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)
(1, 5, 9, 13)	(0, 2, 3, 4, 6, 7, 8, 10, 11, 12, 14, 15)
(2, 6, 10, 14)	(0, 1, 3, 4, 5, 7, 8, 9, 11, 12, 13, 15)
(3, 7, 11, 15)	(0, 1, 2, 4, 5, 6, 8, 9, 10, 12, 13, 14)

Table 4.3: Grouping of acquisitions while targeting multiple key bytes after round 2 Sbox.

We can therefore, conclude that if attacking round 1 needs x traces to extract all 16 key bytes, attacking round 2 would require $4x$ traces owing to multiple acquisitions. This exponentially increases as we progress through the rounds as round 3 requires 15 plaintext bytes fixed. That is, it is possible to have only 1 plaintext byte as a variable in order to extract 1 key byte, which in turn means that the attacker needs to perform separate acquisitions for each key byte. Therefore, extracting all the key bytes while attacking round 3 would involve the acquisition of $16x$ traces, an exponentially higher effort for the attacker.

The same can be observed in the case of PRESENT as well. While round 2 requires a byte of plaintext to be fixed to determine 1 key byte, round 3 requires 7 bytes of plaintext to be fixed leaving only 1 byte to be allowed to vary. Similar to that of AES, it is possible to group the acquisitions for round 2 of PRESENT but not for round 3. That is, if while attacking round 1, an attacker needs x traces, round 2 would require $4x$ traces and round 3 would require $8x$ traces because of the multiple chosen-plaintext based acquisition of power traces.

5

Experimental Results

In this chapter, we discuss our experimental results where we provide results for attacks on AES-128 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 5.3 gives an insight into the deep learning architecture that we use to perform the attacks. Sections 5.4 and 5.5 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 [1] 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 the target device. The setup consisted of a Riscure current probe[2], 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 4.1 and Table 4.1.

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,

$$X^* = X + \mathcal{N}(\mu_x, \sigma_x^2), \tag{5.1}$$

[1] Pinata Board: https://www.riscure.com/product/pinata-training-target/
[2] Current probe: https://www.riscure.com/product/current-probe

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 wherein we select a part of the first trace as the reference and computed the Pearson correlation for each offset within a chosen range for each following trace. We then shifted each trace by the respective offset that maximizes the correlation. Appendix B shows three consecutive traces from our dataset before and after alignment. Since the misalignment might not be very significant and is not used as a common countermeasure, we also test the effectiveness of the attacks on traces with both Gaussian noise and misalignment.

5.2. Power Traces

The next step would be to extract the Points of Interest (PoI) from the given set of traces. For example, while attacking round 3 S-box, we need to select the points in time which roughly correspond to the times when the particular computation of the byte after round 3 S-box would take place for all the traces. If the traces are misaligned, the selected PoIs would only be a rough estimate and not the exact starting and ending points for the computation of the target byte. The first power trace in the set is given in Figure 5.1.

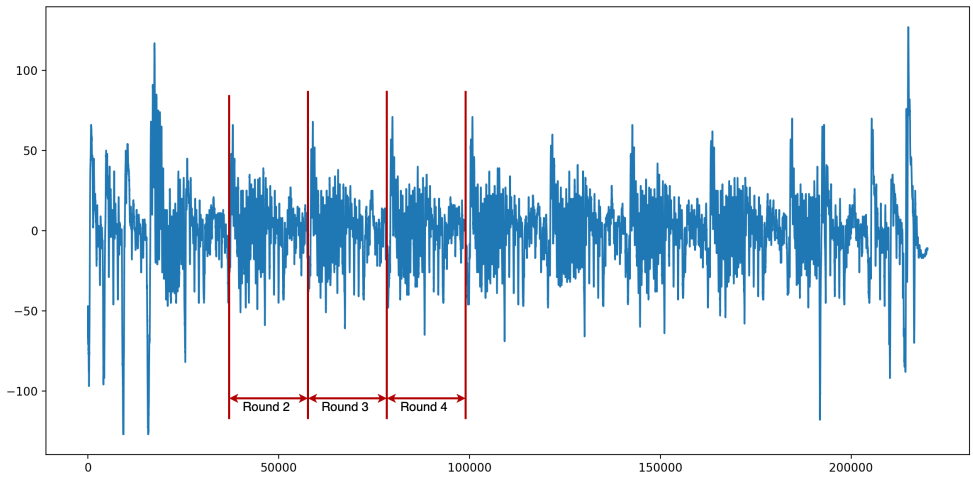

Figure 5.1: The first trace of the dataset

The 10 patterns indicating the the 10 rounds of AES-128 can be clearly seen, with rounds 2,3 and 4 being highlighted. With a simple SPA analysis, and zooming into the power trace, we can approximately extract the PoI required for attacking the S-box of the inner rounds. Figure 5.2 shows a close-up of the above traces focusing on the third round. We can pick the first few points as the Sbox is the first process to be performed in any round of AES. We,

therefore, consider the points 58 000-60 960 as our PoI for attacks using DL-SCA. A similar extraction is done in the case of round 2 and round 4. While the points 37 000-39 500 are used to attack round 2 S-box, the points 77 500-80 000 are used to attack round 4 S-box.

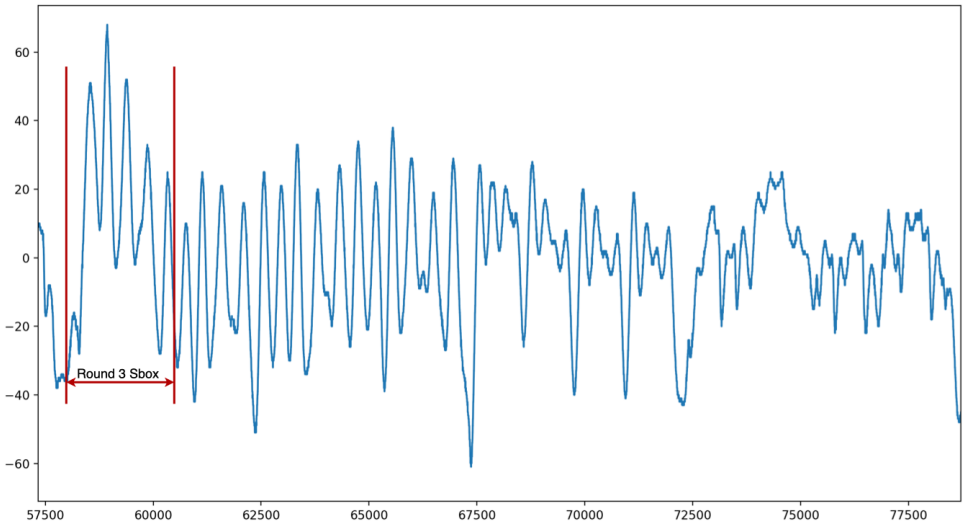

Figure 5.2: Zooming into the 3rd round of AES power trace. 58000-60960 were chosen as the potential PoIs for the purpose of DL-SCA on the 3rd round Sbox.

We shall now have a look at the Deep Learning model's architecture used during the attack, the results of the attack and subsequently compare these results with those obtained from CPA.

5.3. The Deep Learning Model Architecture

CNN architectures analogous to VGG [43] have been shown to give good results in the field of DL-SCA [2, 20] 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. [2].

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 **??**.

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. [2] also showed CNN_{best} has an equally good performance with 50 epochs as well. We observed that while

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

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 for round 3 for example, then has the shape of (2 960 × 64) where 2 960 is the number of PoIs (or features) selected. This CNN architecture is shown in Figure 5.3, while its `tensorflow` summary is given in Appendix-A. Table 5.1 shows the benchmarked values used for CNN_{best} and the values that we consider for this work.

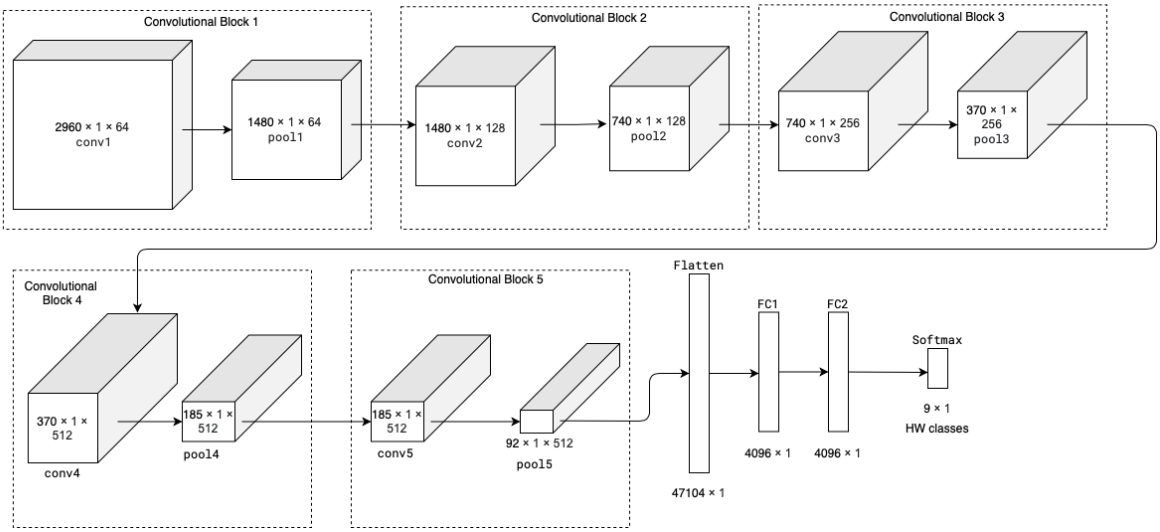

Figure 5.3: Pictorial representation of the CNN_{best} architecture

While we use the above architecture for our proof-of-concept results, we also test our hypothesis with randomized CNN architectures. We observed that most of these random architectures also showed good results in breaking the inner rounds and discuss them in Section 5.8.

5.4. Attacking a Byte After S-box at Round 2

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. (4.25), we compute the hypothesis for attacking key byte 0 as follows,

$$hyp = HW[S(02 * S(p_0 \oplus k_0) \oplus \delta)], \tag{5.2}$$

where $\delta = 03 * S(p_5 \oplus k_5) \oplus 01 * S(p_{10} \oplus k_{10}) \oplus 01 * S(p_{15} \oplus k_{15}) \oplus k_0^1$. 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 5.3. 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 5.4 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. (5.2). 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 5.4. 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 5.5, while CPA does not find the key even after 2 000 traces despite the downward trend that we see in Figure 5.5. The average rank given by CPA, in this case, is 352 after 2 000 traces while it attempts to guess 16 bits of information.

5.5. Attacking a Byte After S-box at Round 3

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,

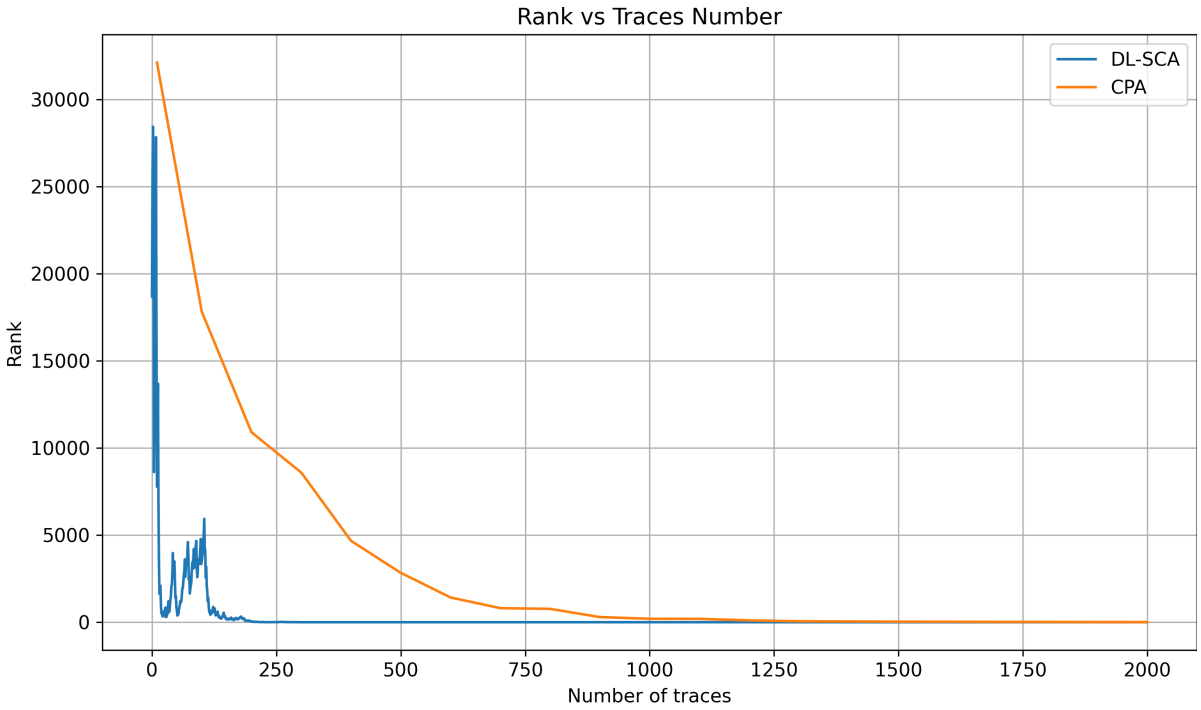

Figure 5.4: DL-SCA and CPA for key byte 0 after S-box on encryption round 2.

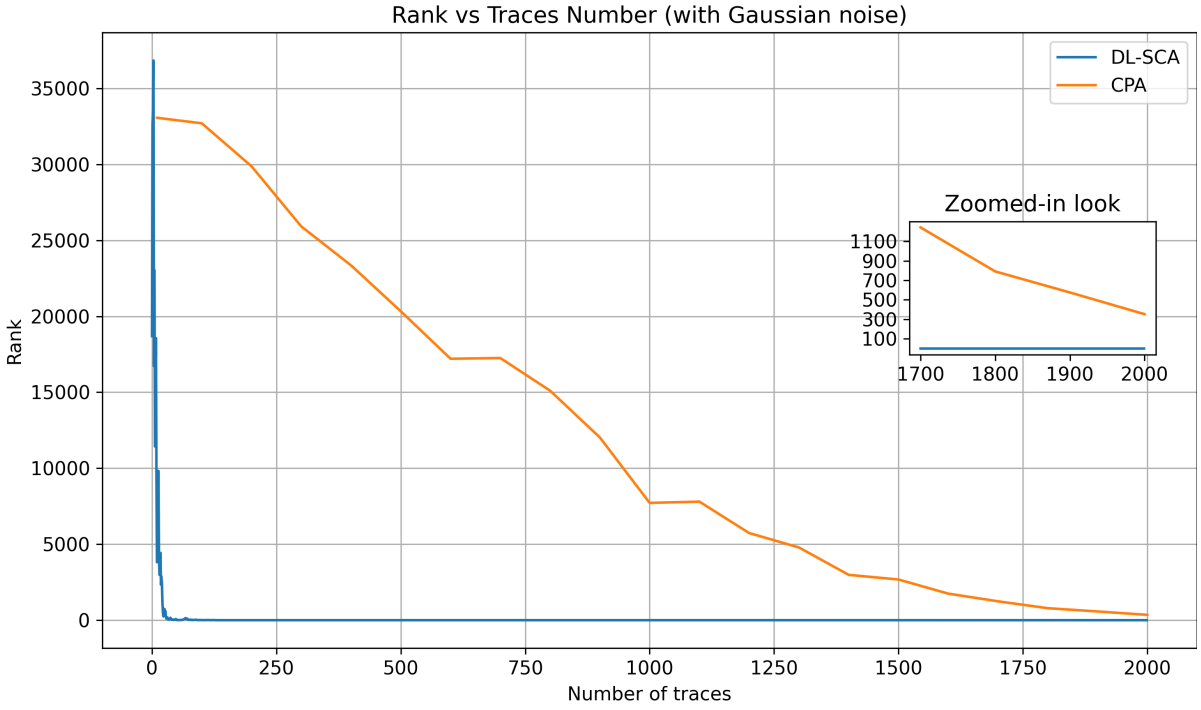

Figure 5.5: DL-SCA and CPA after adding Gaussian Noise for key byte 0 after S-box on encryption round 2.

$$hyp = HW[S(02 * S(02 * S(p_0 \oplus k_0) \oplus \delta) \oplus \gamma)], \tag{5.3}$$

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 5.3. As done for round 2, the label for each trace is computed using Eq. (5.3) 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 5.6 (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 5.6. 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 5.7.

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 5.8. 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 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.*

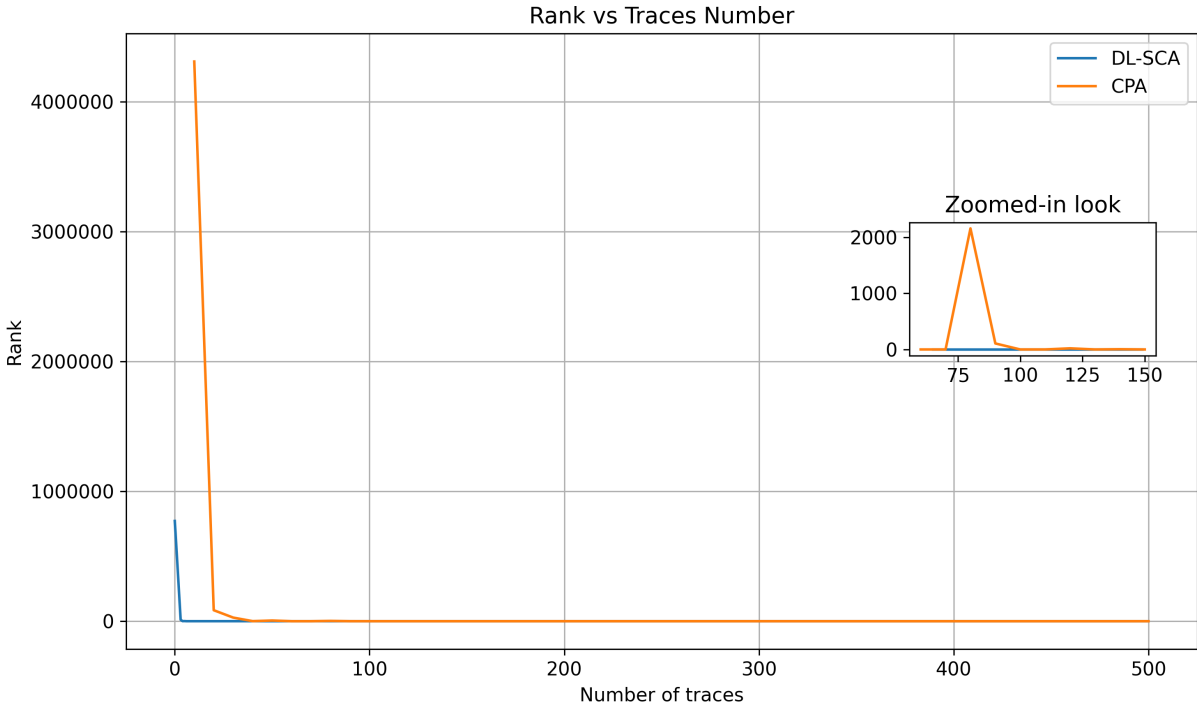

Figure 5.6: DL-SCA and CPA on aligned traces for key byte 0 after S-box on encryption round 3.

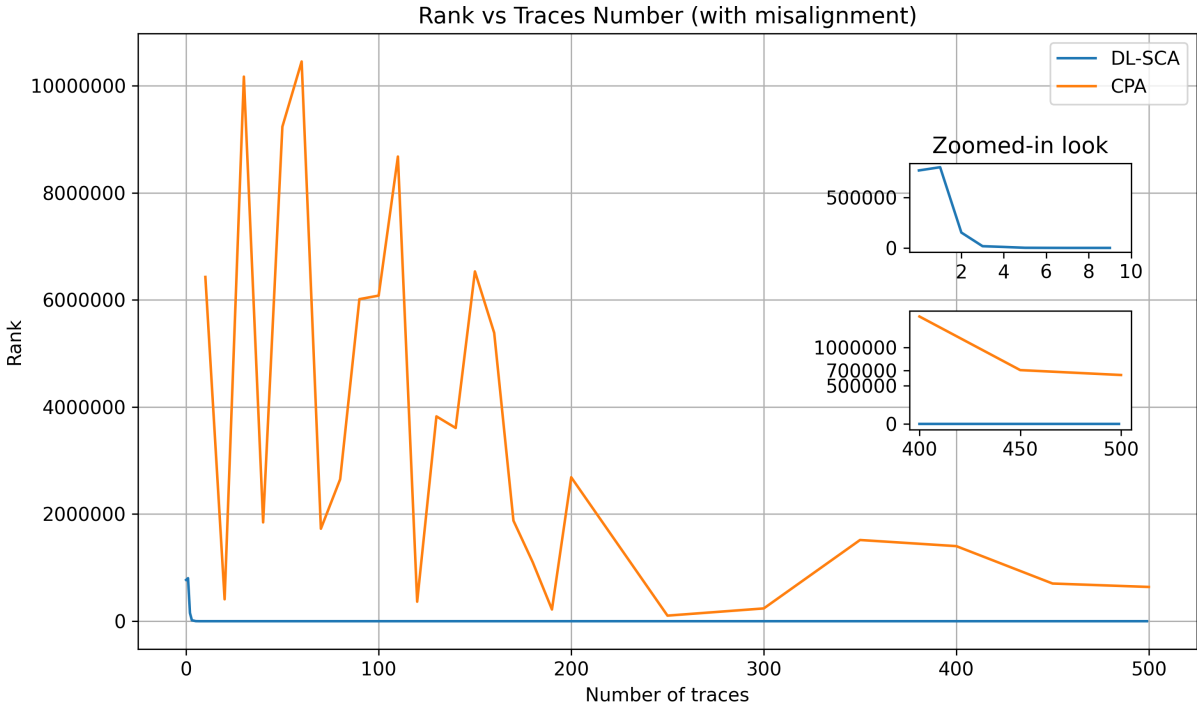

Figure 5.7: DL-SCA and CPA on misaligned traces for key byte 0 after S-box on encryption round 3.

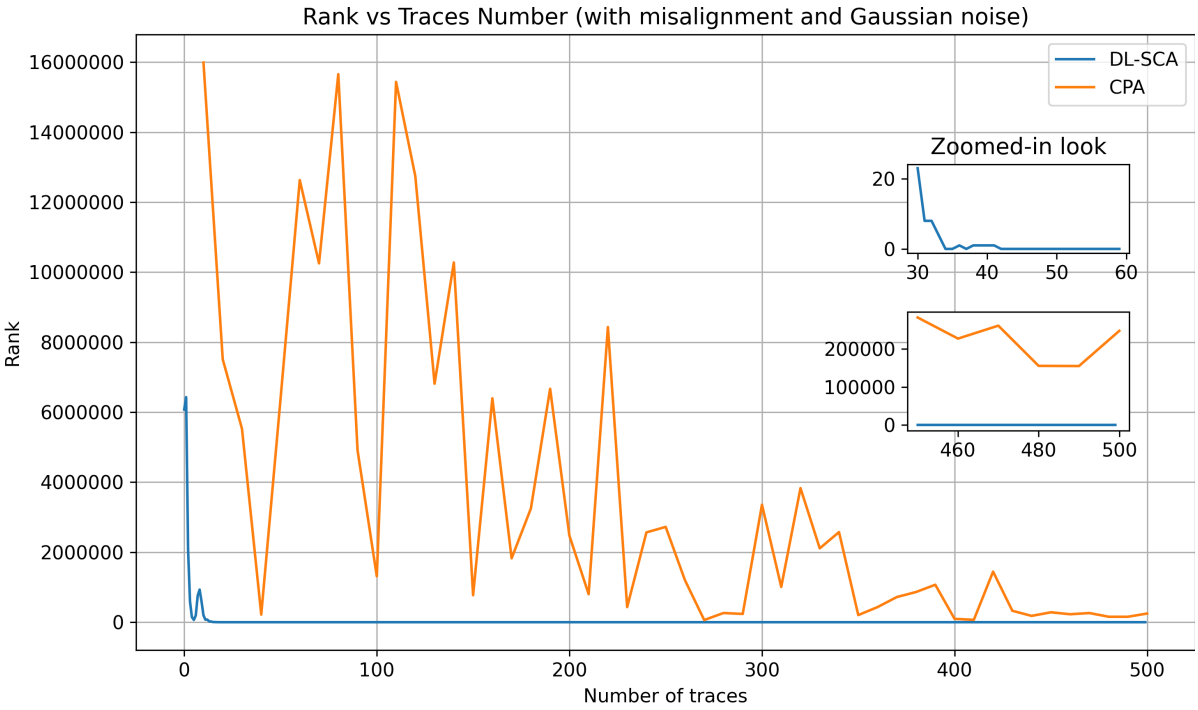

Figure 5.8: DL-SCA and CPA on misaligned traces after adding Gaussian noise for key byte 0 after S-box on encryption round 3.

5.6. Attacking a Byte After S-box at Round 4

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. (4.17) and Table 4.1. 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 carried out on data that are completely different from each other, thereby rendering the attack unsuccessful. The results for the same are shown in Figure 5.9, 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.*

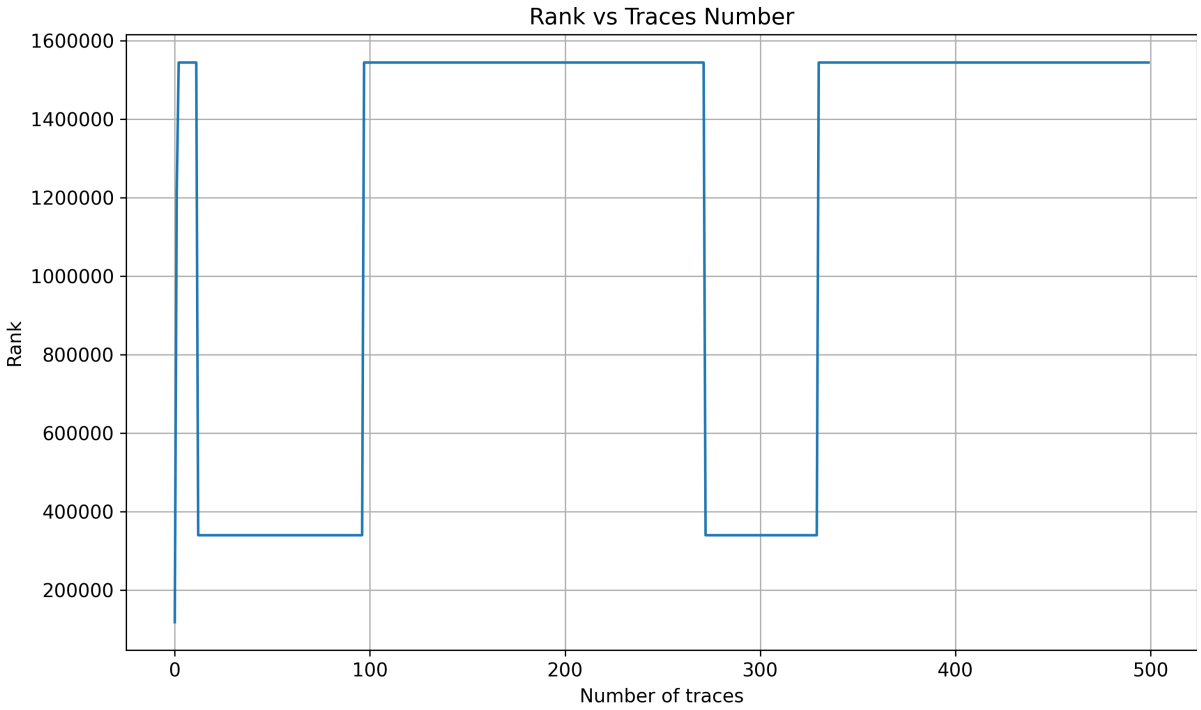

Figure 5.9: DL-SCA on round 4 S-box with different plaintexts being used while training and constant plaintext used for attack. A fixed key was used both for profiling and for attack.

5.7. Sanity checks for DL-SCA Results

At this point, since we are effectively working only on 1 byte and only 1 fixed key during the profiling phase as well, we need to make sure our model is not biased in any way. That is, we need to make sure it is not over-fitting to the key we have provided and will therefore be able to generalize it in case of the use of a different key. We demonstrate this with the help of the following "sanity check" use-cases by taking the round 3 hypothesis as an example. We note here that when the constants are set right, and the hypothesis is computed as per the equations, we obtain a training accuracy of 99% and a validation accuracy of 91%.

1. **Profiling and validation with random key bytes, but attacking with the correct ones** - In order to check a bias towards a particular key byte, we profile and validate with a random set of key bytes instead of using the correct ones. Since we are training with the wrong key bytes, the subsequent attack should ideally be unsuccessful. Apart from observing a reduced validation accuracy of 22.8% while the training accuracy was 71.78%. we also observe an unsuccessful attack as shown in Figure 5.10.

2. **Profiling on a random window of samples, but attacking on the correct window** - As aforementioned, round 3 Sbox occurs in the window (58000-60960). In order to check if the window of samples (PoIs) indeed does affect the model's training, we train the model in a randomly selected window of samples, say, (50 000-52 960). The validation and attack is done on the actual window of (58 000-60 960). Here, we observe a high training accuracy of almost 95% but a low validation accuracy of 24%. Since the profiling is done on an entirely different looking set of samples, the attack should not be successful. Accordingly we get the attack results as shown in Figure 5.11, which do not converge to the actual key byte used.

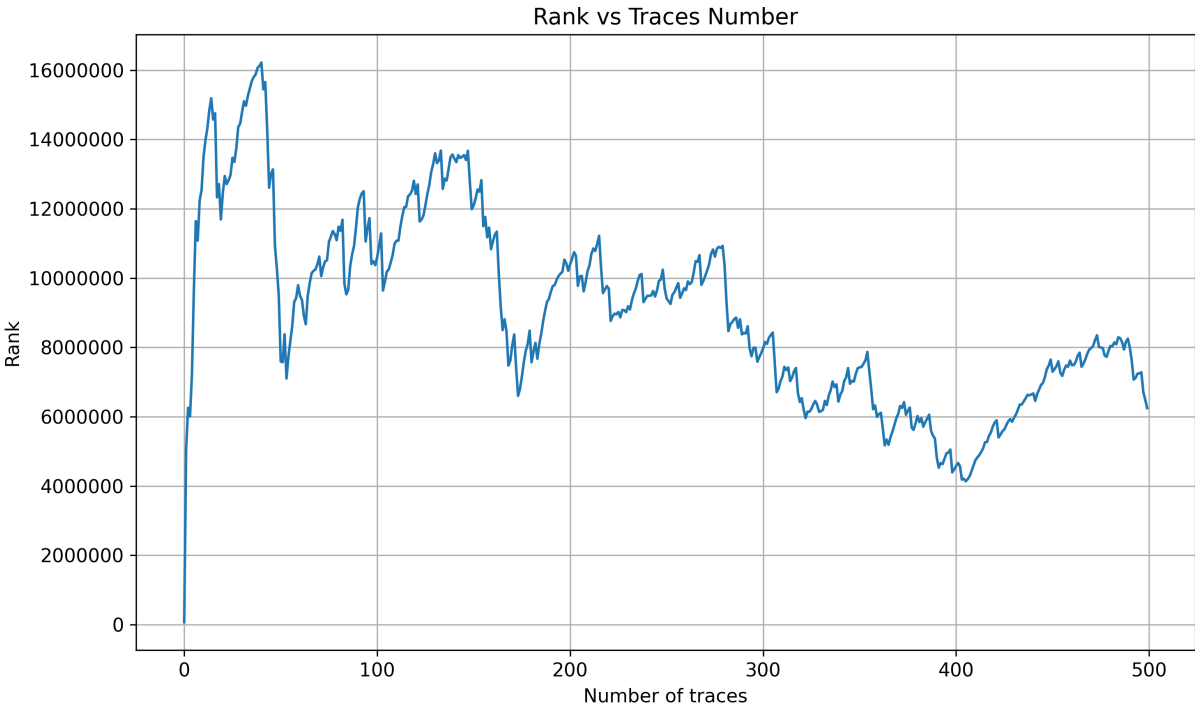

Figure 5.10: Results of profiling and validation with random keys, but attacking with the correct ones.

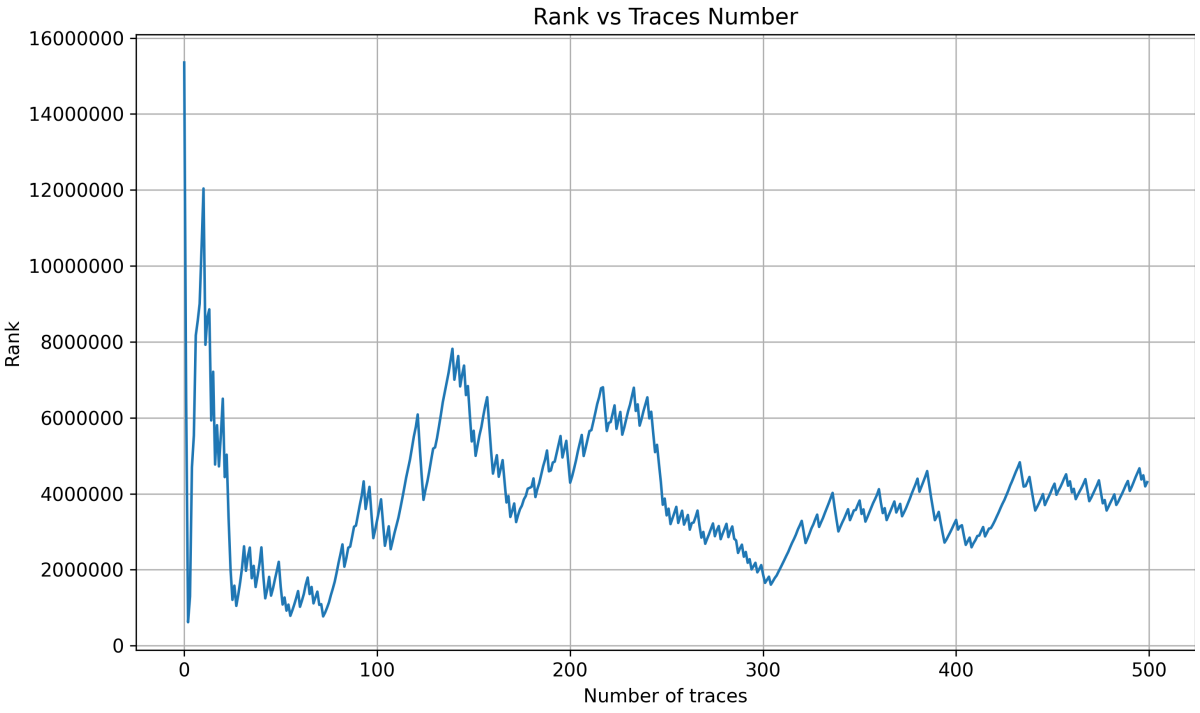

Figure 5.11: Results of profiling on random samples but attacking on the correct ones

3. **Attacking on a random window of samples, but profiling and validating using the correct one** - Here we profile using the right window (58000-60960) but attack using a random window of samples (50 000-52 960). The attack is again not successful, as one would expect, since the traces used for the attack phase are entirely different. The results are shown in Figure 5.12.

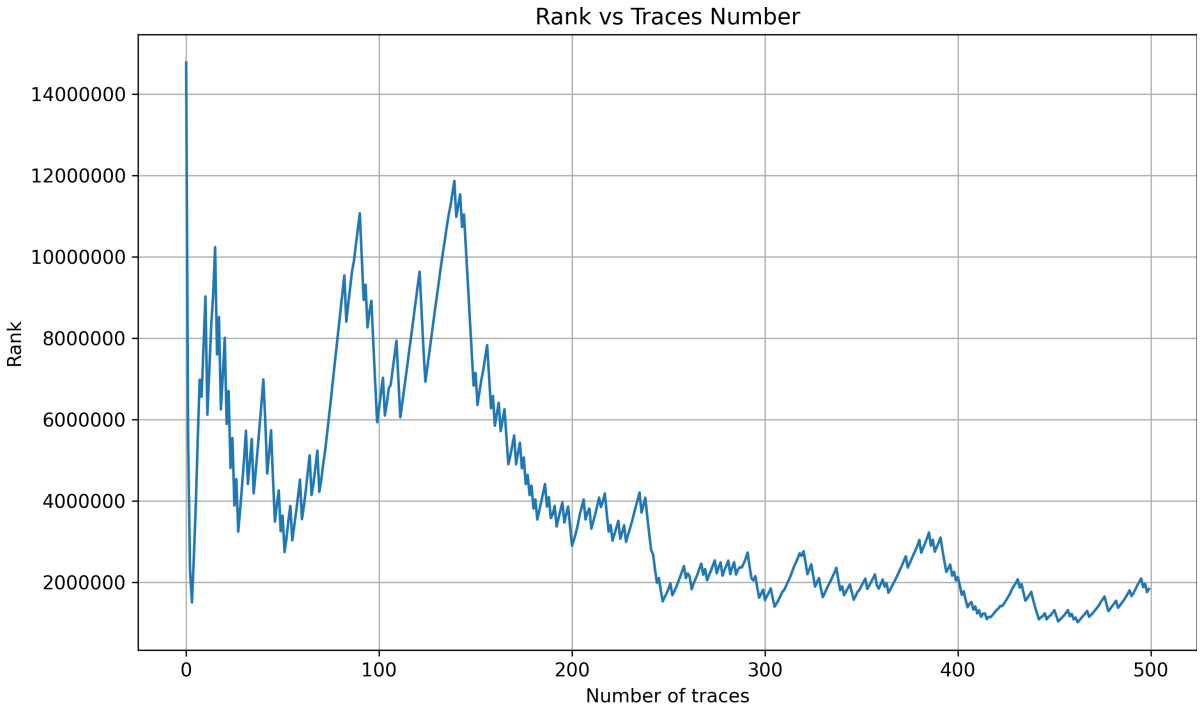

Figure 5.12: Results of attacking on random samples but using the correct samples for profiling and validation.

The above results clearly indicate that the DL model is indeed learning correctly from the traces and is performing as expected during the attack phase without a bias for 1 fixed key.

5.8. Attacks on Rounds 2 and 3 with Randomized CNN Architectures

As mentioned, we would also like to test how randomly generated models fare when performing the above attacks. These architectures have 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`).

As the execution of the attacks on round 3 take a significant amount of time (approx. 45 minutes for the configuration described in Section 5.5, we check the results for 10 randomized models for both rounds 2 and 3. For round 2, 8 out of 10 models were successful while attacking aligned traces, out of which 5 of them performed better than CNN_{best} in terms of the number of traces required to obtain a constant key rank of 0 for all subsequent traces. Similarly, for traces with the Gaussian noise added, 7 out of the 10 random models succeeded in finding the key but only 1 of these performed better than CNN_{best}. These results have been shown in Appendix C.

We also analyse the performance of random models on round 3, in which case we take into consideration misalignment and Gaussian noise as well. We observed that all the random models were successful when attacking the aligned traces. Although all these models

manage to find the key within 30 traces, only one of them finds the correct key with 10 traces thereby performing marginally better than CNN_{best}. As for the misaligned traces and for the ones with Gaussian noise, all the models succeed in the attack but none of them find the key with lesser number of traces than CNN_{best}. The results for attacks using these random CNN architectures have been shown in Appendix C.

As we can observe here that while most of the random models are able to succeed in the attack, few of them are even better than our reference model, CNN_{best}. In order to find the model that works the best for a particular target round, a detailed tuning of the hyperparameters is required such that the attack phase takes the least number of traces to guess the key. An attack on each round therefore needs an appropriately tuned model such that we can achieve the best possible performance while attacking that particular round on the target device.

6

Conclusion

In this chapter, we summarise the work we have done in the duration of this thesis. We first reflect over our hypothesis questions while also addressing our contributions when answering them. We then go through some limitations posed by our work and discuss the prospective future work that can be undertaken in this area to further improve our understanding and effectiveness of side-channel attacks on the inner rounds of AES-128 and PRESENT-80 ciphers.

6.1. On Research Questions and our Contributions

We use this section to answer each of our hypothesis questions as derived from this work.

1. Is it possible to perform side-channel attacks on the inner rounds of AES-128 and PRESENT-80?

   We show in Chapter 4 and Chapter 5 that it is indeed possible to attack any byte of the inner rounds in both AES-128 and PRESENT-80. While Maghrebi et al. [28] do make us aware of the possibility of attacking inner round bytes using chosen plaintexts. We extend on this methodology in order to generalize the computation of any intermediate byte for AES-128 from both input and output in encryption mode. We then extend the same technique to PRESENT-80 where we analyse its structure and formulate its round computation in such a way that we can compute the value of any nibble in any of its inner rounds as well. We include these generalizations while computing the leakage hypothesis for any byte in the inner rounds, which in our case is the Hamming Weight of the byte. We would like to note here that, to the best of our knowledge, such a formulation of PRESENT has not been done before wherein it is possible to compute a nibble in any of its intermediate rounds.

   Since our attack works in chosen plaintext scenarios, its feasibility will not only be determined by the number of bits one needs to guess but also on the practicality of traces acquired from such a dataset containing chosen plaintexts. We demonstrate this in Section 5.6, where we attempt to attack round 4 S-box which, as per our attack design, should have all constant plaintext bytes. Since no byte in the plaintext is allowed to vary, CPA as well as DL-SCA were unsuccessful. The number of fixed plaintext bytes also determine the number of acquisitions that would be needed in

order to recover the entire key. More the number of acquisitions required, more the effort that needs to be put in for a successful attack. For example, as seen in Table 4.1, while round 2 would require only 3 bytes to be fixed for retrieving 1 key byte, round 3 requires 15 plaintext bytes to be fixed. This means that while round 2 would require 4 acquisitions with 3 plaintext bytes fixed, attacking round 3 would require 16 acquisitions in order to find all the 16 bytes of key. This makes attacking round 3 more difficult in terms of the effort required to be put in to extract the key.

2. If the above attack is possible, can Deep Learning methods be used to attack these inner rounds and extract the key?

   We use Deep Learning to attack the S-box after rounds 2 and 3 successfully in Sections 5.4, 5.5. This proves the validity of the chosen plaintext strategy used to compute the leakage hypothesis for the inner rounds. 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 work, DL-SCA becomes a strong candidate to attack (not properly protected) inner rounds from AES.

6.2. Limitations

We take a chosen plaintext approach (in the case of encryption) to attack the inner rounds. This strategy reveals a clear limitation in terms of the availability of the such plaintexts when attacking the inner rounds which can be seen when trying to attack after the fourth round in Section 5.6. In this case, we are dependant on the plaintext and makes the attack more difficult even in scenarios when the attack is possible. For example, we see that in order to attack round 3, we need 3 times more number of acquisitions than is required to attack round 2.

Our work provides a good starting point for attacking the inner rounds for both AES and PRESENT. For AES, we were successfully able to run attacks on the inner rounds. Although the attacks were also tested on traces with Gaussian noise and misalignment, they did not have an explicit countermeasure implemented on the outer rounds. Our work still proves to be valid since we target the inner rounds which are assumed to be unprotected. We also realise that we have only been able to run tests on AES for which we were able to check the effectiveness of DL-SCA and its advantage over CPA. Therefore, another limitation of our work is the absence of traces for PRESENT, which would also give us a good idea about the performance of DL-SCA on the inner rounds of lightweight ciphers.

6.3. Future Work

As we have seen, 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, in the case of AES-128. 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 [27]. In this approach, attacking the Hamming Weight of both intermediates would be the most efficient wherein targeting these two intermediate states at once, the attack would be able to recover the key in a similar way to [4, 41, 46]. Note that this method can be applied without requiring access to input and output for AES. Similar results might be achievable using template attacks as well, but our choice is Deep Learning as it has been shown to outperform template attacks multiple times.

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 the neural network should be able to combine S-box input and output leakages and classify the XORed value correctly, in a similar way in which neural networks can combine leakages in the case of masked AES traces [25].

Another approach with respect to overcoming the limitation caused by the chosen plaintext attacks can be algebraic attacks. Renauld et al. [37] introduce algebraic side-channel attacks which works in an unknown plaintext/ciphertext setting and makes use of information leakages that can be derived from the intermediate rounds. Although this attack differs from the classical approach we take in the sense that there is not one particular round to attack, it would be interesting to compare the performance of DL-SCA on the inner rounds and algebraic attacks using templates introduced in by Renauld et al. in [38].

As we have noted before, there is a dearth of studies on attacking the inner rounds of PRESENT using the classical approach of SCA. While we introduce the theoretical aspects of computing the leakage from any intermediate byte, further study also needs to be done on the performance of DL-SCA on the inner rounds of PRESENT. We leave further investigating of the aforementioned ideas, as well as practical experiments for attacks in decryption mode, as future works.

Bibliography

[1] Mehdi-Laurent Akkar and Christophe Giraud. An implementation of DES and AES, secure against some attacks. In Çetin Kaya Koç, David Naccache, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 309–318. Springer, 2001. doi: 10.1007/3-540-44709-1_26. URL https://doi.org/10.1007/3-540-44709-1_26.

[2] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. 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.

[3] Andrey Bogdanov, Lars R. Knudsen, Gregor Leander, Christof Paar, Axel Poschmann, Matthew J. B. Robshaw, Yannick Seurin, and C. Vikkelsoe. PRESENT: an ultra-lightweight block cipher. In Pascal Paillier and Ingrid Verbauwhede, editors, *Cryptographic Hardware and Embedded Systems - CHES 2007, 9th International Workshop, Vienna, Austria, September 10-13, 2007, Proceedings*, volume 4727 of *Lecture Notes in Computer Science*, pages 450–466. Springer, 2007. doi: 10.1007/978-3-540-74735-2_31. URL https://doi.org/10.1007/978-3-540-74735-2_31.

[4] Hélène Le Bouder, Ronan Lashermes, Yanis Linge, Gaël Thomas, and Jean-Yves Zie. A multi-round side channel attack on AES using belief propagation. In Frédéric Cuppens, Lingyu Wang, Nora Cuppens-Boulahia, Nadia Tawbi, and Joaquín García-Alfaro, editors, *Foundations and Practice of Security - 9th International Symposium, FPS 2016, Québec City, QC, Canada, October 24-25, 2016, Revised Selected Papers*, volume 10128 of *Lecture Notes in Computer Science*, pages 199–213. Springer, 2016. doi: 10.1007/978-3-319-51966-1_13. URL https://doi.org/10.1007/978-3-319-51966-1_13.

[5] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004. doi: 10.1007/978-3-540-28632-5_2. URL https://doi.org/10.1007/978-3-540-28632-5_2.

[6] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 45–68. Springer, 2017.

[7] Suresh Chari, Charanjit Jutla, Josyula R Rao, and Pankaj Rohatgi. A cautionary note regarding evaluation of aes candidates on smart-cards. In *Second Advanced Encryption Standard Candidate Conference*, pages 133–147. Citeseer, 1999.

[8] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 398–412. Springer, 1999. doi: 10.1007/3-540-48405-1_26. URL https://doi.org/10.1007/3-540-48405-1_26.

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

[10] ChipWhisperer. Part 2, topic 2: Cpa on hardware aes implementation, jun 2020. URL https://github.com/newaetech/chipwhisperer-jupyter/blob/\5f2fdd75a2bc4cdbc08f212b0853ec0261a7c2a1/courses/sca201/\Lab%202_2%20-%20CPA%20on%20Hardware%20AES%20Implementation.ipynb.

[11] ChipWhisperer. Part 2, topic 3: Attacking across mixcolumns, nov 2020. URL https://github.com/newaetech/chipwhisperer-jupyter/blob/\5f2fdd75a2bc4cdbc08f212b0853ec0261a7c2a1/courses/sca201/\Lab%202_3%20-%20Attacking%20Across%20MixColumns.ipynb.

[12] ChipWhisperer. Chipwhisperer documentation, jun 2020. URL https://rtfm.newae.com/.

[13] Joan Daemen and Vincent Rijmen. Resistance against implementation attacks: A comparative study of the aes proposals. In *Proceedings of the Second Advanced Encryption Standard (AES) Candidate Conference*, volume 82, 1999.

[14] International Organization for Standardization (ISO). *Information technology – Security techniques – Part 3: Block ciphers*, iso/iec 18033-3:2005 edition, 2005.

[15] Richard Gilmore, Neil Hanley, and Maire O'Neill. Neural network based attack on a masked implementation of aes. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111. IEEE, 2015.

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[17] Christoph Herbst, Elisabeth Oswald, and Stefan Mangard. An aes smart card implementation resistant to power analysis attacks. In *International conference on applied cryptography and network security*, pages 239–252. Springer, 2006.

[18] Annelie Heuser and Michael Zohner. Intelligent machine homicide. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 249–264. Springer, 2012.

[19] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293, 2011.

[20] 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.

[21] 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.

[22] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[23] Paul C. Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *J. Cryptogr. Eng.*, 1(1):5–27, 2011. doi: 10.1007/s13389-011-0006-y. URL https://doi.org/10.1007/s13389-011-0006-y.

[24] Takaya Kubota, Kota Yoshida, Mitsuru Shiozaki, and Takeshi Fujino. Deep learning side-channel attack against hardware implementations of aes. *Microprocessors and Microsystems*, page 103383, 2020.

[25] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked aes. *Journal of Cryptographic Engineering*, 5(2):123–139, 2015.

[26] Jiqiang Lu, Jing Pan, and Jerry den Hartog. Principles on the security of AES against first and second-order differential power analysis. In Jianying Zhou and Moti Yung, editors, *Applied Cryptography and Network Security, 8th International Conference, ACNS 2010, Beijing, China, June 22-25, 2010. Proceedings*, volume 6123 of *Lecture Notes in Computer Science*, pages 168–185, 2010. doi: 10.1007/978-3-642-13708-2_11. URL https://doi.org/10.1007/978-3-642-13708-2_11.

[27] Houssem Maghrebi. Deep learning based side-channel attack: a new profiling methodology based on multi-label classification. *IACR Cryptol. ePrint Arch.*, 2020:436, 2020. URL https://eprint.iacr.org/2020/436.

[28] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[29] Amir Moradi and Tobias Schneider. Improved side-channel analysis attacks on xilinx bitstream encryption of 5, 6, and 7 series. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design - 7th International Workshop, COSADE 2016, Graz, Austria, April 14-15, 2016, Revised Selected Papers*, volume 9689 of *Lecture Notes in Computer Science*, pages 71–87. Springer, 2016. doi: 10.1007/978-3-319-43283-0_5. URL https://doi.org/10.1007/978-3-319-43283-0_5.

[30] Amir Moradi, Markus Kasper, and Christof Paar. Black-box side-channel attacks highlight the importance of countermeasures - an analysis of the xilinx virtex-4 and virtex-5 bitstream encryption mechanism. In Orr Dunkelman, editor, *Topics in Cryptology - CT-RSA 2012 - The Cryptographers' Track at the RSA Conference 2012, San Francisco, CA, USA, February 27 - March 2, 2012. Proceedings*, volume 7178 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2012. doi: 10.1007/978-3-642-27954-6_1. URL https://doi.org/10.1007/978-3-642-27954-6_1.

[31] National Institute of Standards and Technology (NIST). Advanced encryption standard (aes). *FIPS-197*, 2001.

[32] National Institute of Standards and Technology. Advanced encryption standard. *NIST FIPS PUB 197*, 2001.

[33] Christof Paar and Jan Pelzl. *Understanding Cryptography - A Textbook for Students and Practitioners*. Springer, 2010. ISBN 978-3-642-04100-6. doi: 10.1007/978-3-642-04101-3. URL https://doi.org/10.1007/978-3-642-04101-3.

[34] Guilherme Perin, Baris Ege, and Jasper van Woudenberg. Lowering the bar: Deep learning for side channel analysis. 2018.

[35] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. 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. doi: 10.13154/tches.v2020.i4.337-364. URL https://tches.iacr.org/index.php/TCHES/article/view/8686.

[36] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. 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. doi: 10.13154/tches.v2019.i1.209-237. URL https://tches.iacr.org/index.php/TCHES/article/view/7339.

[37] Mathieu Renauld and François-Xavier Standaert. Algebraic side-channel attacks. In *International Conference on Information Security and Cryptology*, pages 393–410. Springer, 2009.

[38] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the aes: Why time also matters in dpa. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 97–111. Springer, 2009.

[39] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(3):677–707, Jul. 2021. doi: 10.46586/tches.v2021.i3.677-707. URL https://tches.iacr.org/index.php/TCHES/article/view/8989.

[40] Phillip Rogaway. Security of cbc ciphersuites in ssl/tls: Problems and countermeasures, 2002. URL http://www.openssl.org/~bodo/tls-cbc.txt.

[41] Sayandeep Saha, Arnab Bag, Debapriya Basu Roy, Sikhar Patranabis, and Debdeep Mukhopadhyay. 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*, volume 12105 of *Lecture Notes in Computer Science*. Springer, 2020. doi: 10.1007/978-3-030-45721-1_22.

[42] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005. doi: 10.1007/11545262_3. URL https://doi.org/10.1007/11545262_3.

[43] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[44] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 443–461, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-01001-9.

[45] Stefan Tillich, Christoph Herbst, and Stefan Mangard. Protecting aes software implementations on 32-bit processors against power analysis. In *International Conference on Applied Cryptography and Network Security*, pages 141–157. Springer, 2007.

[46] Nicolas Veyrat-Charvillon, Benoît Gérard, and François-Xavier Standaert. Soft analytical side-channel attacks. In Palash Sarkar and Tetsu Iwata, editors, *Advances in Cryptology – ASIACRYPT 2014*, pages 282–296, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-45611-8.

[47] Huanyu Wang and Elena Dubrova. Tandem deep learning side-channel attack against fpga implementation of aes. *IACR Cryptol. ePrint Arch.*, 2020:373, 2020.

[48] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):147–168, Jun. 2020. doi: 10.13154/tches.v2020.i3.147-168. URL https://tches.iacr.org/index.php/TCHES/article/view/8586.

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

A

Deep Learning Model used for Round 3 Sbox

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv1d (Conv1D)              (None, 2960, 64)          768
_________________________________________________________________
average_pooling1d (AveragePo (None, 1480, 64)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 1480, 128)         90240
_________________________________________________________________
average_pooling1d_1 (Average (None, 740, 128)          0
_________________________________________________________________
conv1d_2 (Conv1D)            (None, 740, 256)          360704
_________________________________________________________________
average_pooling1d_2 (Average (None, 370, 256)          0
_________________________________________________________________
conv1d_3 (Conv1D)            (None, 370, 512)          1442304
_________________________________________________________________
average_pooling1d_3 (Average (None, 185, 512)          0
_________________________________________________________________
conv1d_4 (Conv1D)            (None, 185, 512)          2884096
_________________________________________________________________
average_pooling1d_4 (Average (None, 92, 512)           0
_________________________________________________________________
flatten (Flatten)            (None, 47104)             0
_________________________________________________________________
dense (Dense)                (None, 4096)              192942080
_________________________________________________________________
dense_1 (Dense)              (None, 4096)              16781312
_________________________________________________________________
dense_2 (Dense)              (None, 9)                 36873
=================================================================
Total params: 214,538,377
Trainable params: 214,538,377
Non-trainable params: 0
_________________________________________________________________
```

B

Alignment of Traces

The Figure B.1 shows three consecutive traces from our dataset. The Misaligned traces are the measurements captured by default and the Aligned traces are the measurements obtained after the alignment operation. The window of samples shown are the ones used while attacking a byte after round 3 S-box.

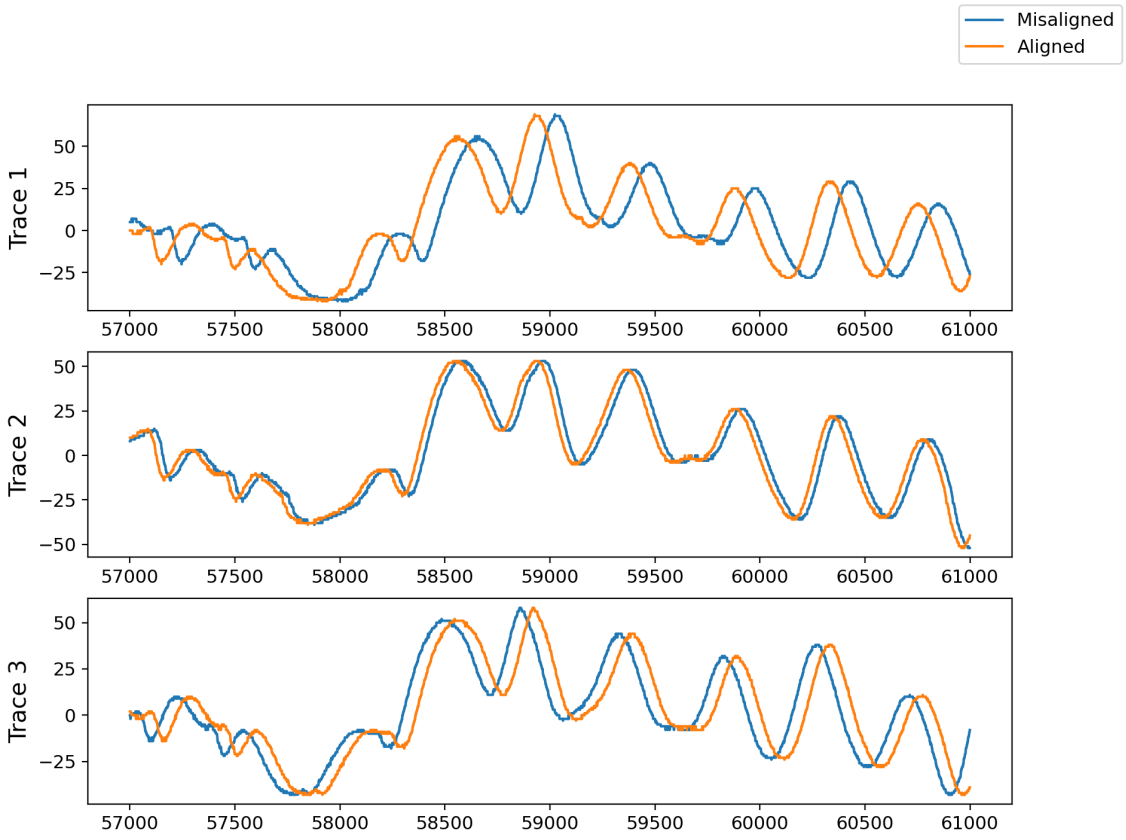

Figure B.1: An example of three consecutive traces from our dataset before and after alignment.

C

DL-SCA Results using Random Models

Figure C.1 shows results with 10 randomly chosen Deep Learning architectures while attacking round 2 with aligned traces. Figure C.2 shows results of another 10 randomly generated models while attacking round 2 with added Gaussian noise.

Figure C.3 show the attack results of 10 random models on aligned traces by targeting a byte after round 3 S-box. Figure C.4 shows the result of attacking round 3 S-box on misaligned traces while Figure C.5 shows the result of attacking the same but with both Gaussian noise and misalignment.

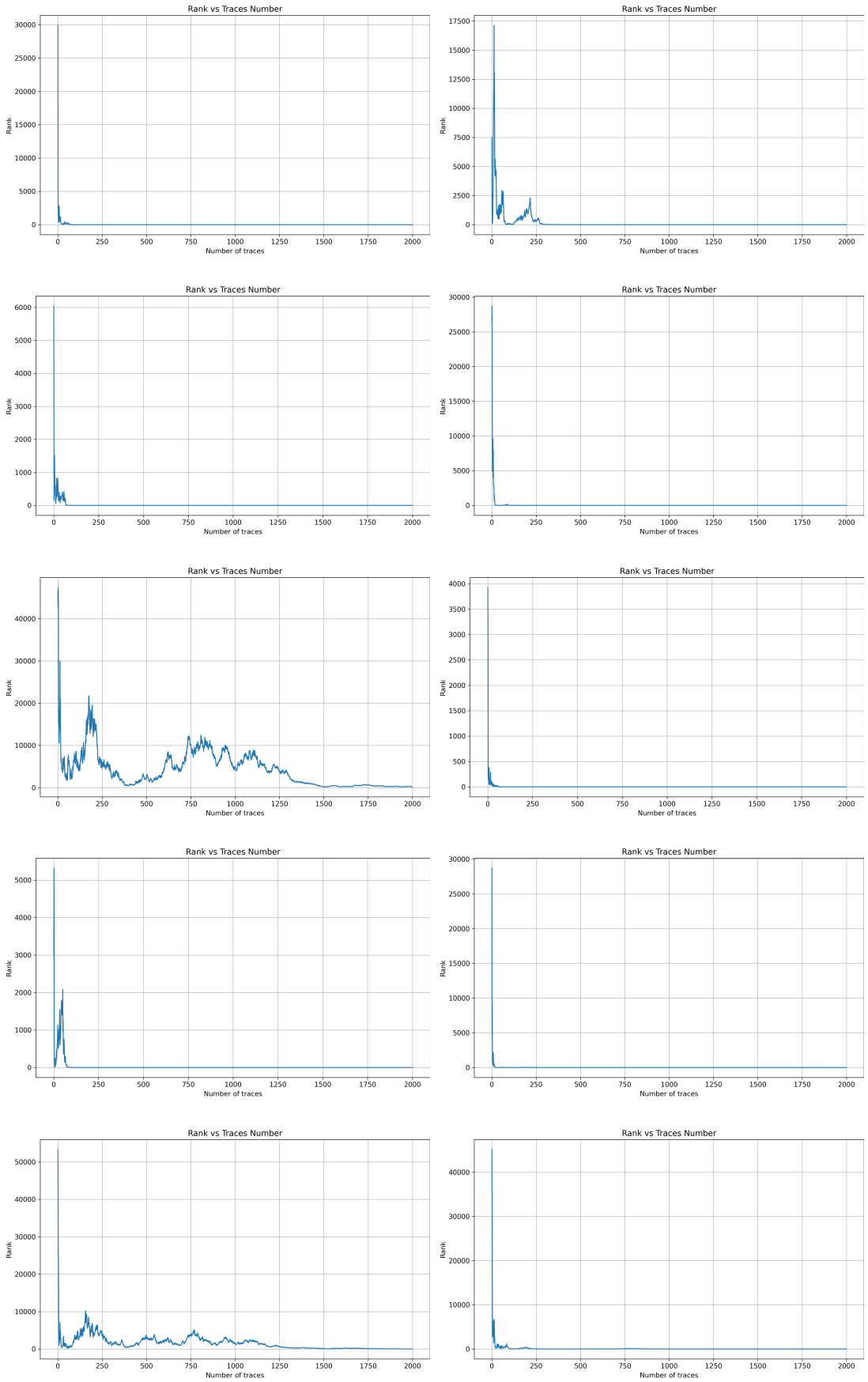

Figure C.1: DL-SCA using random models on aligned traces for round 2.

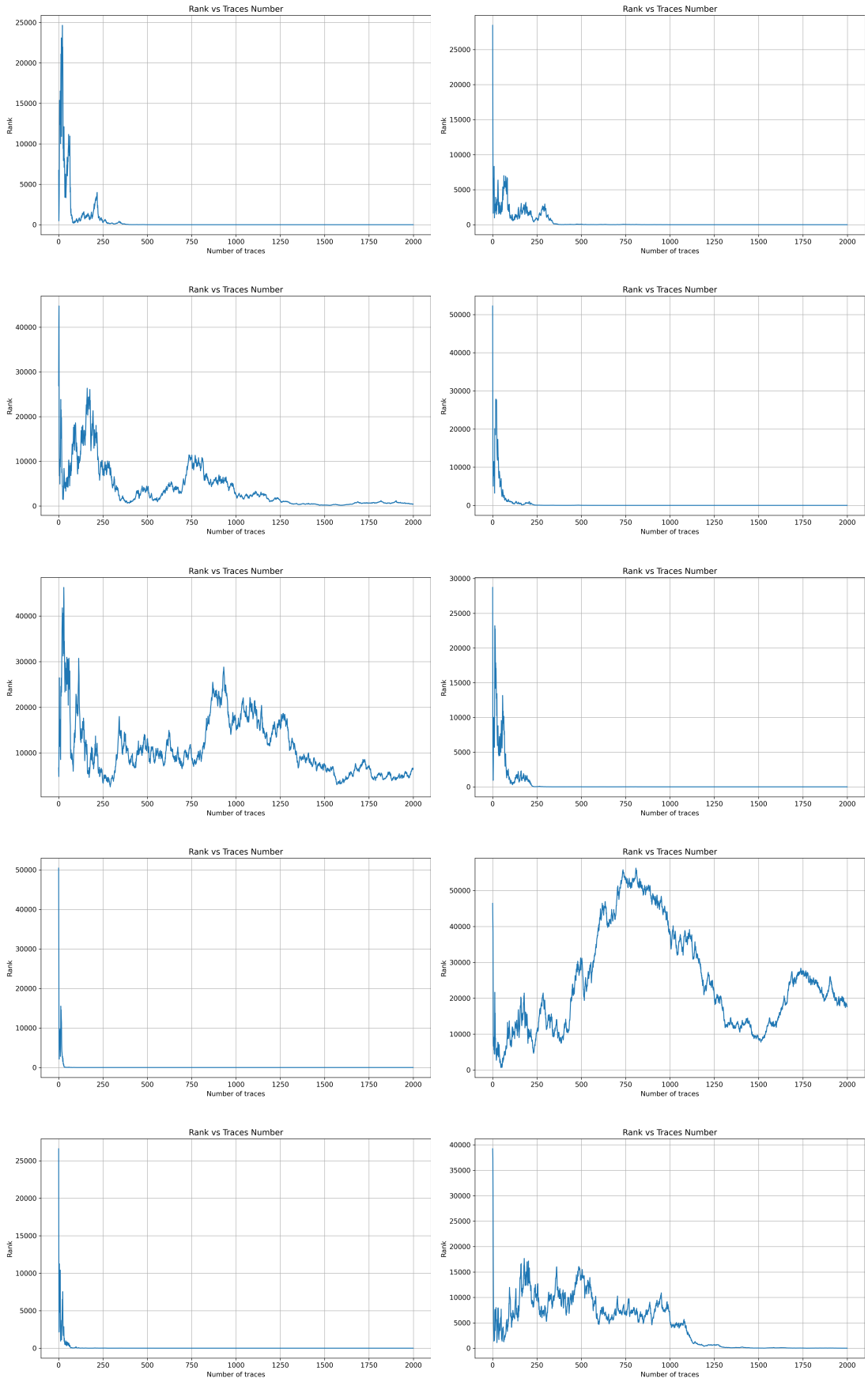

Figure C.2: DL-SCA using random models on traces with gaussian noise for round 2.

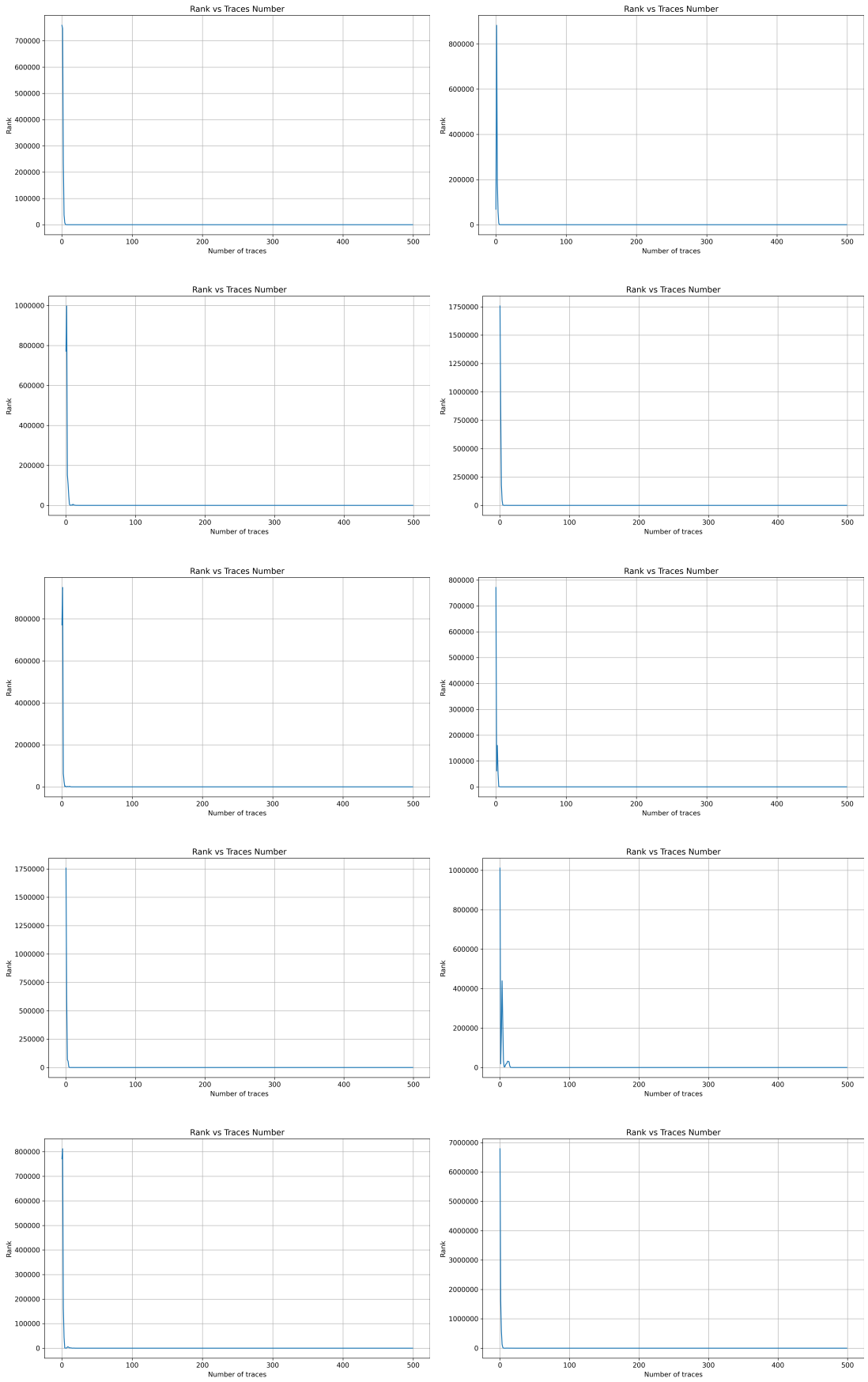

Figure C.3: DL-SCA using random models on aligned traces for round 3.

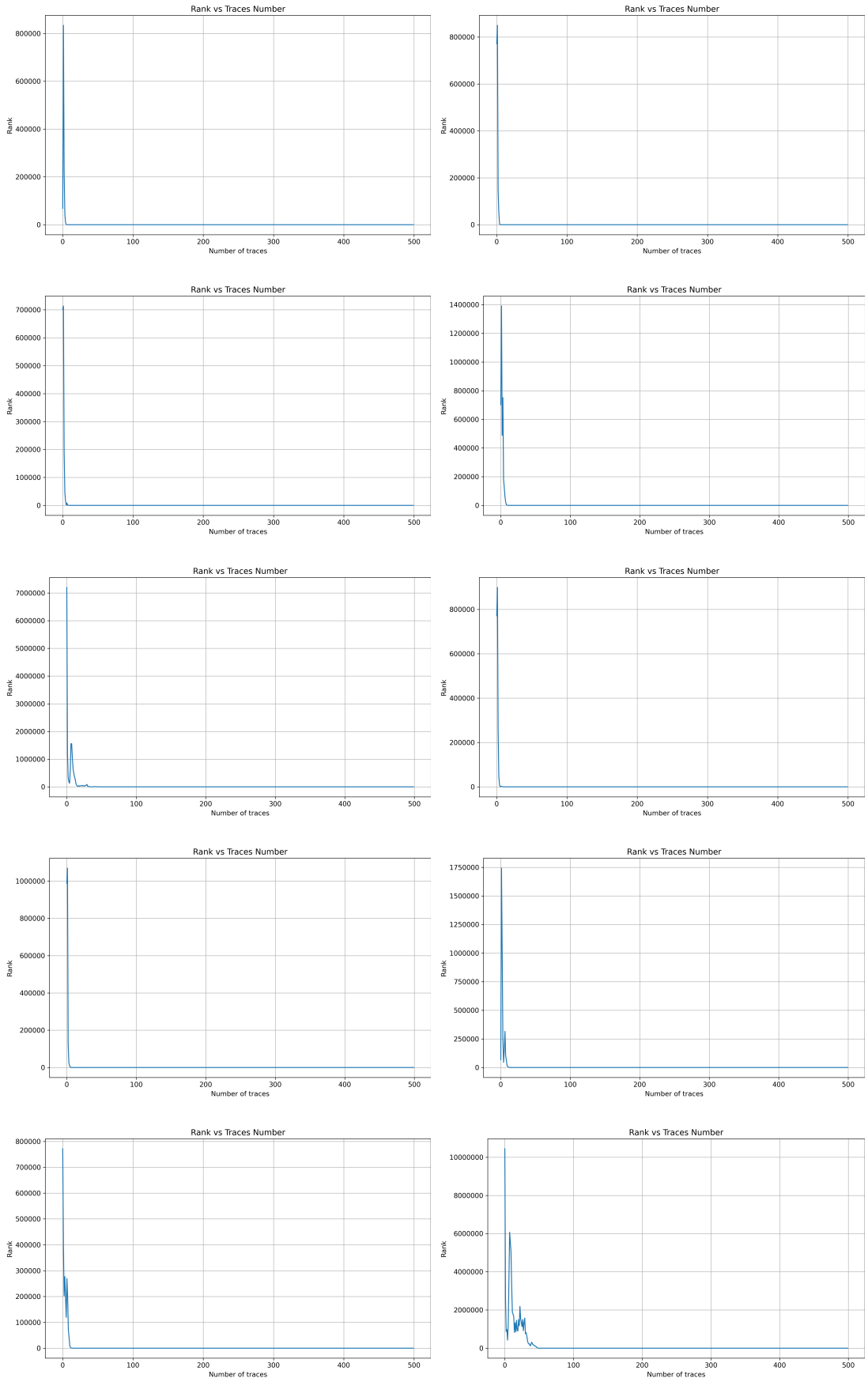

Figure C.4: DL-SCA using random models on misaligned traces for round 3.

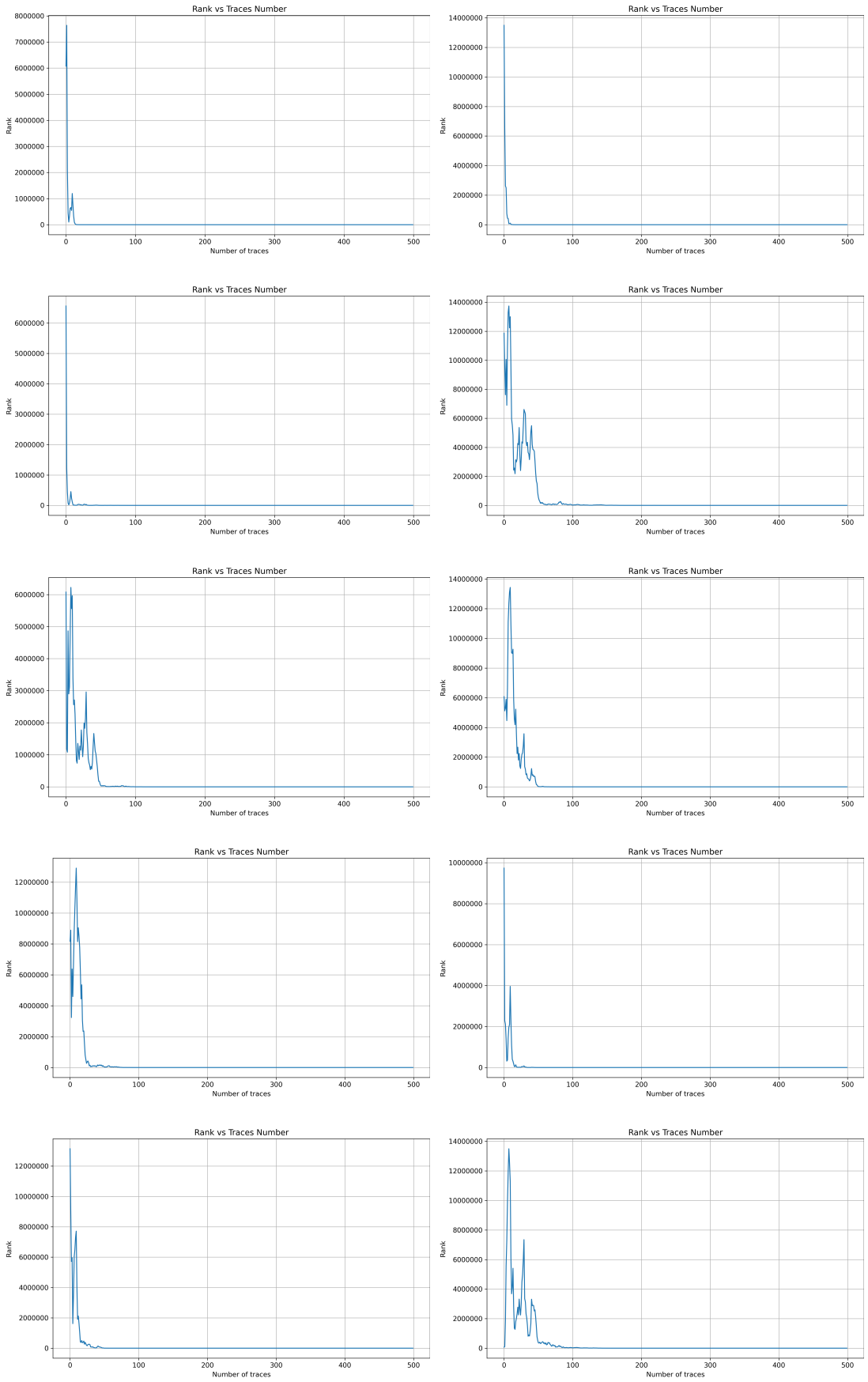

Figure C.5: DL-SCA using random models on misaligned traces with Gaussian noise for round 3.