Hardware-based implementations in Side-Channel Analysis

A comparison study of DL SCA attacks against HW and SW AES and a novel methodology

Wolfgang Bubberman

Delft University of Technology

TU Delft

Hardware-based implementations in Side-Channel Analysis

A comparison study of DL SCA attacks against HW and SW AES and a novel methodology

by

Wolfgang Bubberman

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on August 24th, 2022 at 14:00.

Student number:	4704673	
Project duration:	November 22nd, 2021 – August 24th, 2022	
Thesis committee:	Dr. ir. S. Picek,	TU Delft, supervisor
	Prof. dr. ir. I. Lagendijk,	TU Delft
	Dr. E. Isufi,	TU Delft

This thesis is confidential and cannot be made public until August 24th, 2022.

TU Delft

Abstract

Side-Channel Attacks (SCA) attempt to recover the secret cryptographic key from an electronic device by exploiting the unintended physical leakages of said device. With the devices that are being attacked becoming more sophisticated, so is SCA. In the past few years, the focus of the research in the field of SCA shifted towards the application of the powerful method of Deep Learning (DL). DL SCA methods can operate in a similar way as before seen methods in the profiled setting, such as Template Attack. In the profiled setting, SCA first creates a profile on a copy of the target device, to then subsequently use that same profile to perform a more powerful attack on the target device. DL SCA has proven to be quite the effective method, even showcasing its success against implementations utilising countermeasures. However, as DL SCA is fairly novel there still exist gaps in the knowledge we have of how to make DL SCA effective in all possible situations. Most research bases itself on software-based implementations, whilst rarely hardware-based implementations are discussed as they are often seen as more difficult to attack. Our contribution in this work is to showcase the difference in difficulty between hardware-based implementations and software-based implementations that both use countermeasures. We explore the attack performance of several state-of-the-art methods on hardware-based implementations with countermeasures and give insight into why their performance is the way it is. We also attempt to make a base methodology for attacking hardware-based implementations, both with and without countermeasures, as the current field of research is lacking this. Showcasing our suggested methodology, we achieve better than state-of-the-art results on a hardware-based implementation with countermeasures and competitive with the state-of-the-art results on a hardware-based implementation without countermeasures.

Wolfgang Bubberman
Delft, August 2022

Preface

First and foremost, I would like to thank Stjepan, my supervisor. The weekly meetings we had and his sagacious insights made all of this possible. Thank you for putting up with my stupid questions and prodding me in the right direction at the right times. I would also like to thank Sengim, who has been a partner throughout my studies of Computer Science, both in the bachelor and the master. The days and nights we worked together on courses, projects and finally our own theses were a pleasure to have been spent with you. You have taught me more than you think you did and encouraged me to better at the worst of times. Most of all you have been an amazing friend. Last, and certainly not least, I would like to thank my friends, my mother and my girlfriend for being interested in my work, and putting up with my rambles about Side-Channel Analysis whilst not understanding a single word I said due to my somewhat chaotic way of explaining things. The work on this thesis was at times hard but rewarding, the many struggles I had gave me lessons and experiences I am grateful for. Doing this work forced me to take a step back and realize that I can have it all, but at the cost of many things I value indirectly. This all-in-all was a life changing event and I hope it to be a closure of one of the most tempestuous chapters of my life, and the beginning of the next chapter. I would like to close this preface by saying that I am saddened by the fact my father cannot see how far I have made it due to his untimely departure of this life, but I am hopeful that he would have been proud of me.

Wolfgang Bubberman
Delft, August 2022

Contents

List of Figures

List of Tables

1 Introduction

In the past few years, the prevalence of security measures in cyberspace has increased dramatically [22, 49]. One of the most used forms of security measurements is the usage of encryption. Encryption is used in small embedded devices and on the internet to secure information transport. Encryption algorithms, like AES and RSA, are theoretically secure [37], as their keys cannot be derived from just the input and output of the devices using them. However, due to the leakages coming from the devices using these encryption algorithms [1, 41], this guarantee of security might not be as strong as one might think.

Making use of Side-Channel Attacks (SCA), attackers can use the physical leakages to derive leaked information during the execution of the method of encryption. The hardware executing the computations needed for the cryptography generates these physical leakages and the leakages can exist in many different forms. Forms such as: electromagnetic (EM) emanation [48], power consumption [33], sound [3], cache-timings [66], or even heat [27]. Many different forms of SCA exist; since the first Differential Power Analysis (DPA) by Kocher *et al.* [33], we have seen the Correlation Power Analysis (CPA) [6], Template Attack (TA) [11] and many more.

While DPA and CPA are direct attacks, and therefore directly attack the physical leakages to find some statistical connection between the traces and the sensitive values generated by a secret key, there also exist profiled (non-direct) attacks, of which the Template Attack was the first [12]. When using profiled SCA, the attacker has a copy of the device they want to attack and can therefore model the expected behaviour of the target device. This expected behaviour model can then be used to gain additional information, which subsequently leads to a significantly stronger attack than DPA or CPA can provide with their direct attack.

With the evolution of the different forms of SCA, the encryption side of things evolved as well. Countermeasures were introduced to impede the SCA. These countermeasures came in the forms of hiding countermeasures, which hide important features in the physical leakages, and masking countermeasures, which attempt to hide the correlation of the physical leakage with the intermediate values. To overcome these introduced countermeasures, the usage of machine learning emerged in the domain of SCA [30, 36, 44, 47, 48, 51, 67]. Machine Learning proved to be effective at overcoming the countermeasures, and especially Deep Learning (DL) has seen widespread application and success in the past few years [36]. An advantage of using DL is that there is less of a need for feature engineering and feature selection, as the nature of DL allows it to deal with countermeasures automatically.

When SCA first came around, the field struggled with finding the right metrics to use for Deep Learning, as standard metrics were found to be misleading for the domain of Side-Channel Analysis [46]. However, that has changed, and several different metrics have been developed specifically for Side-Channel Analysis that are proven to be effective. Namely, the Success Rate (SR) and Guessing Entropy (GE) metrics are commonly used within the domain now with widespread success.

Several approaches and even methodologies [23, 67] regarding developing Deep Learning methods for software-based implementations in the domain of Side-Channel Analysis have been proposed. However, there is not a straightforward approach available for their hardware-based counterparts. It is uncertain if the same approaches are as practical on hardware-based implementations as on software-based implementations, and papers that compare both types of implementations when employing countermeasures do not exist. This lack of comparison makes it unclear if hardware-based implementations are harder to attack or whether current methods are tailored to software-based implementations, as they are more prevalent in the literature.

Looking into the differences between these two types of encryption implementations and the effectiveness of current state-of-the-art approaches could give critical insights into the subject of how we approach creating models for attacking specific implementations. Analyzing this further could help us understand why hardware-based implementations are often seen as more challenging to attack and allow us to construct our own methodology of approaching said hardware-based implementations with more success than ever before.

To be able to compare the hardware-based implementations to their software-based counterparts, we formulate our first research question as the following:

What is the difference in the attack performance between software-based implementations and hardware-based implementations?

From this, we want to establish an approach on how to attack and analyze hardware-based implementations successfully. Therefore, we formulated our second research question as:

Which novel design elements can we introduce to improve the attack performance on higher-order hardware-based data sets?

We focused on higher-order protected data sets for our second question. We believe that if we can successfully attack data sets that include some form of countermeasures, we are likely to be effective in attacking data sets that do not.

After discussing some needed background material on Deep Learning, Cryptography, Side-Channel Analysis, and the data sets used throughout this work in chapter 2, we will discuss the related work in chapter 3. We restate our research questions and propose our sub-questions in chapter 3 to aid in answering the main research questions. Following these sections, we will showcase our work in answering our first research question in chapter 4. Then we continue with the proposal of our novel attack method based on the conclusions found in chapter 4 in chapter 5. Finally, we answer our research questions in the last chapter, chapter 6. In this final chapter, we also aim to address the limitations of our work and options for future research.

2

Background

This chapter aims to introduce and explain the background principles of the research presented in this work. First, we will discuss Deep Learning, what it is, how we train Deep Learning networks, and what kind of Deep Learning networks are there. After that, we will discuss cryptography. This section is critical for understanding what kind of environment everything is taking place in. Then, we will move on to Side-Channel Analysis and, finally, the data sets used within this work.

2.1. Deep Learning

Deep Learning (DL) is a form of Machine Learning (ML) that uses multiple layers of neurons, which are interconnected, creating a network to model underlying patterns in a data set. These neural networks that DL creates, attempt to simulate the way brains work and learn from the data inserted into the network. These types of networks allow DL to work with unstructured data, as opposed to ML, which needs to know the hierarchy of the features in the data inserted into it [20].

In the past few years, the field of Deep Learning has seen many leaps in its adoption and development. It has seen adoption in many different fields, fields such as image classification [35, 43], speech recognition [15, 68] and even medical diagnosis [21, 57]. DL has shown that it can be a powerful tool for classification tasks.

Classification with DL is done using a supervised learning environment. A supervised learning environment is an environment where the network has to label or classify unlabeled data based on a set of already labelled data [9]. In the domain of SCA, the labelled data is often called the profiling set, and the unlabeled data the attack set. This is as we profile our network with the labelled data and “attack” the unlabeled data by trying to label it using the network.

2.1.1. Training Deep Learning Networks

As hinted above, we need to train our DL network to classify incoming data. We do this by training the DL network for a certain amount of **epochs**, which is a frame of time used to describe the amount of time needed for the DL network to process all profiling data once. All the data processed by the network is split up in **batches** of equal size. The splitting in batches is done so that the network can make slight modifications in its classification while it is training. How to modify itself, the network is based upon comparing the predictions of a processed batch with the data labels in that same batch. This comparison is made with a **loss function**,

and this loss function describes how good or bad the network is performing in its predictions. During training, the network tries to minimize the loss found by this loss function. The network does this by finding the **gradient** of the loss function. Using an **optimizer**, the network can then update its weights to the correct amount based on this gradient. The network also controls how significant these updates to the weights based on the gradient are via the **learning rate** of the network. A higher learning rate means that the network would update its weights more than the network would with a lower learning rate.

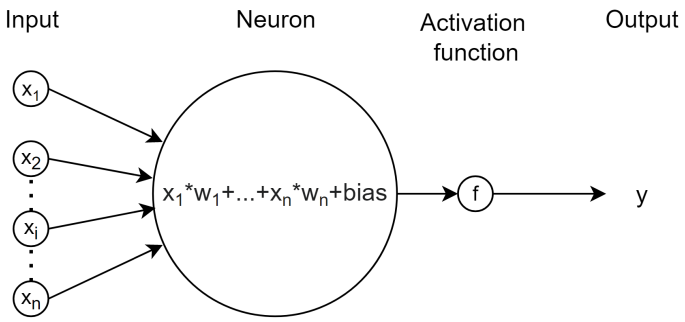

Figure 2.1: An illustration of a neuron in a Deep Learning network.

There exist several different options for the choice of optimizer, examples are Stochastic Gradient Descent (SGD) [5], RMSprop [14], and Adam [31]. SGD is the first known optimizer, simply updating the weights in the direction of the gradient. RMSprop and Adam are more modern optimizers and employ more advanced techniques such as the decaying moving average and adaptive learning rate. These optimizations of the optimizer lead to faster convergence and are therefore more popular in their usage.

What are we training? As quickly mentioned above, we train the neurons' weights in our network. These neurons take the sum of their inputs, multiply these inputs with the respective weights of each input, add a bias and then pass all that to an activation function to produce its output for the next layer. Figure 2.1 showcases this process, and it can be described as a function as well:

$$OutputNeuron(x_1, x_2, \ldots, x_n) = ActivationFunction(x_1 * w_1 + x_2 * w_2 + \ldots + x_n * w_n + bias) \tag{2.1}$$

Here x_i describes the input and w_i the weights of each respective input of the neuron.

As the input of an activation function is a linear combination of the inputs, weights, and the bias, and we might have non-linear patterns in the data we are training with, activation functions come in many different, often non-linear, forms. Four activation functions used throughout this work are:

ReLU [19]: one of the most straightforward activation functions, it takes the input or 0 if the input is smaller than 0.

$$relu(x) = max(0, x) \tag{2.2}$$

Tanh [56]: the Hyperbolic Tangent function which is the hyperbolic analogue of the Tan circular function used throughout trigonometry. The output of this activation function will always be

between [-1, 1].

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

SELU [32]: here the scale (λ) and alpha are predefined most of the time as 1.05070098 and 1.67326324 respectively, but they can be inferred from the input data.

$$selu(x) = \begin{cases} \lambda * x, & \text{if } x \geq 0 \\ \lambda * \alpha(exp(x) - 1), & \text{if } x < 0 \end{cases} \tag{2.4}$$

SoftMax [17]: is an activation function often used for the output neurons of the network. All its outputs are between and including [0, 1] and sum to 1. Here n represents the amount of neurons within in the layer.

$$softmax(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \text{ for i = 1, 2, ..., } n \text{ and } x = (x_1, x_2, ..., x_n) \in \mathbb{R}^n \tag{2.5}$$

2.1.2. Deep Learning Architecture Types

There are different DL architecture types, and in this subsection, we will discuss the three main architecture types used within this work.

Multilayer Perceptron

Multilayer Perceptrons (MLPs) is a network that consists of an input layer, a certain amount of hidden layers, and an output layer [18]. Every layer is a **fully connected layer**, which means that every neuron in each layer connects to every other neuron of the previous layer. The number of neurons in the first layer of an MLP is the number of features of the input data. Similarly, the amount of neurons in the output layer is decided. However, this time the amount of neurons is based on the number of different classes we want our MLP to distinguish. This simplistic design makes the MLP a popular choice in the field of DL [8] and relatively easy to implement. When implementing an MLP for a certain classification task, the main two hyper-parameters varied and played with are the number of hidden layers and the activation function used within these aforementioned hidden layers. See Figure 2.2 for a depiction of an MLP.

Convolutional Neural Networks

However, fully connected layers are not the only layers one can use when creating a network that uses DL. **Convolutional layers** are a common type of layer used within the field of DL [2], and when a network uses at least one of these layers, we call the network a Convolutional Neural Network (CNN). These convolutional layers are composed of filters or kernels, which are a lot like neurons as they have sets of weights for specific inputs, but they only operate on a subset of the input each time. The larger the kernel size, the larger the input subset will be. The stride of the convolutional layer describes how many data points are between the application of each filter and, in turn, creates a new feature map every time. An example of such a layer can be seen in Figure 2.3.

Another commonly used layer used within CNNs is a **pooling layer**. These layers reduce the input size for the next layer as they do not learn any parameters. These pooling layers divide the input into overlapping subsets and reduce each of these subsets into a single value. This reduction is often made by either averaging the subsets, called average pooling or taking the maximum of each subset, called max pooling. These pooling layers are often used right after

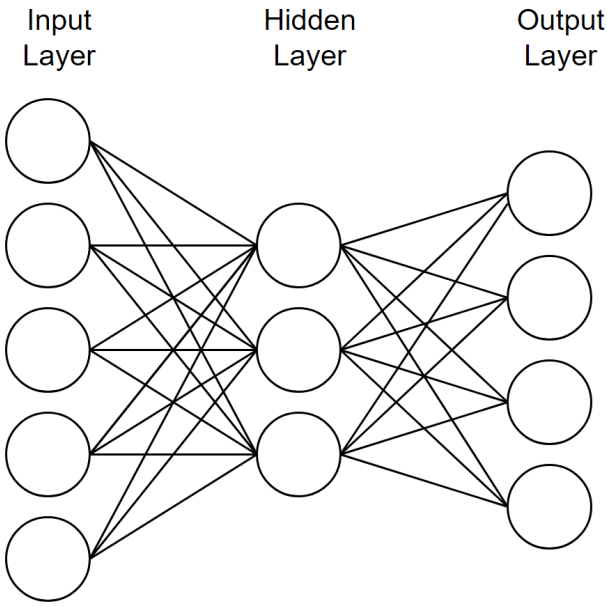

Figure 2.2: An illustration of an MLP with an input layer with 6 features, one hidden layer consisting of 3 neurons and an output layer of 4 classes.

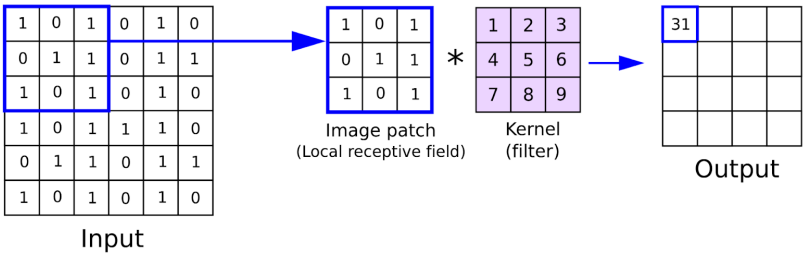

Figure 2.3: An illustration of a convolutional layer with a kernel size of 3 by 3 and a stride of 1.

a convolutional layer to stabilize the output generated by the convolutional layers.

An example of pooling layers can be found in Figure 2.4. An interesting type of pooling layer that is sometimes used is the global pooling layer. This pooling layer takes the average or maximum of the entire feature map and returns that as a singular output.

Residual Neural Networks

This work will also discuss a particular type of CNN, a Residual Neural Network (ResNet). These networks are constructed similarly to a CNN but significantly deeper. When a CNN gets very deep, the effect of the gradient vanishing problem becomes too large, and the weights in the layers at the beginning of the network struggle to update accordingly [25]. ResNets aim to resolve this problem by using shortcuts throughout the network to skip layers in the network and propagate the gradient. They do this by dividing the network's structure into residual blocks, which are blocks made up of several convolutional layers. Then to the output of this block, a shortcut is added from the output from a previous block, which helps alleviate the gradient vanishing problem. An example of how this looks can be found in Figure 2.5.

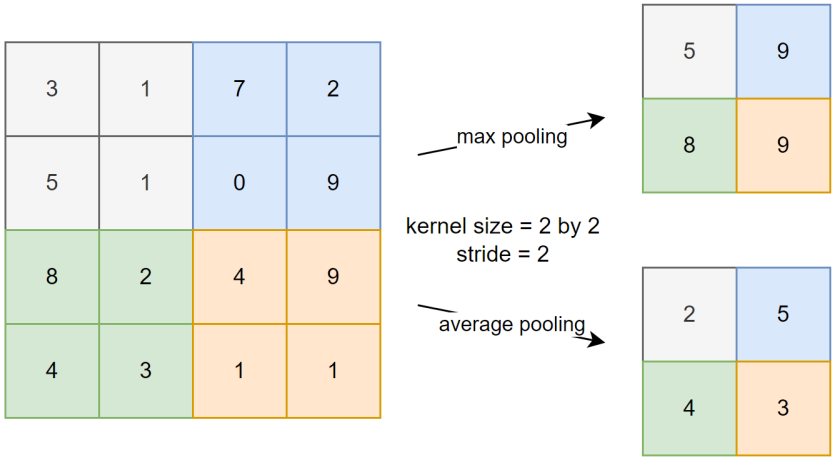

Figure 2.4: An illustration of a max pooling layer and an average pooling layer, both with a kernel size of 2 by 2 and a stride of 2.

2.2. Cryptography

In this section, we will discuss the specifics of cryptography. We will also give an overview of the Advanced Cryptography Standard (AES). It is the type of cryptographic algorithm used in this work and the most common symmetric-key algorithm. To conclude this section, we will discuss the countermeasures as they are a vital factor in some of our research.

Cryptography aims to construct protocols that can still accomplish the task of conveying information even in the presence of an adversary. This is done by **encrypting** the input, or **plaintext**, of the cryptographic algorithm with an **encryption key**. This process of encryption results in a **ciphertext** which later can be **decrypted** by using the same encryption key in the case of a symmetric-key encryption algorithm or a different key in the case of a symmetric-key encryption algorithm. Applying this to a setting where an adversary is present, the adversary cannot get to the data of the input after the input has been encrypted, therefore enabling confidential communication between two points. Cryptographic algorithms have another distinction as well: there exist block ciphers, which operate on blocks of input data, and stream ciphers, which operate on individual bits of data.

2.2.1. Advanced Cryptography Standard

The Advanced Encryption Standard (AES) [42] is the replacement of the Data Encryption Standard (DES), which was deemed unsafe due to its relatively short 56-bit key size [54], and comes in a variety of forms. All AES implementations are symmetric-key block ciphers operating on blocks of 128 bits. There exist three main variations: the 128-, 192-, and 256-bit key variants, where the variants take up 10, 12, and 14 rounds in total, respectively. These rounds are made up of certain operations acting on an internal state of a four-by-four grid, with each cell being 1 byte or 8 bits and thus totalling 128 bits. An overview of the AES versions discussed can be seen in Table 2.1.

2.2.2. Countermeasures

With people attempting to break cryptography methods, for example, using Side-Channel Analysis, which will be covered in the next section, countermeasures are developed to counteract these attacks. Many different types of countermeasures can be used in cryptography. However, we will discuss only **masking** and **hiding countermeasures**, as we have found these to be the most prevalent in our work and the field of Side-Channel Analysis.

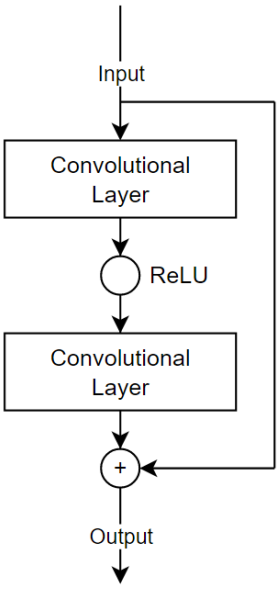

Figure 2.5: An illustration of a residual block with 2 convolutional layers, and a ReLU activation layer.

Operation	Description	Number of rounds
AddRoundKey	Internal state ⊕ round key 1.	Not applicable.
SubBytes	Substitute each internal state value in the Sbox.	9, 11, or 13 rounds.
ShiftRows	Shift the rows of the internal state.	
MixColumns	Mix the columns of the internal state.	
AddRoundKey	Internal state ⊕ current round key.	
SubBytes	Substitute each internal state value in the Sbox.	1 round.
ShiftRows	Shift the rows of the internal state.	
AddRoundKey	Internal state ⊕ last round key.	

Table 2.1: Overview of the structure of the AES symmetric block cipher. The Sbox is a non-linear substitution table used in several byte substitution transformations and in the Key Expansion routine to perform a one-for-one substitution of a byte value.

Masking countermeasures

Masking countermeasures attempt to hide the correlation of the physical leakage with the intermediate values by applying a mask to these intermediate values in our cryptographic algorithms.

When applying a masking countermeasure, we use the XOR operation with the output of the Sbox operation and a mask to get the new, masked output of the Sbox. This applying of a mask can be done multiple times to achieve higher-order masking but requires specific masking values to be effective [53, 55]. We can write an example of first-order Boolean Masking out as:

$$Sbox = Sbox[k_i \oplus input_i] \oplus mask_i \tag{2.6}$$

where k_i is the round key, $input_i$ is the input for this round, $mask_i$ is the mask for the input of this round and i is the round number.

There also exists arithmetic masking, which does not use the XOR operation, but the arithmetic

modulo operation. However, this is less prevalent in use, and we are not aware of any usage of arithmetic masking within the data sets that we used in this work.

Hiding countermeasures
Hiding countermeasures are countermeasures that try to hide interesting features in the physical leakage completely by either creating random or constant noise in the physical leakage to prevent correlation with the intermediate values. Many forms of hiding countermeasures exist, and therefore we will not discuss all of them. However, the most interesting and applicable ones are the desynchronization, additive noise, and the clock jitter hiding countermeasures.

With the desynchronization hiding countermeasure, the alignment of the traces is disturbed. This alignment of the traces is critical for an adversary. It allows them to distinguish patterns in the traces and find where the physical leakage operation occurs, which is vital information for figuring out the secret key. We can mess with the alignment and make the adversary's job significantly harder by waiting for a random amount of time before certain sensitive operations. Applying this countermeasure with other countermeasures is a good idea and could be done without any interference.

By applying dummy operations or using a component that generates much noise, we can utilize the countermeasure of additive noise. This extra noise complicates an adversary's attack by making it harder to find a correlation between the physical leakage and the intermediate values due to the amount of noise in the traces. Therefore, the Signal-To-Noise (SNR) ratio is lowered by this countermeasure and can be pretty effective in thwarting an adversary. Note that this countermeasure is also applied a posteriori and can either be done by using Gaussian noise or a uniform distribution. The former is a more realistic simulation, and the latter is easier to apply. We do not make use of this in our work, but it is an important countermeasure to know.

The addition of a clock jitter to the traces is often seen as a bad thing in many implementations, but it is not in the field of SCA. This is due to the introduction of randomness in the time domain of the traces [7]. Unlike the desynchronization countermeasure, it does not disturb the alignment of traces globally but does this locally. Therefore, realigning the traces after introducing a clock jitter makes it so that an adversary would not have enough information to create a connection between the physical leakages and the intermediate values. A clock jitter is simulated by randomly adding and removing features within the traces and cannot be applied a posteriori, unlike the desynchronization mentioned above and additive noise countermeasures.

It should be noted that hiding countermeasures are not always intentionally generated by cryptography designers. There are cases where hiding countermeasures simply happen because the device is extremely low power, which leads to noise in the amplitude domain. It could also happen that the acquisition of SCA traces has difficulties to measure a window that always starts at the same time, which leads to desynchronization.

2.3. Side-Channel Analysis

Going over all possible key values for the AES 128-bit version, we have 2^{128} possible key options. Using an implementation of attack that only looks at the input and output of AES, this can be brought down to 2^{126} [60]. However, this still is too large a search space. Luckily, Side-Channel Analysis, or Side-Channel Attacks in the context of an adversary, can bring this search space down dramatically. Side-Channel Analysis is the analysis of the physical

leakages that come from a device. These leakages can come in many different forms, forms such as: electromagnetic (EM) emanation [48], power consumption [33], sound [3], cache-timings [66], or even heat [27]. This section will go deeper into what SCA utilizes to obtain its results, different forms of SCA, and the metrics used within the domain.

2.3.1. Leakage Models

To make use of the physical leakages, SCA creates a leakage model based upon the observations made of the implementation it is attacking. Three of the most commonly used leakage models used by SCA methods are the **hamming weight** (HW), the **hamming distance** (HD) and **identity** (ID) model.

The HW model models the physical leakages in correlation to the hamming weights of the intermediate values dependent on the secret key of the cryptographic algorithm. The hamming weight of a binary number can be defined as the number of bits that are 1 in that particular binary number. The HD model finds the correlation between intermediate value and secret key by using the hamming distance between those respective two values. The hamming distance between two binary numbers can be seen as the number of bits between two numbers that have been flipped. Therefore we can write the hamming distance using a function of the hamming weight of a number:

$$HammingDistance(x_1, x_2) = HammingWeight(x_1 \oplus x_2) \tag{2.7}$$

where x_1 and x_2 are the two numbers we want the hamming distance between of. The identity model is the most straightforward model, was it models the physical leakage to be in direct correlation with the intermediate values.

2.3.2. Non-profiled Attacks

Non-profiled SCA is the most basic form of SCA. In these types of implementations of SCA, the physical leakages of a device are attacked directly, without first building up a profile. These forms are easier to implement due to not needing a copy of the device that needs to be attacked, but they deliver less successful results and need more input data to function. Some examples of non-profiled attacks are Differential Power Analysis (DPA) [33], Correlation Power Analysis (CPA) [6], and Simple Power Analysis (SPA) [33]. SPA is one of the most basic forms of a non-profiled attack, as it exploits the information of the power usage directly and tries to obtain the key this way. SPA often is not strong enough, and DPA can be used to obtain better results, as demonstrated by Kocher *et al.*.

2.3.3. Profiled Attacks

As hinted towards in the former subsection, using a copy of the device that an adversary wants to attack, the adversary can first build up a profile of the device. This first stage is often called the profiling stage or phase, followed by a stage where the intended device is attacked with the usage of the profile obtained in the profiling stage. This second stage is called the attacking stage or phase. The main difference between the versions of profiled attacks is how the first stage is executed and thus how the profile is created.

The first, and most known, profiled attack is the Template Attack (TA) [12, 50]. Template Attack uses the Bayes Theorem in combination with assuming that the physical leakages follow a multivariate Gaussian distribution due to the leakages for consecutive features not being fully independent. While TA has been quite successful in the past, it has been outperformed by machine learning and especially deep learning techniques. Especially when countermeasures are introduced, TA performance seems to drop off in comparison to other more advanced

techniques [30, 36, 44, 47, 48]. TA is also significantly more sensitive to desynchronization, as it requires feature selection beforehand.

The comparison between the classification that machine learning does and the profiling stage of a profiled SCA can quickly be drawn. This is the case, as we can see the profiling stage as a classification problem, where the adversary needs to classify intermediate values based on the leakage traces. This similarity between the two does not mean we can directly apply known machine learning techniques to the domain of SCA, but using the ideas of known machine learning techniques promising results can be obtained [45]. The community has demonstrated several different machine learning techniques, techniques such as Support Vector Machines (SVM) [24, 45] and random forests [45, 46]. One of the main advantages of using machine learning techniques over TA is that often, but not always, significantly fewer traces could be used in the profiling stage to obtain similar results.

One specific branch of machine learning, deep learning, has shown to be the most successful at implementing the profiling stage of profiled SCA. When the complexity of an implementation rose, TA and general machine learning started to struggle. However, the application of deep learning seemed promising due to its nature of filtering out the most important features [34]. Deep learning proved to be successful where other methods were not [30, 36, 47, 64] but did introduce new problems to the field; now the correct hyper-parameters needed to be picked, and the correct architecture of the neural network needed to be constructed to obtain optimal results. Zaid *et al.* introduced a methodology to overcome these new caveats [67], which in turn got some critique by Wouters *et al.* [63]. Kim *et al.* also proposed a novel deep learning approach of using the VGG architecture from the image classification domain for SCA [30] to optimize deep learning in SCA for CNNs.

2.3.4. Side-Channel Metrics

While the machine learning domain has a lot of different metrics to give an idea of how well the classification of a method is performing, these machine learning metrics do not seem to apply that well to the domain of SCA [46]. These machine learning metrics fall short of being successful in the domain SCA because these machine learning metrics only consider the classification of a specific trace, not a group of traces. In SCA, the information from a group of traces is responsible for the secret key of the data set under attack. Another reason is that most of these machine learning metrics only have a positive or negative classification, while in SCA, we tend to rank our guesses in order of success. For these reasons, different metrics have been developed specifically for SCA, two of which are quite popular within the domain, and those are **Guessing Entropy** and **Success Rate**.

Guessing Entropy

One of the ways to measure the vulnerability of encryption against a side-channel analysis (SCA) is the Guessing Entropy (GE), proposed originally by Massey et al. [38]. GE measures the average number of key candidates to test after the side-channel attack [58]. The higher the GE, the more wrong key guesses must be checked before the correct key value is considered. Therefore, GE measures the average computation cost required for a successful side-channel attack and is a good leakage evaluation metric [69].

Guessing Entropy can be defined as:

$$GE(N_a) = \mathbf{E}(g_{S_a}(k^*))$$

where $\mathbf{E}$ is the mean function, N_a is the set of the attack traces, g_{S_a} is a vector of key guesses ordered by their predicted log-likelihood, for a random subset $S_a \subset N_a$, and $g_{S_a}(k^*)$ is the index of the correct key hypothesis k^*. Definition based on Massey *et al.* [38].

Success Rate

Success rate (SR) is also a commonly used metric in SCA [58]. Success rate measures the probability that an attacker guesses the correct key within a certain number of leakage measurements [52]. An intuitive way of assessing the Success Rate is to perform the attack several times and estimate the Success Rate based on this. However, this might be too expensive, both time and computation-wise. Therefore, suggestions have been made for approximations of the Success Rate, for example, the one by Standeart *et al.* [59].

Success Rate can be defined as:

$$SR(N_a) = Pr[g_{S_a}(k^*) = 1]$$

where $Pr[x]$ is the probability of x, N_a is the set of the attack traces, g_{S_a} is a vector of key guesses ordered by their predicted log-likelihood, for a random subset $S_a \subset N_a$, and $g_{S_a}(k^*)$ is the index of the correct key hypothesis k^* [40]. Note that if $g_{S_a}(k^*) = 1$ the attack is successful because the correct key is the highest ranked key.

2.4. Data sets

Throughout the field of SCA, many different data sets are used in academic research. Some are made publicly available such that consistency throughout the literature can exist. The data sets we consider in this work are described in this section and are the following: **ASCAD**, **AES_HD** and **AES_HD_MM**.

2.4.1. ASCAD

The **ASCAD** database [48] consists of several different protected software implementations of AES on an ATMega8515. Most notably **ASCAD** with fixed key and **ASCAD** with random keys. The sensitive value that is attacked for these implementations is the output of the first Sbox and can be described as the following:

$$Leakage(k_i) = SBox(p_i \oplus k_i \oplus r_{in}) \oplus r_{out} \tag{2.8}$$

where i describes the i'th byte of the plaintext (p) or key (k) and r_{in} and r_{out} are the masking schemes.

The fixed key version of **ASCAD** has 50 000 profiling traces, and 10 000 attacking traces where all of these traces were measured with the same key and random plaintexts. The depth of these traces was limited to a pre-selected window of 700 features of the original 100 000 features by the authors to attack the third key byte.

The random key version of **ASCAD** has 200 000 profiling traces, and 100 000 attacking traces where all of these traces were measured with the random keys and random plaintexts. The depth of these traces was limited to a pre-selected window of 1 400 features of the original 250 000 features by the authors to attack the third key byte.

2.4.2. AES_HD

The **AES_HD** data set is an unprotected hardware-based implementation of AES on a Xilinx Virtex-5 FGPA. Due to it being a hardware-based implementation, there is significantly more noise than on software-based implementations like the **ASCAD** implementations.
Throughout the research, different leakage models were considered. A common leakage model of hardware implementations exploits the register update from the last round to output ciphertext. This can be defined as the following:

$$Leakage(C_i, C_j, k^*) = InverseSBox[C_i \oplus k^*] \oplus C_j \tag{2.9}$$

where C_i and C_j denote the two ciphertexts, with i and j describing the inverse ShiftRows operations of AES. For this, $i = 7$ and $j = 11$ are commonly picked when starting from 0, papers like [67], [64] and [28] for example do this. It should be noted that this leakage model can be used for a version of **AES_HD** which only contains traces and labels, but then $i = 15$ and $j = 11$ should be picked, with the key byte being 15 instead of 0.

This data set consists of 100 000 traces with 1 250 features for each trace and can be split however the user likes into a group of profiling and attacking traces.

2.4.3. AES_HD_MM

AES_HD_MM is the protected implementation version of **AES_HD**. It makes use of the multiplicative masking countermeasure and uses the same leakage model as **AES_HD**. The implementation of the data set performed masked AES on a SASEBO-GII FPGA board. As it uses the same leakage model, this means that the data set is a second-order hardware-based implementation [16].

The data set is quite large, with 5 600 000 traces consisting of 3 125 features.

2.4.4. Data set related terms

Here is a rundown of terms you will often see paired with the description of a data set.

Hardware-based implementation: is an implementation of AES which is made with hardware as the name might indicate, this means that no software is used to make the circuit encrypt and decrypt but it does this via interconnected registries and circuits.
Software-based implementation: is an implementation of AES which does make use of software. Of course, a software-based implementation still needs to run on hardware, but via the usage of software, the encryption and decryption can be implemented.
First-order protection: is the protection of AES implementations by having a mask that is XORed with the sensitive values. An example of this is Equation 2.6.
Higher-order protection: applies the same principle as first-order masking schemes, but with more parts to the mask, ensuring attacks have to learn to predict all of these parts of the mask to get to the raw sensitive value. Higher-order protection is therefore more secure.

3

Related work

Since the first time machine learning was applied to the field of Side-Channel Analysis by Hospodar *et al.* [26], the field of SCA has changed into a more deep learning-focused field. Many papers in the field of SCA find novelties and different approaches, and even develop methodologies on how to approach designing a network for SCA [30, 36, 48, 63, 67]. In the past two years, we have seen some interesting papers that produce results that are better than state-of-the-art and introduce approaches which have not been applied to the field of SCA before [51, 64]. However, most papers discuss software-based implementations when introducing their novelty or approach. Papers discussing hardware-based implementations [28, 67], let alone hardware-based implementations that employ countermeasures [62], are few and far between.

3.1. Deep learning in Side-Channel Analysis

The first time deep learning was applied to the field of SCA was in the paper [36] by Maghrebi *et al.* in 2016. Since then, the field has exploded, and many improvements in the application of MLPs and especially CNNs to the field of SCA have been made. One of the significant steps since then was the introduction of the **ASCAD** database by Prouff *et al.* [48]. This data set has been used as a benchmark for varying deep learning methods in SCA. Prouff *et al.* introduced not only the base version of this **ASCAD** data set but also a 50desync and 100desync version, which have a shift of 50 and 100 traces per window frame, respectively, making the attack on the data sets harder than before. Later on, Prouff *et al.* added a variant of **ASCAD** with a variable key, which is used as a benchmark for software-based implementations that have a countermeasure.

Another pillar in the SCA community is the methodology paper by Zaid *et al.* [67]. This methodology suggested ways we can create networks of minimal size for standard public data sets with state-of-the-art performance. At the time, the networks that Zaid *et al.* proposed significantly improved the known results of these data sets mentioned above. With the added benefit of the networks being relatively small, the paper gained much cognizance in the community of SCA. Shortly after the publication of Zaid *et al.*'s, Wouters *et al.* [63] revisited this methodology and made several improvements to it, lowering the trainable parameters significantly and maintaining similar performance by applying pre-processing instead of using convolutional layers with a filter size of one. Wouters *et al.* explain several misconceptions in their paper in detail as well, and with that, they are a notable addition to the knowledge base we have so far in the field of SCA.

In 2018, Picek *et al.* published a deeper look into the impact of the class imbalance inherent in the hamming weight model [46]. They showcased that the trace labels correspond to the hamming weight of the intermediate value under attack. Using SMOTE, Picek *et al.* were able to reduce the amount of needed attack traces to have a successful attack significantly. In this paper, Picek *et al.* also made a showcase on how traditional machine learning metrics can be quite deceptive in the context of SCA and how we can use metrics that are specific to SCA to obtain more reliable, and with that, better results.

3.2. State-of-the-art methods in the deep learning of Side-Channel Analysis

The state-of-the-art for deep learning SCA has evolved significantly over these past few years. Recently, the methodology from Zaid *et al.*, [67] has been surpassed. One of the more interesting new state-of-the-art methods is the AutoSCA paper by Wu *et al.* [64]. In this paper, Wu *et al.* automate the finding of the hyper-parameters using Bayesian Optimisation. Using this automated method, Wu *et al.* find well-performing networks regardless of the data set, leakage model, or neural network type.

Another relatively new approach that achieves state-of-the-art results is in a paper from Rijsdijk *et al.* [51]. In their paper, Rijsdijk *et al.* showcase how one can use the original findings of Baker *et al.* [4] for designing neural network architectures using reinforcement learning in the field of SCA. For this, a few adjustments were made to the work of Baker *et al.*; new, SCA-specific reward functions based on Guessing Entropy were used, and SCA-field-specific assumptions were made. This SCA-field-specificity resulted in minimal networks with state-of-the-art performance seen in the data sets showcased throughout the work.

Recently, we have seen different works pop up that make use of ResNets [28, 29, 39, 70] and have interesting results. However, the work done by Zhou *et al.* is hard to reproduce due to the lack of clarification on how exactly their architecture is constructed and the usage of proprietary data sets. The work done by Masure *et al.* also has some points of critique, as, while it attacks the new data set **ASCAD_v2**, it makes this attack easier by allowing some knowledge of the masks during the profiling phase. On top of this, the results are hard to compare due to this paper being the first to publish results regarding the **ASCAD_v2** data set. The ResNet paper that gained the most traction from the above is the one done by Jin *et al.*. They claim that they can achieve better than state-of-the-art performance due to the usage of an attention mechanism. While the explanation of how they did their work is fairly clear, and we could reproduce their results, a motivation regarding why and when to use their ResNet of choice is unclear. The paper by Karayalçin *et al.* builds upon the work of Jin *et al.*, modifying the residual blocks to minimize the amount of tuning to be done. Their results indicate that ResNets work especially well when the number of profiling traces and features in a trace is large. However, in their work, the usage of hardware-based implementations is missing.

3.3. Lack of research into hardware-based implementations

While there are published results of research into hardware-based implementations, like those of **AES_HD** and **AES_HD_MM**, they are far and few between. For example, the results that Zaid *et al.* published for **AES_HD** might seem promising. However, it later came to light that the results most likely were obtained by using a different version of **AES_HD**, as reproductions showed that the same results were not obtainable using the known leakage model of **AES_HD**

and the version of the data set published by Picek *et al.*.

The paper “Back to the Basics: Seamless Integration of Side-Channel Pre-Processing in Deep Neural Networks” by Won *et al.* [62] is the only paper we are aware of that evaluates the performance of profiled side-channel analysis between hardware-based and software-based implementations which both employ countermeasures. However, this paper leaves several open questions, such as; how does the performance of state-of-the-art networks differ in both types of implementations, why is there a difference in performance, and what can be done to optimize for hardware-based implementations outside of the suggested network. We think the paper does not make the comparison entirely, as the paper is more of a demonstration of their novel design for CNN, the Multi-scale Convolutional Neural Networks (MCNN).

There is currently a piece of research published that performs a complete comparison of hardware-based and software-based implementations. Such research could give critical insights into how we should approach both types of implementation and their key differences. It also could explain the differences in results we see for the two types of implementations. In the next section, section 3.4, we will propose research questions that do aim to give these insights, as mentioned earlier.

3.4. Research Questions

Based on the research done in this chapter, the existing related work, and the field of interest of this paper itself, the following research questions have been formulated:

- **Research question 1:** What is the difference in the attack performance between software-based implementations and hardware-based implementations?
  - Is it possible to break a higher-order hardware-based implementation using deep learning SCA, and what kind of attack performance can we expect?
  - What is the impact on attack performance of the usage of different state-of-the-art models on a hardware-based higher-order implementation?
  - Do the countermeasures we see in hardware-based implementations prove to be more effective in lowering attack performance than the countermeasures we find in software-based implementations?
- **Research question 2:** Which novel design elements can we introduce to improve the attack performance on higher-order hardware-based data sets?
  - Is there a consensus or methodology on approaching higher-order hardware-based data sets?
  - Can we improve the results from state-of-the-art models for higher-order hardware-based data sets with some adjustments, and how can we do this?
  - How does the network size of our novel design element compare to other methods regarding its trainable parameters?

Our first research question will narrow the current gap in knowledge about the difference in attack performance of higher-order software-based implementations and hardware-based implementations. Our second research question will build upon this by giving insight into how we used the obtained information from the first research question to create novel design elements. The basis on which we will answer our first research question will be done in the next chapter, chapter 4. The research into answering our second research question will be done in the

chapter after that, chapter 5. Both research questions and their respective sub-questions will be answered in our concluding chapter, chapter 6.

4

Software-based vs. hardware-based implementations

In this chapter, we review the differences between software-based and hardware-based implementations and how the attack performance between the two differs. First, we look at the current approaches used throughout the literature to attack higher-order implementations and evaluate the effectiveness of those methods. The data sets we specifically use for this are **ASCADr** and **AES_HD_MM**. Then, we look into applying different state-of-the-art methods to a hardware-based implementation with countermeasures and evaluate the attack performance of the different state-of-the-art methods. Finally, we evaluate the effectiveness of countermeasures between software-based and hardware-based implementations and try to conclude if we can deem one of the two more difficult to break.

4.1. Motivation

In chapter 3, we found that while throughout the literature, hardware-based implementations are sometimes evaluated and discussed within the field of profiled side-channel analysis, it is hardly the norm. This part of the field is nearly non-existent. It is an interesting part to evaluate for developing a better methodology regarding how and why we should construct our networks in specific ways. Zaid *et al.* [67] gave practical advice on how to create efficient CNN architectures for profiled attacks by publishing their methodology. However, a critical review by Wouters *et al.* [63] showed that Zaid *et al.* did not always give the optimal solution. On top of this, neither of the two evaluated more complex hardware-based implementations and only limited themselves to the software-based implementations; **ASCAD**, **DPAv4** and **AES_RD**, and the hardware-based implementation **AES_HD**. We consider just **AES_HD** not to be enough regarding analysis of hardware-based implementations, as it is an unprotected implementation, unlike a hardware-based implementation such as **AES_HD_MM**, which uses multiplicative masking. The paper by Won *et al.* [62] does evaluate the performance of profiled side-channel analysis between hardware-based and software-based implementations, which both employ countermeasures, but only as a demonstration of their own novel design element. Therefore, we deemed it necessary to write a comprehensive analysis of the attack performance of hardware-based and software-based implementations, which both employ countermeasures, to illustrate the difference between them and the difficulties of attacking the two different forms of implementation.

4.2. Experimental setup

To properly compare hardware-based and software-based implementations, we want to look at different applications of state-of-the-art models and see how those applications impact the attack performance of the aforementioned implementations. Therefore, we first looked at finding an implementation that was able to defeat **AES_HD_MM** and tried to reproduce and improve upon these results to see if it was feasible to break this data set and what kind of attack-performance one might expect. The next steps for our comparison were to attempt and use different state-of-the-art models on both **AES_HD_MM** and **ASCAD** to see if the attack performance was comparable and in which situations one proved to be easier to attack and overcome than the other. Two state-of-the-art models that seemed promising were chosen, *AutoSCA* [64] using Bayesian Optimization and Reinforcement learning for tuning the hyperparameters [51]. We will discuss the former in subsection 4.2.1 and the latter in subsection 4.2.2 later in this section.

4.2.1. AutoSCA

AutoSCA is a tool created by Wu *et al.* that tries to automate the deep learning hyperparameter tuning using Bayesian Optimization. This Bayesian Optimization seems to perform well, regardless of the data set, leakage model, or neural network type. Therefore, it is an interesting candidate for us, as it could effectively break hardware-based implementations that employ countermeasures. The paper connected to the tool itself also illustrates a comparison between Bayesian Optimization and Random Search for hyper-parameter tuning. Moreover, it concludes that while Random Search performs quite well, this is most likely due to the relatively easy data sets used within the paper and that many architectures can reach top performance. Because of this, we limited the usage of *AutoSCA* in our context to Bayesian Optimization as the search space for Random Search would get too large.

The original code[1] was modified by us in such a way that experiments with varying leakage model, model type, epochs, number of attacks, number of attack traces, number of profiling traces and batch-size could be run. We picked ID and HW for the leakage model, CNN and MLP for model type, 3 for the number of attacks, 10 and 50 for the epochs, 5 000 attack traces, 45 000 profiling traces, and a batch size of 50. All three different objective functions AutoSCA offers for Bayesian Optimization were used as well; L_m[65], key rank, and Accuracy.

4.2.2. Reinforcement Learning for Profiled Side-Channel Analysis

Reinforcement learning can be used to tune the convolutional neural network hyperparameters for profiled side-channel analysis, as shown by Rijsdijk *et al.*. The product that they presented was a modification of MetaQNN by Baker *et al.* [4] that uses two different reward functions employing the side-channel metric Guessing Entropy and has some assumptions that we know to be correct in the domain of side-channel analysis to speed up the computation of the model. This way, it was shown that state-of-the-art attack performance could be achieved with less trainable parameters than other well-known models. All in all, this seems like a promising method to use for the hardware-based implementations that use countermeasures, as it can deliver state-of-the-art attack performance for commonly used software-based implementations that use countermeasures like **ASCAD** with random keys (**ASCADr**).

For our setup, the original code[2] by Rijsdijk *et al.* needed only the addition of a model folder for **AES_HD_MM** with a hyper-parameter file and a state-space-parameters file. This model

[1] https://github.com/AISyLab/AutoSCA/blob/main/Auto_SCA.py
[2] https://github.com/AISyLab/Reinforcement-Learning-for-SCA

folder was created by copying another model folder and then modifying the needed parameters such as the input size to 3 125, the traces per attack to 5 000, the names to **AES_HD_MM** and the paths to the right files and folders.

4.3. Results

In this section, we will discuss the results that the experiments which were described in section 4.2 gave and analyze each of these results.

4.3.1. Breaking AES_HD_MM

In the paper by Won *et al.* [62] it is shown that breaking **AES_HD_MM** is possible, but to verify this, we chose to reproduce this result. To defeat **AES_HD_MM**, they used several different models; the Big Network (BN) from the ASCAD paper [48] by Prouff *et al.*, that same model BN but with the applied moving average technique with *n=100*, BN with Principal Component Analysis (PCA) applied and their own Multi-Scale Convolutional Neural Networks (MCNN). The one we chose to reproduce was the PCA version of the BN as we thought it out of the scope of the research to use the MCNN in our comparison, and the PCA version of the BN proved to perform the best of the three BN versions.

As can be seen in Figure 4.1, it is possible to break the **AES_HD_MM** data set with a generally extensive network that is proven to generalize well. However, the attack performance is far from ideal. We needed roughly 5 000 attack traces to reach a Guessing Entropy (GE) of below 10. While it is known that we can break data sets like **ASCAD** fixed key (**ASCADf**) within less than a few hundred traces with relative ease (note that this is also illustrated by Won *et al.*).

The reason why we see this kind of performance is that the BN used is not optimized for a higher-order hardware-based data set like **AES_HD_MM**. Logically, this leads to a somewhat disappointing performance with a lot of room for improvement. To realize this is because of the optimized model is important, as it gives us the indication that, by using networks that are optimized for higher-order hardware-based implementations, we can obtain significantly better attack performance.

By applying a model that was optimized for **AES_HD** by Zaid *et al.*, we were expecting to obtain better results than Won *et al.* were able to for **AES_HD_MM**. The best results we obtained experimenting with this model are presented in Figure 4.2.

It can be seen that while the model optimized for **AES_HD** by Zaid *et al.* works quite well for **AES_HD** as it reaches a Guessing Entropy of 0 after roughly 600 traces, it does not work for the **AES_HD_MM** data set. The results we obtained showed that we could not break the data set when using 45 000 profiling traces and 5 000 attack traces, while this was previously enough when applying the BN. A reason why we assumed this might be, is that the BN is quite a bit larger than the **AES_HD** model from Zaid *et al.*, this would allow it to be more generalizable and thus be more effective at attacking a seemingly challenging data set like **AES_HD_MM**. This illustrates at **ASCADf** is easier to break than **AES_HD_MM**.

Something that was discovered after running these experiments and observing their results was that the model that is used for **AES_HD** by Zaid *et al.* is most likely based on a different version of **AES_HD** that is significantly easier. We came to this conclusion when reproducing the work of Zaid *et al.* and saw significantly worse attack performance. As mentioned in subsection 2.4.2, there are different versions of **AES_HD** and it looks like the original results

were obtained with the version that was already pre-labelled. Zaid *et al.* most likely did not know this when doing and publishing their research, and therefore the model they suggest for **AES_HD** is not as applicable to our situation as we initially thought, as we are using the version of **AES_HD** that only has traces and labels.

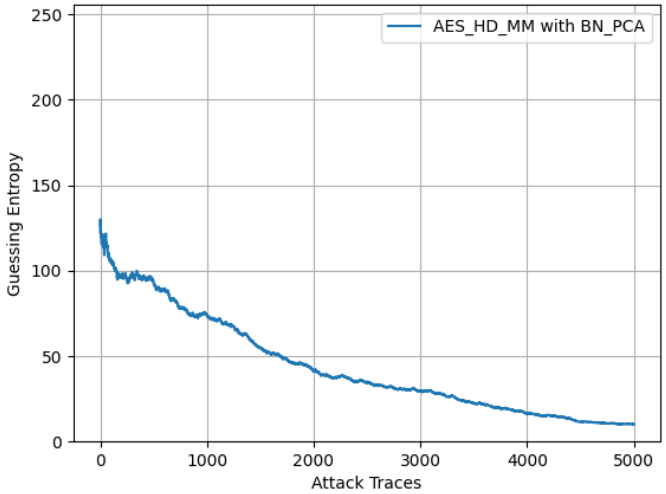

Figure 4.1: Guessing entropy results for the BN_{PCA} model on the **AES_HD_MM** data set, reproduced from Won *et al.*.

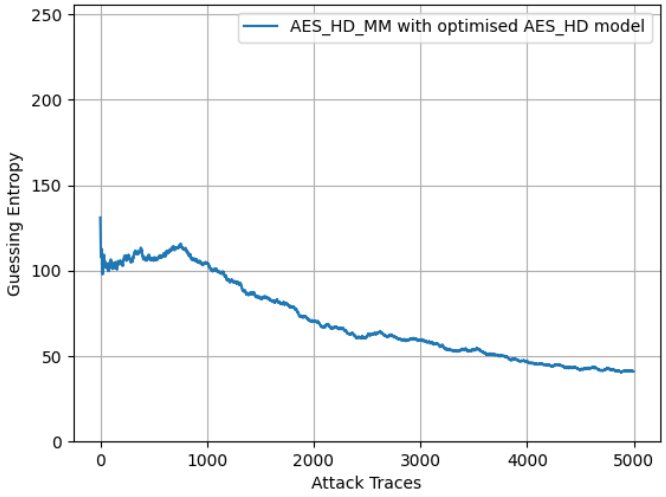

Figure 4.2: Guessing entropy results for the **AES_HD** model from Zaid *et al.* on the **AES_HD_MM** data set.

4.3.2. Applying AutoSCA

As discussed in subsection 4.2.1, *AutoSCA* is a promising state-of-the-art method that tunes a model towards the data set it is attacking. In the paper that introduces this method, results are displayed for attacking a software-based implementation that uses countermeasures: **ASCADr**. We chose to compare against the best performing variants of **ASCADr** with *AutoSCA* to give a most-fair comparison later on against our results from attacking **AES_HD_MM** with *AutoSCA*.

ASCAD with random keys

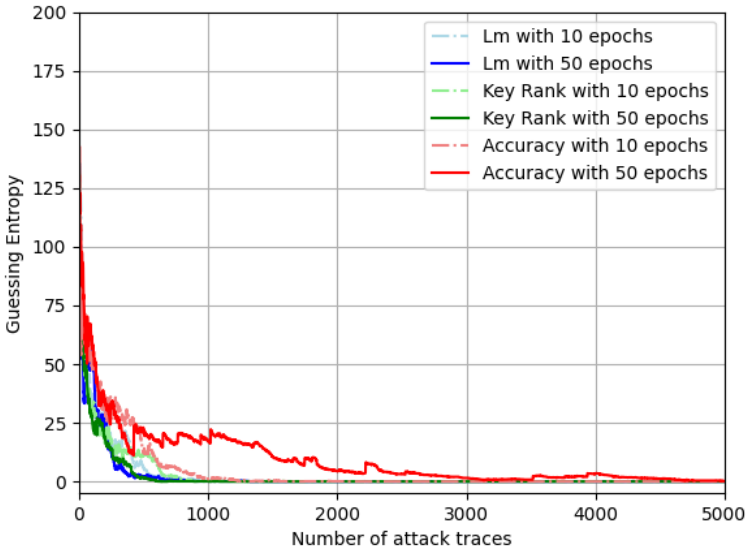

Figure 4.3: Guessing entropy results for *AutoSCA* using MLP models and the HW leakage model on **ASCADr**.

As can be seen in Figure 4.3 with all the objective functions and the MLP models that *AutoSCA* offers it is possible to break **ASCADr**. All of our results, except for the result with the Accuracy objective function and 50 epochs, can break the data set in under 1 000 attack traces, with the best results being delivered by the L_m objective function. This objective function with 50 epochs only needs roughly 500 attack traces to reach the coveted 0 Guessing Entropy. The aforementioned outlier, the Accuracy objective function with 50 epochs, only seems to converge to 0 Guessing Entropy after 3 000 attack traces. It even seems to go up afterwards and then settle again. However, this is most likely a result of some inherent randomness, as the results presented by Wu *et al.* showcase that Accuracy with 50 epochs should converge to 0 Guessing Entropy at around 1 500.

The rest of our results for this architecture do not differ too much from the original results obtained by Wu *et al.*, apart from the fact that we outperform their results with, of course, the exception of the Accuracy objective function at 50 epochs. Their best result was the L_m objective function at 10 epochs, which resulted in a Guessing Entropy of 0 at roughly 800 attack traces.

In Figure 4.4 we see that *AutoSCA* does not always prove successful when employing CNN architectures regarding **ASCADr**. None of the configurations of the CNN variants seemed to work in our experiments, except for the results we obtained with the Accuracy objective

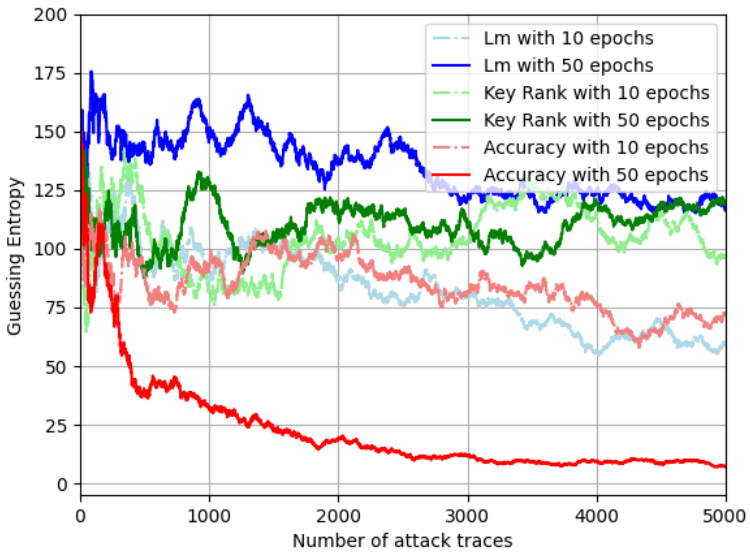

Figure 4.4: Guessing entropy results for *AutoSCA* using CNN models and the ID leakage model on **ASCADr**.

function at 50 epochs. In our experiments, this setting did not go to 0 Guessing Entropy but reached a Guessing Entropy of just under 10 and kept hovering there. This incessant hovering is an interesting contrast to the results we obtained for the MLP variants of *AutoSCA* we experimented with, as in that case, Accuracy with 50 epochs seemed to perform the worst. The same hyper-parameters were searched for as Wu *et al.* did in their work.

The results that Wu *et al.* present for the CNN variants of *AutoSCA* are a lot different from ours, as they can break the data set with the Key Rank objective function, which we are not. They need roughly 3 000 attack traces for this, and there seems to be no real difference between the 10 and 50 epoch variants. The results of the L_m and the Accuracy objective functions they report are within the domain of random guessing, which is similar to our results for the L_m and the Key Rank objective functions.

To conclude the results of the **ASCADr**, it seems to be more than possible to break this data set with the *AutoSCA* model, with the MLP variants seeming to be the most effective. This conclusion is confirmed by both our results and the original results published by Wu *et al.*. It is a good showcase of the state-of-the-art performance of the model for software-based implementations with countermeasures.

AES_HD_MM

However, as can be seen in Figure 4.5, Figure 4.6, Figure 4.7 and Figure 4.8, the results we obtained for **AES_HD_MM** with *AutoSCA* are not as successful as the ones that were obtained for the **ASCAD** data set. We used 45 000 profiling traces for **AES_HD_MM** and attacked it with 5 000 traces in these following experiments. These are the same amounts as used in subsection 4.3.1 where we were able to successfully attack **AES_HD_MM** with the BN model.

In Figure 4.5, we see that when we attack **AES_HD_MM** with *AutoSCA* using CNN-type models and train the best-obtained model for 10 epochs, all three objective functions are not able to break the data set. The Accuracy objective function seems to do the best, being the only

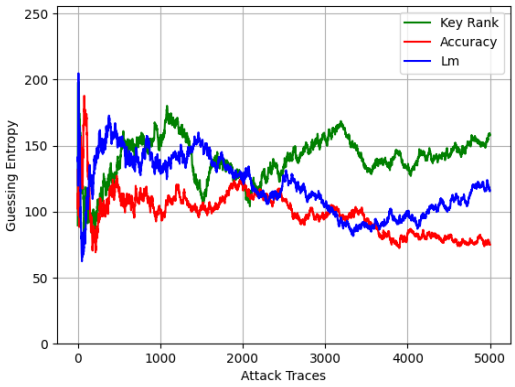

Figure 4.5: Guessing entropy results for *AutoSCA* using CNN models and 10 epochs on **AES_HD_MM**.

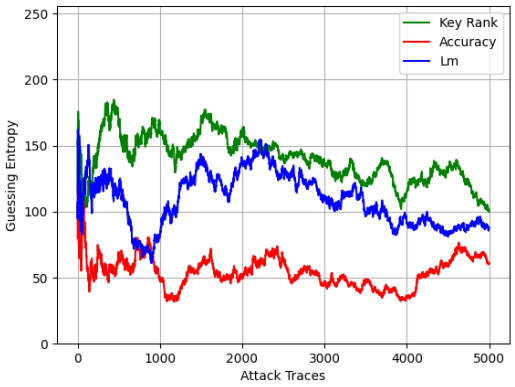

Figure 4.6: Guessing entropy results for *AutoSCA* using CNN models and 50 epochs on **AES_HD_MM**.

objective function that steadily decreases, all be it slowly. This trend of Accuracy outperforming the other objective functions continues when training the final model for 50 epochs, as can be seen in Figure 4.6.

Here in Figure 4.6, it can be seen that Accuracy outperforms the other objective functions but still does not break the data set. At best, it seems to hover around 50 Guessing Entropy, which can be explained by random variance.

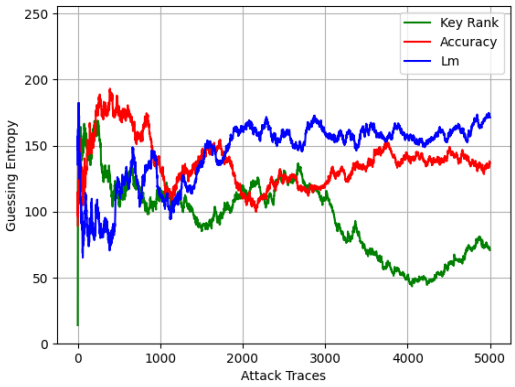

Figure 4.7: Guessing entropy results for *AutoSCA* using MLP models and 10 epochs on **AES_HD_MM**.

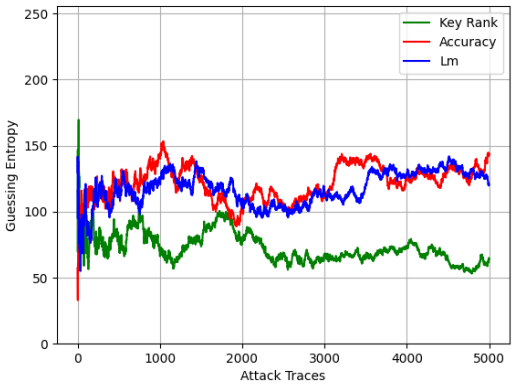

Figure 4.8: Guessing entropy results for *AutoSCA* using MLP models and 50 epochs on **AES_HD_MM**.

When **AES_HD_MM** is attacked with *AutoSCA* using MLPs, the results are not that much better, and we are not able to successfully attack the data set. However, the Key Rank objective function outperforms the other two instead of Accuracy. As can be seen in Figure 4.7, around 3 000 attack traces, the performance of the Key Rank objective function increases somewhat, and it starts to be performing better than the other two and hitting the lowest Guessing Entropy of 50 it obtains around 4 000 traces. This difference in attack performance is most likely caused by random variance, and not a result of Key Rank being a suitable objective function for this situation, as the attack performance decreases again and does not settle at a GE of lower than 10.

When we increase the number of epochs we train to 50, the performance of Key Rank becomes significantly more consistent as compared to when training with 10, which can be seen in Figure 4.8. However, this still does not result in a successful attack. The best result obtained is still far above the desired 0 Guessing Entropy at roughly 55 Guessing Entropy at its best.

Experiments with an increased number of traces

As the results presented in this subsection show that it is not possible to break **AES_HD_MM** with *AutoSCA* and the number of traces chosen, we decided to increase the amount of profiling and attack traces. We did this to see if it would be possible to successfully attack **AES_HD_MM** with the objective functions that performed the best for CNN and MLP models, respectively.

We decided to quadruple the total amount of traces used roughly; instead of 45 000 profiling traces, we used 200 000 as this is the amount that **ASCADr** uses. Furthermore, instead of the 50 000 traces for testing, we used 100 000 for the same reason as the increase in profiling traces. We also increased the amount of used attack traces from 5 000 to 20 000 to ensure that if it converges to 0 Guessing Entropy after 5 000 attack traces, as was used previously, we will now see that happen in our results.

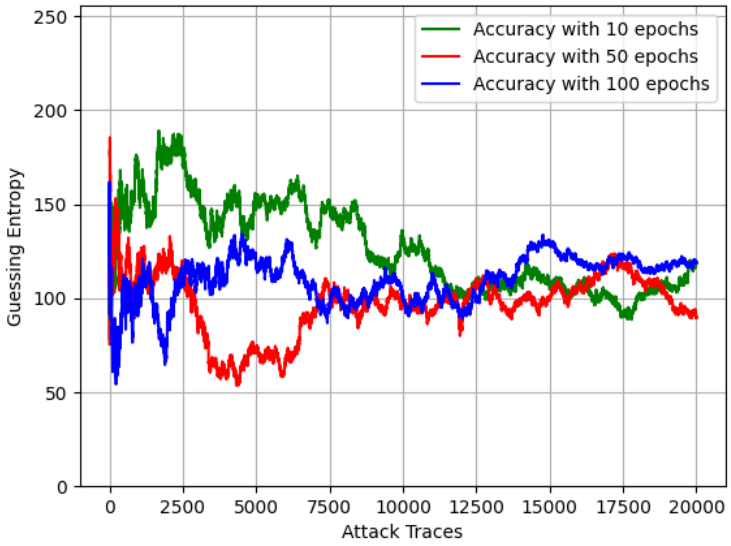

Figure 4.9: Guessing entropy results for *AutoSCA* using CNN models and an extended version of **AES_HD_MM**.

However, something interesting happened here, instead of the expected result of the added traces improving the performance of *AutoSCA* for CNN models, the models still do not seem to converge, as can be seen in Figure 4.9. All three of the train epoch variants we tested seem to result in a performance that is only slightly better than random guessing. The best performing variant of the three was the 50 epoch variant, ending with a Guessing Entropy of just under 100. Interestingly, the 50 epoch variant also has a significant drop around 3 000 traces and slowly goes up again from this point. The 100 epoch variant has similar erratic behaviour initially and then stabilizes around a Guessing Entropy of 126. Only the 10 epochs variant seems to decrease somewhat consistently, but as it starts with the highest Guessing Entropy

of the three, and the increase in performance mellows out around 12 500 attack traces, the final result of this variant is still barely better than random guessing. All in all, a successful attack was not possible with the CNN models *AutoSCA* created for **AES_HD_MM**.

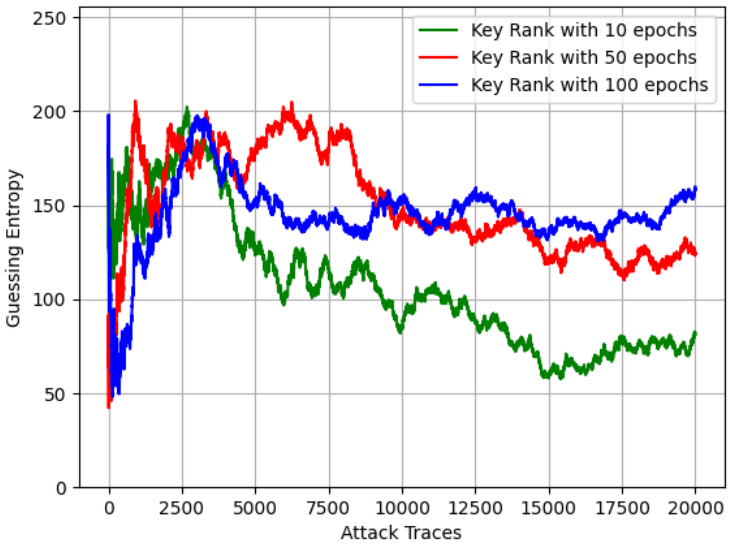

Figure 4.10: Guessing entropy results for *AutoSCA* using MLP models and an extended version of **AES_HD_MM**.

When analyzing the results we obtained in Figure 4.10, we can see that an erratic behaviour can be found in the beginning for all three epoch variants, which stops when all three hit a Guessing Entropy just below 200 around 2 600 attack traces. From that point out, the 50 epoch variant stays roughly the same until 7 500 attack traces and slowly decreases to a guessing Entropy of 126 when it reaches the final 20 000 attack traces. After showcasing similar behaviour, the 100 epoch variant arrives at a Guessing Entropy of 160. Similar to the CNN models variant of this extended version of **AES_HD_MM**, the 10 epoch variant seems to perform the most consistent with a slight but steady decrease in Guessing Entropy after peaking. When using MLP models, the 10 epochs variant performs the best and reaches a final Guessing Entropy of roughly 80. However, again, we need to deem it not possible to break **AES_HD_MM** using MLP architectures produced by *AutoSCA*. The reason for the behaviour shown in Figure 4.9, is that the network *AutoSCA* provided is most likely too small, and cannot generalize the complexities of **AES_HD_MM** well enough, leading to this unsuccessful attack performance.

Something of note to say about *AutoSCA* is that when selecting the final model to train with, it selects the model that performs the best according to its objective function and which is the smallest. This process often resulted in relatively small models, and as was seen in subsection 4.3.1, a bigger model seems to generalize better and obtain better results. However, one of the advantages of smaller models is that they seem to be less prone to overfitting [10]. A hypothesis we have is that the amount of profiling traces is potentially so large in our extended version of **AES_HD_MM** that it does not result in a successful attack.

However, the more likely scenario is that the earlier results we obtained with the original version

of **AES_HD_MM** would not converge, and thus increasing the amount of attack traces used would not improve the results of the model. With this being most likely the case, we need to find another method of attack to have a successful attack.

4.3.3. Applying Reinforcement Learning for Profiled Side-Channel Analysis

The following results that will be presented are all generated using the *Reinforcement Learning for Profiled Side-Channel Analysis* model, which was discussed in subsection 4.2.2. Something of note about this approach is that it takes significantly longer to compute these models. When running these models and using an NVIDIA RTX 3080 Ti GPU, the time consumption was upwards of three days to complete the process. This time consumption was significantly lower for the other approaches so far, as the most intense thus far was *AutoSCA* with roughly twelve hours for its most extensive run.

ASCAD with random keys

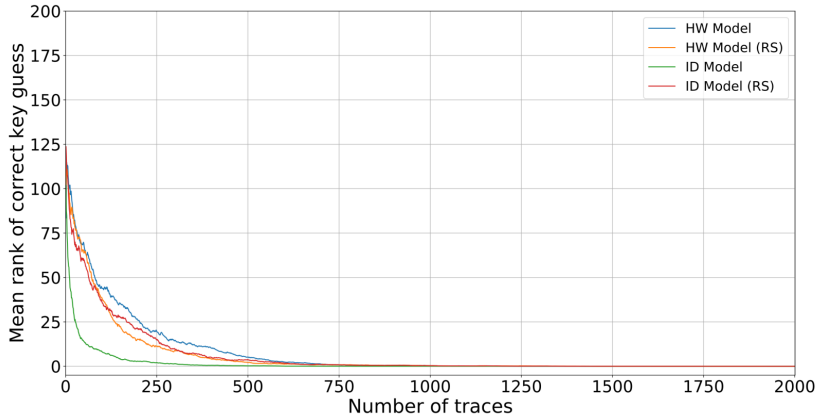

Figure 4.11: Guessing entropy results for **ASCADr** when using the *Reinforcement Learning for Profiled Side-Channel Analysis* model. Figure is taken directly from [51]

As shown in Figure 4.11, a result from [51], it is clear that it is possible to break a software-based implementation that uses a countermeasure. Both the leakage models and their reward function variants perform well and can break the data set. The ID leakage model that uses the regular reward function instead of the small reward function (RS) offers the best performance out of the four configurations. It already reaches the desired 0 guessing entropy around 300 traces, while the other three take roughly 600 traces.

AES_HD_MM

When we attacked **AES_HD_MM**, we were sceptical of the success of the results we thought we would obtain. This scepticism came from the fact that we noticed that small architectures had not been very successful thus far, and the *Reinforcement Learning for Profiled Side-Channel Analysis* model delivers relatively small architectures, as can be seen in the work that Rijsdijk *et al.* present. However, we were pleasantly surprised by the results that we received, which can be seen in Figure 4.12. The *Reinforcement Learning for Profiled Side-Channel Analysis* models were able to break the **AES_HD_MM** data set and obtain the best performance we had seen for this data set. The variant that uses the normal reward function seems to perform the best. It reaches the desired 0 Guessing Entropy at around 3 500 attack traces, roughly 800 traces earlier than the RS variant.

We noted that the RS variant was less erratic in the changing of Guessing Entropy as compared to the standard reward function. This stability might be because the reward function

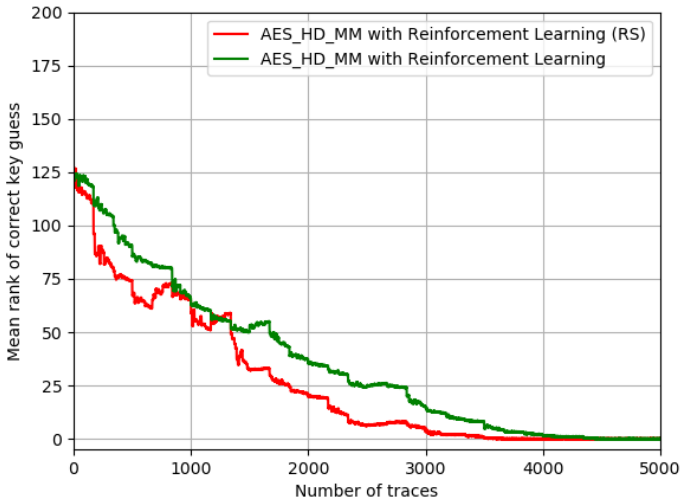

Figure 4.12: Guessing entropy results for **AES_HD_MM** when using the *Reinforcement Learning for Profiled Side-Channel Analysis* model.

rewards smaller, high-performing networks more as compared to the standard reward function. As these networks are smaller, they might behave more consistently.

4.4. Discussion

When comparing all the results we obtained for both software-based and hardware-based implementations using countermeasures, we came to the following conclusion; it is always possible to obtain a successful attack result on the former when using state-of-the-art approaches like *AutoSCA* or *Reinforcement Learning for Profiled Side-Channel Analysis*, while it is not with the latter. The performance of the **ASCADr** varied throughout the state-of-art-approaches, with the best *AutoSCA* approach offering us a successful attack within 800 traces, and *Reinforcement Learning for Profiled Side-Channel Analysis* taking roughly 300 traces. These results seem to significantly outperform the results we obtained for the hardware-based implementation that uses countermeasures, **AES_HD_MM**, when using the same state-of-the-art methods. We needed more than ten times the attack traces to successfully attack this data set compared to **ASCADr** when using *Reinforcement Learning for Profiled Side-Channel Analysis*, and we were unable to break this data set using *AutoSCA*.

This is because said methods are not optimized for attacking hardware-based implementations and therefore do not perform well. A hypothesis we have that might explain their substandard performance is that the size of the architectures that *AutoSCA* and *Reinforcement Learning for Profiled Side-Channel Analysis* provide are relatively small, and hardware-based implementations with countermeasures require larger-scale models.

From our point of view, this is due to what we had seen in subsection 4.3.1; there, the larger model (BN) significantly outperformed the smaller model (the one optimized for **AES_HD**). These results lead us to believe that the smaller models fail to model the complexities and the noise that hardware-based implementations with countermeasures have and are therefore not suited for attacking hardware-based implementations.

The figures we provided regarding our results with *AutoSCA* and **AES_HD_MM** clearly reflect

this. The models *AutoSCA* generates with their method end up being too small and cannot obtain successful attack results. Even when the size of the profiling traces is dramatically increased, like in Figure 4.9 and Figure 4.10 *AutoSCA* is not able to deliver a successful attack, showcasing this problem.

We think that because the *Reinforcement Learning for Profiled Side-Channel Analysis* method is more flexible in the sizes of the networks it provides, due to its reward function, it is able to deliver successful attacks as can be seen in Figure 4.12. It must be noted that it is interesting that the RS reward function somewhat outperforms the normal reward function, as the RS reward function tends to deliver smaller network sizes. Therefore, we think that the RS reward function does not inhibit the size of the network too much if it comes to a better performing network in the case of hardware-based implementations and can actually deliver better results due to its stability improvements.

The results we obtained in this chapter, both good and bad, give us a direction to go into for our next chapter, as we can start to experiment with larger networks and aspects of *AutoSCA* or *Reinforcement Learning for Profiled Side-Channel Analysis* to create a novel attack method that works well on higher-order hardware-based implementations.

Considering our results and other results in the field of Profiled Side-Channel Analysis, we deem it more challenging to obtain a successful attack on hardware-based implementations that use countermeasures than on software-based implementations that use countermeasures.

5

Constructing a novel attack method for hardware-based implementations

This chapter considers a novel method for attacking hardware-based implementations with and without countermeasures. We hope to establish a base methodology with this chapter that describes a plan of attack for these hardware-based implementations. The methodology should be constructed so that it can be improved upon in the future. Similarly to chapter 4 we start with motivating why this needs to be examined, and a methodology could prove to be a helpful addition to the field of SCA. We will also discuss what our novelty is, why it is novel and why it makes sense to try out this novelty. Afterwards, we discuss our experimental setups to clarify how we came to our suggestion for the finalised method to attack hardware-based implementations. Finally, we analyze the results of the experiments and discuss them in detail.

5.1. Motivation

In chapter 4, we showcased that often hardware-based implementations are more difficult to attack and analyze than their software-based counterparts. A methodology or a plan of attack for when a new hardware data set needs to be analyzed would be a great addition to the current field of knowledge of SCA. To our knowledge, no such methodology exists currently, and we hope to give at least a basis for one in this chapter. Our proposed methodology will be based on the novelty of using multiple pre-processing branches in the network and a significantly increased depth in our network via the usage of ResNets. Whilst both methods have been used before in the domain of SCA, they never have been applied in a direct manner to hardware-based implementations and optimized for them. Therefore, we think it makes sense to see if we can use the success of these methods on software-based implementations in tandem with what we found in chapter 3 and chapter 4 to create a novel approach for attacking higher-order hardware-based data sets. We will discuss the method we found in the end, and how one can apply it to hardware-based implementations with state-of-the-art performance. Therefore this could be useful to build upon for future research. At the time of writing, this method is a novelty in its application to hardware-based higher-order data sets. Therefore, it could pave the way for improved attack methods for these aforementioned data sets.

5.2. Experimental Setup

The two ways we considered to be effective in attacking hardware-based implementations, both in our experiments and the research we did, were the size (especially depth) of the network and the pre-processing of the data sets. We look into both directions, by first creating an

approach based on the MCNN of Won *et al.*, and afterwards, we propose the application of ResNets, which are known for their sizable depth and thus the size of the network. Both are novel in their application of being directly optimized for hardware-based implementations. We built and customized our methods for the **AES_HD** and **AES_HD_MM** data sets to showcase that if our methods are successful, they can be both effective for hardware-based implementations that both use and do not use countermeasures.

5.2.1. MCNN approach

Seeing the success of the MCNN introduced by Won *et al.* we thought that by modifying their approach, we could create a optimized version of their network for hardware-based implementations. As Won *et al.* also published their work with the results of their network on the **AES_HD** and **AES_HD_MM** data sets, we do not deem our own approach completely novel, but we see it as a step towards finding an approach for successfully attacking hardware-based implementations.

In our experiments with MCNN variations we decided to try out the following types of preprocessing:

- **MA**: The moving average is a statistic that captures the average of a data set over the span of several sliding windows of certain sizes. This allows us to reduce the number of inputs to consider but still get a good estimate of what the data is supposed to represent. This pre-processing technique was used in the paper by Won *et al.* as well, here we decided to use the 50/100/400 window size variations.
- **PCA**: Principal Component Analysis is the process of reducing its input to its most important features. This is used by Won *et al.* to reduce the input to the 50 most important features, we decided to test a reduction to 100 features as well.
- **EA**: Elastic Alignment attempts to counteract desynchronization caused by jitter or delays. It is based on a dynamic time warping algorithm adopted from speech recognition and has seen some success in its application to DPA [61]. This type of pre-processing was mentioned by Won *et al.* but it is unclear how much they experimented with this.
- **SMOTE**: The Synthetic Minority Oversampling Technique is a well-known re-sampling method that oversamples by generating synthetic minority class instances [13]. This can be used to balance the data and hopefully improve the performance, this pre-processing technique was not utilised by Won *et al.* but explored with some success by Picek *et al.*.

In the experiments done by Won *et al.* the combination of MA and a branch of unaltered input (called “original”) seems to work well and returns in almost all their experiments. For this, they chose an MA window size of 100 and PCA with a feature reduction of 50. We, therefore, assumed this approach to already have been validated to a degree and took this with us in the experiments that we ran. We in the end ran the following combinations of pre-processing for our MCNN:

- “Original”, MA-100, PCA-100
- “Original”, MA-400, PCA-50
- “Original”, MA-100, PCA-50, SMOTE
- MA-50, SMOTE

- “Original”, SMOTE
- “Original”, MA-400, SMOTE
- “Original”, MA-100, PCA-50, SMOTE
- “Original”, MA-100, PCA-50, SMOTE, EA

Note that this list of experiments is far from exhaustive, but we deemed these experiments to be the most interesting as they were either close to the original MCNN by Won *et al.* or in a completely different direction. An exhaustive search over all the combinations of the chosen pre-processing methods could be performed but we ran out of time to do this.

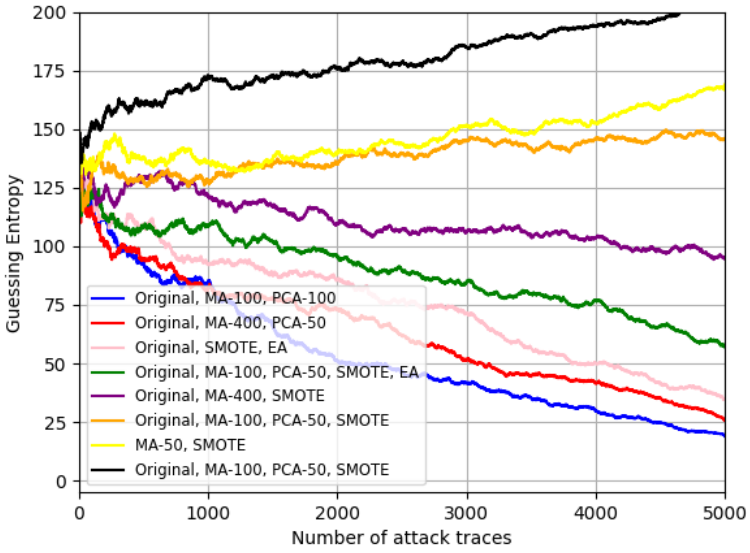

Figure 5.1: Different MCNN setups on the **AES_HD_MM** data set.

As can be seen in Figure 5.1, in some cases our MCNN setups come close to that of those of Won *et al.*. However, we do not deem this approach an improvement upon their work and far from a successful approach to hardware-based implementations. Of course, it could be that we did not try out the right combinations of pre-processing techniques and to fully validate this, an exhaustive approach could be taken. We deem with the current results the usage of different branches of pre-processing a promising, yet not good enough for being the new state-of-the-art, approach in attacking hardware-based implementations.

5.2.2. ResNet approach

We considered using a variant of the residual blocks that Jin *et al.* used for our ResNet. Varying versions of these blocks were tested, but in the end, we settled on using a version that did not use batch normalization with a kernel size of 2 for both **AES_HD** and **AES_HD_MM**.

The number of residual blocks for the two different data sets ended up being slightly different from what we expected. We expected to use the maximal amount of residual blocks while reducing the feature map size by a factor of two for each block, leading to $\lfloor \log_2(1250) \rfloor = 10$

and $\lfloor \log_2(3125) \rfloor = 11$ respectively for the two data sets as was done by Karayalçin *et al.* [29]. However, we ended up using 8 residual blocks for **AES_HD** and 9 for **AES_HD_MM**, resulting in 16 and 18 convolutional layers, respectively. The number of filters that were chosen in each convolutional layer was computed by $min(2^{i-1}, 256)$ with i being the number of the residual block, starting from 1. We also varied neurons a bit and came to the conclusion that 10 neurons worked best for **AES_HD** and 16 neurons worked best for **AES_HD_MM**.

Both architectures can be seen in more detail in Figure 5.2 and Figure 5.3.

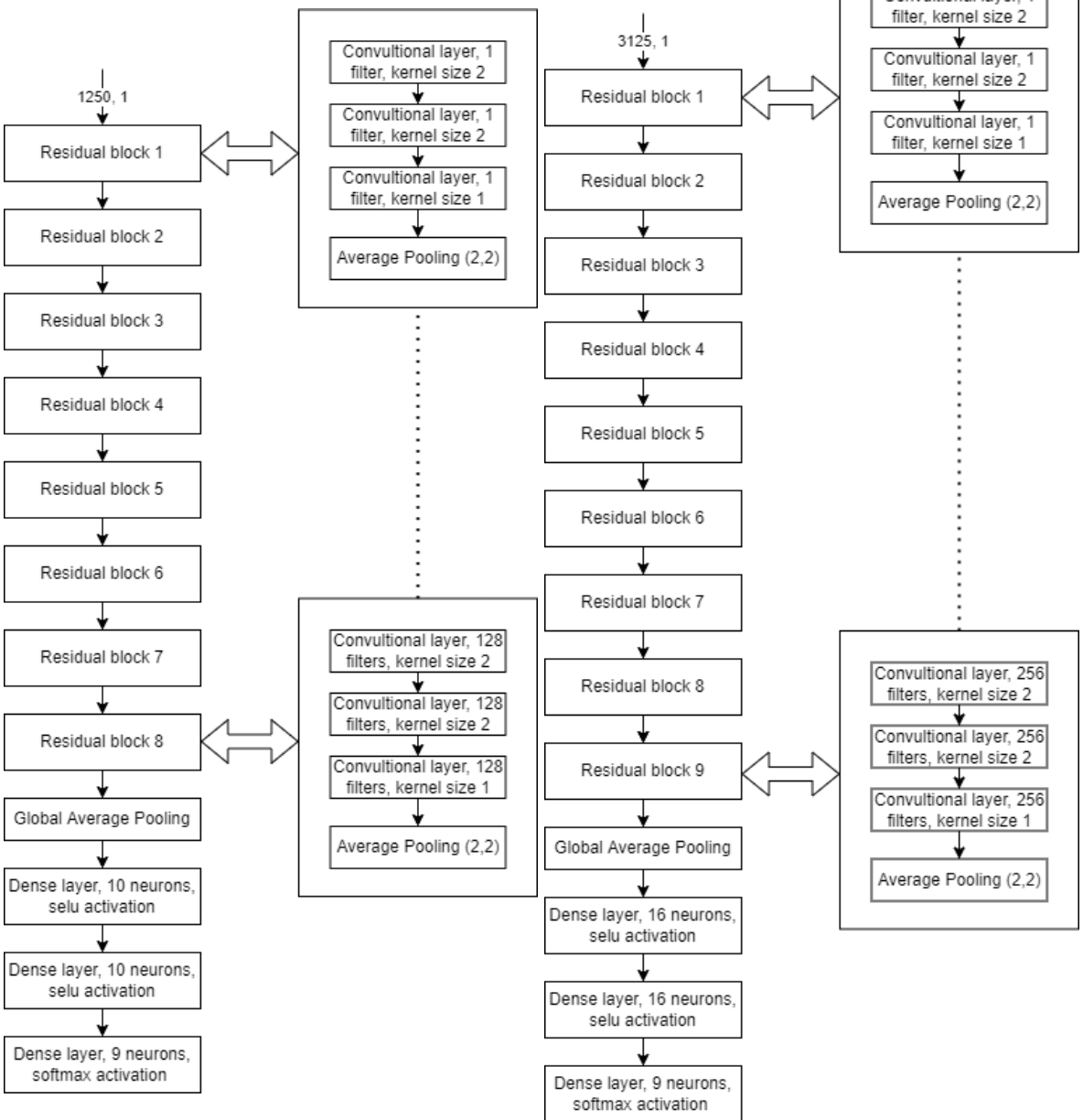

Figure 5.2: ResNet architecture used for **AES_HD**.

Figure 5.3: ResNet architecture used for **AES_HD_MM**.

We came to the conclusion of using this particular network setup, by performing grid search optimisation on hyper-parameters we pre-selected. These hyper-parameters were partly based

on the findings of Zaid *et al.* in their methodology as we thought that to be a good starting point for our own approach. The table of this grid searchTable 5.5 can be found in the discussion, and we would like to note that, as the trend goes for grid-based searches, far from all the results of this grid search were successful.

Size comparison of network

The size of the networks that we found is also something to consider. For example, do these networks have many trainable parameters and do they over-fit their respective data sets? Are they unnecessarily large, and do they cost too many resources to train as there might be better and smaller alternatives? To consider this, we analyzed the size of our networks and compared them to networks of similar and varying sizes.

For **AES_HD**, there are considerably more networks published that are successful in their attack of the data set than for **AES_HD_MM**. This abundance of results leads to that comparison being more in-depth and will give a better view of the performance related to the size of the network, which needs to be held into consideration.

	Our ResNet	**Wouters et al.**[63]	**Zaid et al.**[67]	**Zaid et al. (BN)**	**Jin et al.**[28]
Trainable params	78 723	2 020	3 282	142 044	~300 000

Table 5.1: Table showcasing the comparison of different architectures sizes on **AES_HD**

As we can see in Table 5.1, the two networks made for **AES_HD** by Wouters et al. and Zaid et al. are considerably smaller, so outperforming these networks is something we can hope to expect with a network that is $38,97$ and $23,99$ times larger respectively. The BN which we saw used in chapter 4 is $1,8$ times as large, and the estimated size of the Jin *et al.* network is roughly four times as large. Therefore if we can outperform those networks for **AES_HD**, it would showcase that we can improve current results with smaller networks. We could sadly not find a network of similar size to our own network to compare against.

	Rijsdijk et al.[51]	**Our ResNet**	**Wu et al.**[64]	**Zaid et al. (BN)**[67]
Trainable parameters	192 105	311 905	349 596	142 044
Non-trainable parameters	0	0	1 344	0
Total parameters	192 105	311 905	350 940	142 044

Table 5.2: Table showcasing the comparison of different architectures sizes on **AES_HD_MM**

For **AES_HD_MM** we found it hard to make this comparison, as there are not many known results for the data set. We chose to compare to the results we knew the total amount of trainable parameters to, this being the results that Wu *et al.* and Rijsdijk *et al.* obtained. Our network is not the largest, being $0,89$ times the size of the network by Wu *et al.*, as can be seen in Table 5.2, but it is larger than the other two networks of which we know the results. Due to the network of Wu et al. having their unique three different branches that use pre-processing, they also have some non-trainable parameters, which we chose to showcase in our comparison. Therefore, if we can have comparable results to the network of Wu et al. for **AES_HD_MM**, we showcase that the current state-of-the-art for **AES_HD_MM** can be improved upon without increasing the size of the network. The best performing network that we were able to get with Rijsdijks *et al.* method is smaller by a significant amount, roughly 120 000. Therefore we expect to outperform it based on our results so far.

We also chose to make a comparison against the architecture we used to first successfully break the **AES_HD_MM** data set with to give a frame of reference for what someone can expect when using an architecture that was not made with attacking hardware-based implementations in mind.

5.3. Results

In this section, we will present the result we obtained for **AES_HD** and **AES_HD_MM** using the models and method described in section 5.2.

5.3.1. AES_HD

Applying the ResNet we developed for the **AES_HD** to the **AES_HD** data set, we gain results that are better than the former state-of-the-art presented by Zaid *et al.* and the improvement thereof presented by Wouters et al. as can be seen in Figure 5.4. Our ResNet network reaches a Guessing Entropy of 0 around the 4 000 attack traces mark. To obtain this result, we used 50 000 profiling traces, 5 000 validation traces and 5 000 attack traces.

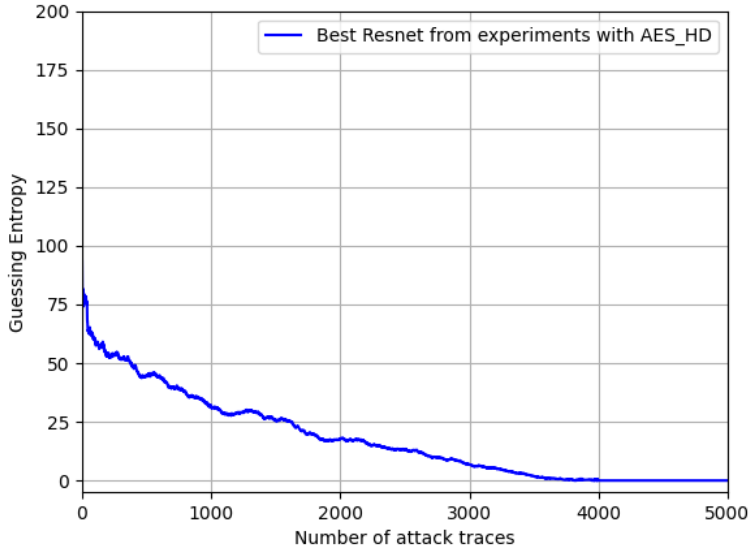

Figure 5.4: Guessing entropy results for our ResNet on **AES_HD**.

Something of note is that while our network seems to perform better than the former state-of-the-art of Zaid *et al.* and the improved state-of-the-art of Wouters *et al.*, in [28], Jin *et al.* show that with their application of a Convolutional Block Attention Module (CBAM) to ResNets they can reach a Guessing Entropy of 0 within 2 100 attack traces. They speculate this is because the attention network can focus on the leakage regions and ignore the unnecessary points by using CBAM. This attention network reduces the influence of the environmental and algorithmic noise, which are very prevalent in **AES_HD** and thus improves performance.

An overview of this comparison can be seen in Table 5.3

	Jin et al.[28]	**Our ResNet**	**Wouters et al.**[63]	**Zaid et al.**[67]	**Zaid et al. (BN)**
NoT for GE <1	2 100	3 900	5 800	5 900	11 300

Table 5.3: Table showcasing the comparison of different architectures on **AES_HD**

Taking these performance results in mind, we can also now reflect on the sizes of the networks we listed in Table 5.1. We outperform the two smaller networks, those by Wouters et al. and Zaid, by roughly 2 000 attack traces each. A sizeable improvement but smaller than we hoped for the increase in the size of the network. We also outperformed the BN originally introduced by Zaid et al. This was expected as this network is not optimized for hardware-based implementations. However, it is still good to see that we outperform this network with roughly a third of the attack traces and half the size. The network by Jin et al. outperformed our network with quite a margin. However, the network uses roughly four times the trainable parameters and is, because of that, significantly more expensive to train. Therefore, we deem our network to give good results for its relative size.

5.3.2. AES_HD_MM

When using our ResNet on **AES_HD_MM**, we receive results that are significantly better than the original publishing of the data set, which used DPA [16] as can be seen in Figure 5.5. This makes sense, as the application of profiled attacks is a lot more sophisticated than a direct attack, and the field of profiled Side-Channel Analysis has evolved and changed a lot since 2014 when this attack was performed. Ding *et al.* could only reach a success rate of 90% with 500 000 traces, which showcases the difficulty of attacking a masked hardware AES implementation.

However, comparing our results to more recent results, such as the ones published in [64], we still see a significant improvement in results. When using our extended version of **AES_HD_MM** we can reach a Guessing Entropy of 0 within 2 200 attack traces, while the best result we could find was from Wu *et al.*, taking roughly 4 500 attack traces.

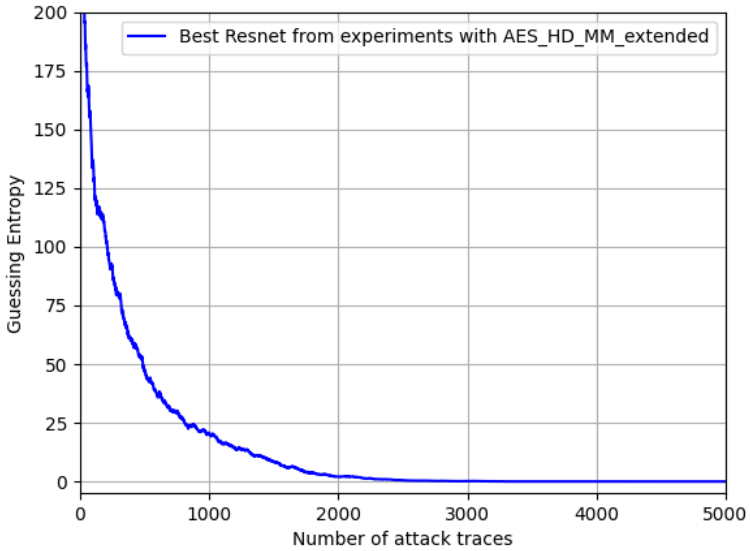

Figure 5.5: Guessing entropy results for our ResNet on **AES_HD_MM**.

An overview of the comparison between different architectures can be seen in Table 5.4. Regarding the size of the networks that obtained the results we can see in Table 5.4, we refer to the parameters we found in Table 5.4. Our ResNet optimized for **AES_HD_MM** outperforms the network by Wu *et al.* in both size and amount of attack traces needed for a successful attack. It also significantly outperforms the BN, the smallest network we have results for

	Our ResNet	**Rijsdijk et al.**[51]	**Wu et al.**[64]	**Zaid et al. (BN)**	**Ding et al.**[16]
NoT for SA	2 200	3 500	4 500	6 200	500 000

Table 5.4: Table showcasing the comparison of different architectures on **AES_HD_MM**

AES_HD_MM. The factor of the difference in needed attack traces is greater than the factor of the difference in the size of the networks. These factors are $2,82$ and $2,05$, respectively. While the ratio of size and attack performance is similar to that which we obtained with the method from Rijsdijk *et al.*, our ResNet can be trained significantly faster. Therefore, we deem our ResNet to be better than the state-of-the-art for **AES_HD_MM**.

5.4. Discussion

In this chapter, we found that we could obtain results competitive with the state-of-the-art for hardware-based implementations with our ResNets. This result showcases the power of the novel design element in our method. With the modification that Karayalçin *et al.* did of the residual blocks of Jin *et al.*, we showcased that by making some hyper-parameter adjustments and taking ideas of state-of-the-art models, we can get good results and even improve state-of-the-art results for specific data sets. The sizes of our proposed networks are not the smallest nor the largest networks that perform well for their data sets, and we consider their size-to-performance ratio more than acceptable. As mentioned in section 5.1, there is currently no consensus or methodology on how to approach higher-order hardware-based data sets, which is something we will create here now.

We also presented our experiments with various setups of MCNN. With the results that Won *et al.* presented in their work, we expected that with slight modifications, and novel pre-processing methods in these setups, we could improve upon their work. However, this was not the case and our best results were less successful at attacking **AES_HD_MM** than the original MCNN. We still think this approach of using a neural network with multiple pre-processing branches an interesting choice for attacking hardware-based implementations, and we think that with more experiments and a more in-depth look it could be made successful, but deemed this out of the scope of the research and decided to focus on our ResNets.

Hyper-parameter	**Value**
Optimizer	{SGD, RMSprop, Adam}
Weight initialization	{Uniform distribution, He uniform}
Learning rate	{0.01, 0.001, 0.0001} with One-Cycle Policy
Batch size	{50, 64, 128, 256, 512}
Epochs	{10, 25, 50, 75, 100, 125, 150}
Activation function dense layers	{tanh, ReLU, SeLU}
n° of neurons dense layers	{10, 16, 32, 64, 128, 256}
n° of layers dense layers	{1, 2, 3, 4, 5}
Starting filter size res block	{1, 2, 4, 8, 16, 32}
Kernel size res block	{2, 4, 8, 12, 16}
n° of convolutional layers res block	{1, 2, 3, 4, 5}

Table 5.5: Grid search optimisation on hyper-parameters for our suggested ResNet architecture.

As hinted towards in subsection 5.2.2, the majority of the strength of our proposed ResNets lies in their depth, which is decided by the number of residual blocks. This was suggested by

Karayalçin *et al.* and in our experiments, this proves to be true for hardware-based implementations. The number of residual blocks we propose to use for a data set can be calculated by taking the log_2 of the number of features of that data set and then flooring that. This number should be the highest amount of residual blocks one needs for their data set. We then propose that one can try and decrease the number of trainable parameters by lowering this number until the results start to deteriorate.

We propose performing a grid search optimization on the hyper-parameters for the rest of our parameters, just like Zaid *et al.* did in their methodology. See Table 5.5 for the hyper-parameters that we propose for the grid search optimization.

We realize that this is far from a conclusive methodology. However, we hope to lay the groundwork for a future consensus on how to approach designing the architectures for hardware-based data sets of a higher order.

We did consider, but thought to be out of scope for the research, applying Bayesian optimization as AutoSCA does, or even random search optimization instead of the grid-based search optimization we used. As the amount of hyper-parameters is quite large and might grow with improvements to our base methodology as we learn more about the field, these other optimization forms could prove helpful.

6

Conclusions

In this final chapter, we will answer our research questions, as stated in section 3.4. Then we will discuss the limitations of our work. And finally, we will propose how these limitations could be addressed and additional future work to improve on the work done.

In chapter 4 we evaluated the differences between higher-order software-based and hardware-based implementations using state-of-the-art methods. Here we confirmed the prevalent hypothesis throughout the SCA field; higher-order hardware-based implementations are more challenging to attack than their software-based counterparts. In chapter 5, we introduced our novel attack method of applying ResNets to hardware-based implementations and showcased excellent results, even improving upon the best-known results for **AES_HD_MM**.

Research question 1: What is the difference in the attack performance between software-based implementations and hardware-based implementations?
Answer: The difference in attack performance we see is that hardware-based implementations, both with and without countermeasures, need more attacking traces on average to recover the correct key. We showed this in chapter 4 with the usage of two state-of-the-art methods, by applying said methods to hardware-based implementations and reproducing results that were had for software-based implementations. The state-of-the-art method *AutoSCA* was very successful at attacking software-based implementations. However, we could not obtain a successful attack on a hardware-based implementation with *AutoSCA*.

We also showcased that it is possible to break higher-order hardware-based implementations using deep learning SCA with even naive approaches like using a known successful large network that would most likely fit itself towards the data set well enough. This was done by attacking the **AES_HD_MM** data set using the Big Network (BN) from Zaid *et al.* and PCA as pre-processing. This resulted in the attack performance of 6 200 attack traces to reach a Guessing Entropy of below 1. Whilst this attack might have been succesfull in recovering the correct key, it does, however, take a significant amount of attack traces and we can do better as showcased in our later experiments.

Running our experiments, we found out that while the reinforcement learning method developed by Rijsdijk *et al.* was decently effective at breaking **AES_HD_MM**, *AutoSCA* was not. The performance we saw with the reinforcement learning method was the best we had seen so far, with 3 500 attack traces needed to get a Guessing Entropy of below 1, *AutoSCA* was not able to break the data set at all. *AutoSCA* being unable to break the data set indicates

that these state-of-the-art methods are not all applicable to every data set out there and that even with a significant amount of tweaks and improvements, they do not necessarily work.

Finally, as our results indicated that attacking a software-based implementation using countermeasures like **ASCAD** with random keys is significantly easier than attacking a hardware-based implementation using countermeasures like **AES_HD_MM**, we concluded that the countermeasures in the latter data sets seemingly prove more effective than in the former data sets.

Research question 2: Which novel design elements can we introduce to improve the attack performance on higher-order hardware-based data sets?
Answer: We can improve the attack performance on higher-order hardware-based data sets using our novel design element of ResNets. We showed this in chapter 5 by showcasing how our proposed ResNets were constructed, what their amount of trainable parameters and then comparing their needed amount of traces to attack a hardware-based data set successfully. Using our proposed novel attack method, we could also obtain good results on hardware-based data sets that do not employ countermeasures, which was something we did not fully expect.

While working towards a new novel design element for attacking higher-order hardware-based data sets, it became apparent that there is no consensus or methodology on approaching these data sets, as mentioned earlier. We make the first attempt at this in chapter 5 by showcasing the structure of our novel design element and how we choose critical parameters for this.

The grid-search part of our novel design element is heavily inspired by the approach of Zaid *et al.* [67], which employs a similar grid-search tactic. The idea of ResNets came from Jin *et al.* and Karayalçin *et al.*, and as our results are better than their results, we deem it so that improvements can be made on attack performance by making adjustments to state-of-the-art models or previous state-of-the-art models. We also mention in section 5.4 that applying the Bayesian Optimization that *AutoSCA* uses might improve our novel design element even more, which would show this too, but this needs to be researched before we can claim this.

Our novel design element compares decently to other network sizes. For **AES_HD** it seems to be of average size, and for **AES_HD_MM**, one of the larger ones. Taking the performance of other networks into account, we deem the sizes of our novel design element good. They might not be the smallest networks available with good results, but they perform like one of the best networks or better than the state-of-the-art by a significant amount.

6.1. Summary of Scientific Contributions

An overview of the main scientific contributions of this work is as follows:

- We showcase that it is more challenging to successfully attack hardware-based implementations than software-based implementations using state-of-the-art results and empirical results.
- We find that state-of-the-art methods have varying amounts of success in hardware-based implementations that incorporate countermeasures.
- We show the effectiveness of ResNets by introducing a novel design method for attacking hardware-based implementations that uses ResNets.

- We show that we can create a novel design element that performs better than state-of-the-art with an analysis of successful networks and modifications to state-of-the-art methods.

6.2. Limitations

A shortcoming of our comparison of hardware-based and software-based implementations using countermeasures is that we only used **ASCAD** with variable keys and **AES_HD_MM**. The comparison would be better, and a more concrete answer could be drawn if more data sets were used. However, currently, there are no other hardware-based implementations using countermeasures publicly available than **AES_HD_MM**, so it is unclear how to resolve this part of the limitation at this time.

While we do use two different state-of-the-art methods in our analysis of hardware-based and software-based implementations, this does not represent the full spectrum of state-of-the-art profiled SCA methods available at the moment.

A caveat of our methodology is that we suggest finding the number of residual blocks by taking the log_2 of the number of features of the data set and then flooring that number. After this, improvement can be made by lowering this floored number until the network's performance goes down, but of course, this adds extra time and computation to finding the ideal network.

Another limitation is that the grid search hyper-parameters we suggest for our novel method were successful for the data sets we used within this work; they might not be for other hardware-based implementations. While careful consideration went into choosing these hyper-parameters, many other options exist and could prove to be even more effective.

6.3. Future Work

The limitations in the previous section provide many different options for future work, and we suggest a few options for this here in this section.

The comparison and analysis of the difference in attack performance between hardware-based and software-based implementations could improve if more different data sets were used. This is a clear step for future work and could be done by using other data sets like **ASCAD** with a desync or the new **ASCADv2**. Other hardware-based implementations using countermeasures than **AES_HD_MM** are not publicly available at the time of writing, but they could be created and made publicly available to resolve this. This would also be a great addition to the field of research, as it would allow for other research into hardware-based implementations using countermeasures.

A clear step for more future work in comparing the difference in attack performance between hardware-based and software-based implementations is using other state-of-the-art methods. The method proposed by Jin *et al.* could be used, methods that are not published or invented yet, and even our method could be used in this future work. This would make the comparison findings more complete and a better basis for future hypotheses or improvements to the literature.

Significant improvements to our suggested novel method can be made; we suggest applying Bayesian Optimization like *AutoSCA* does, which could optimize run times and results dras-

tically for larger search spaces. Another improvement to our novelty would be improving the choice of residual blocks. This could be done by creating a method that would suggest several options for the number of residual blocks by using our log_2 method or a more advanced concept and then using that in the hyper-parameter optimization.

Finally, a possible direction for future work is looking into the application of our novel method in software-based implementations. Our methodology that we developed specifically for hardware-based implementations could potentially prove successful in this domain, but that remains to be seen.

Bibliography

[1] Dakshi Agrawal, Bruce Archambeault, Josyula R Rao, and Pankaj Rohatgi. The em side—channel (s). In *International workshop on cryptographic hardware and embedded systems*, pages 29–45. Springer, 2002.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 international conference on engineering and technology (ICET)*, pages 1–6. Ieee, 2017.

[3] S. Abhishek Anand and Nitesh Saxena. A sound for a sound: Mitigating acoustic side channel attacks on password keystrokes with active sounds. In Jens Grossklags and Bart Preneel, editors, *Financial Cryptography and Data Security - 20th International Conference, FC 2016, Christ Church, Barbados, February 22-26, 2016, Revised Selected Papers*, volume 9603 of *Lecture Notes in Computer Science*, pages 346–364. Springer, 2016. doi: 10.1007/978-3-662-54970-4_21. URL https://doi.org/10.1007/978-3-662-54970-4_21.

[4] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016. URL http://arxiv.org/abs/1611.02167.

[5] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[6] 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.

[7] 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.

[8] MT Camacho Olmedo, Martin Paegelow, Jean-François Mas, and Francisco Escobar. Geomatic approaches for modeling land change scenarios. an introduction. In *Geomatic Approaches for Modeling Land Change Scenarios*, pages 1–8. Springer, 2018.

[9] Rich Caruana and Alexandru Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd international conference on Machine learning*, pages 161–168, 2006.

[10] Rich Caruana, Steve Lawrence, and C. Lee Giles. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In Todd K. Leen, Thomas G. Dietterich, and Volker Tresp, editors, *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO,*

USA, pages 402–408. MIT Press, 2000. URL https://proceedings.neurips.cc/paper/2000/hash/059fdcd96baeb75112f09fa1dcc740cc-Abstract.html.

[11] 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.

[12] Suresh Chari, Josyula R Rao, and Pankaj Rohatgi. Template attacks. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 13–28. Springer, 2002.

[13] Nitesh V. Chawla, Kevin W. Bowyer, Lawrence O. Hall, and W. Philip Kegelmeyer. SMOTE: synthetic minority over-sampling technique. *J. Artif. Intell. Res.*, 16:321–357, 2002. doi: 10.1613/jair.953. URL https://doi.org/10.1613/jair.953.

[14] Yann N. Dauphin, Harm de Vries, Junyoung Chung, and Yoshua Bengio. Rmsprop and equilibrated adaptive learning rates for non-convex optimization. *CoRR*, abs/1502.04390, 2015. URL http://arxiv.org/abs/1502.04390.

[15] Li Deng and John C. Platt. Ensemble deep learning for speech recognition. In Haizhou Li, Helen M. Meng, Bin Ma, Engsiong Chng, and Lei Xie, editors, *INTERSPEECH 2014, 15th Annual Conference of the International Speech Communication Association, Singapore, September 14-18, 2014*, pages 1915–1919. ISCA, 2014. URL http://www.isca-speech.org/archive/interspeech_2014/i14_1915.html.

[16] A. Adam Ding, Liwei Zhang, Yunsi Fei, and Pei Luo. A statistical model for higher order DPA on masked devices. In Lejla Batina and Matthew Robshaw, editors, *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, pages 147–169. Springer, 2014. doi: 10.1007/978-3-662-44709-3_9. URL https://doi.org/10.1007/978-3-662-44709-3_9.

[17] Rob A Dunne and Norm A Campbell. On the pairing of the softmax activation and cross-entropy penalty functions and the derivation of the softmax activation function. In *Proc. 8th Aust. Conf. on the Neural Networks, Melbourne*, volume 181, page 185. Citeseer, 1997.

[18] M.W Gardner and S.R Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric Environment*, 32(14): 2627–2636, 1998. ISSN 1352-2310. doi: https://doi.org/10.1016/S1352-2310(97)00447-0. URL https://www.sciencedirect.com/science/article/pii/S1352231097004470.

[19] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon, David B. Dunson, and Miroslav Dudík, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011. URL http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

[20] Ling Guan, Lei Gao, Nour El-Din El-Madany, and Chengwu Liang. Statistical machine learning vs deep learning in information fusion: Competition or collaboration? In *IEEE 1st Conference on Multimedia Information Processing and Retrieval, MIPR 2018, Miami, FL, USA, April 10-12, 2018*, pages 251–256. IEEE, 2018. doi: 10.1109/MIPR.2018.00059. URL http://doi.ieeecomputersociety.org/10.1109/MIPR.2018.00059.

[21] Abdul Mueed Hafiz and Ghulam Mohiuddin Bhat. A survey of deep learning techniques for medical diagnosis. In *Information and communication technology for sustainable development*, pages 161–170. Springer, 2020.

[22] AMA Hawamleh, Almuhannad Sulaiman M Alorfi, Jassim Ahmad Al-Gasawneh, and Ghada Al-Rawashdeh. Cyber security and ethical hacking: The importance of protecting user data. *Solid State Technology*, 63(5):7894–7899, 2020.

[23] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: a survey. *Journal of Cryptographic Engineering*, 10 (2):135–162, 2020.

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

[25] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

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

[27] Michael Hutter and Jörn-Marc Schmidt. The temperature side channel and heating fault attacks. In *International Conference on Smart Card Research and Advanced Applications*, pages 219–235. Springer, 2013.

[28] Minhui Jin, Mengce Zheng, Honggang Hu, and Nenghai Yu. An enhanced convolutional neural network in side-channel attacks and its visualization. *CoRR*, abs/2009.08898, 2020. URL https://arxiv.org/abs/2009.08898.

[29] Sengim Karayalcin and Stjepan Picek. Resolving the doubts: On the construction and use of resnets for side-channel analysis. Cryptology ePrint Archive, Paper 2022/963, 2022. URL https://eprint.iacr.org/2022/963. https://eprint.iacr.org/2022/963.

[30] 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 Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):148–179, 2019. doi: 10.13154/tches.v2019.i3.148-179. URL https://doi.org/10.13154/tches.v2019.i3.148-179.

[31] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

[32] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 971–980, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/5d44ee6f2c3f71b73125876103c8f6c4-Abstract.html.

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

[34] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 20–33. Springer, 2015.

[35] Shutao Li, Weiwei Song, Leyuan Fang, Yushi Chen, Pedram Ghamisi, and Jón Atli Benediktsson. Deep learning for hyperspectral image classification: An overview. *IEEE Trans. Geosci. Remote. Sens.*, 57(9):6690–6709, 2019. doi: 10.1109/TGRS.2019.2907932. URL https://doi.org/10.1109/TGRS.2019.2907932.

[36] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, volume 10076 of *Lecture Notes in Computer Science*, pages 3–26. Springer, 2016. doi: 10.1007/978-3-319-49445-6_1. URL https://doi.org/10.1007/978-3-319-49445-6_1.

[37] Prerna Mahajan and Abhishek Sachdeva. A study of encryption algorithms aes, des and rsa for security. *Global Journal of Computer Science and Technology*, 2013.

[38] James L Massey. Guessing and entropy. In *Proceedings of 1994 IEEE International Symposium on Information Theory*, page 204. IEEE, 1994.

[39] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected AES implementation on ARM. *IACR Cryptol. ePrint Arch.*, page 592, 2021. URL https://eprint.iacr.org/2021/592.

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

[41] Amir Moradi. Side-channel leakage through static power. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 562–579. Springer, 2014.

[42] James Nechvatal, Elaine Barker, Lawrence Bassham, William Burr, Morris Dworkin, James Foti, and Edward Roback. Report on the development of the advanced encryption standard (aes). *Journal of Research of the National Institute of Standards and Technology*, 106(3):511, 2001.

[43] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning. *CoRR*, abs/1712.04621, 2017. URL http://arxiv.org/abs/1712.04621.

[44] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 4095–4102. IEEE, 2017. doi: 10.1109/IJCNN.2017.7966373. URL https://doi.org/10.1109/IJCNN.2017.7966373.

[45] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 4095–4102. IEEE, 2017.

[46] 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 Cryptol. ePrint Arch.*, page 476, 2018. URL https://eprint.iacr.org/2018/476.

[47] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In Anupam Chattopadhyay, Chester Rebeiro, and Yuval Yarom, editors, *Security, Privacy, and Applied Cryptography Engineering - 8th International Conference, SPACE 2018, Kanpur, India, December 15-19, 2018, Proceedings*, volume 11348 of *Lecture Notes in Computer Science*, pages 157–176. Springer, 2018. doi: 10.1007/978-3-030-05072-6_10. URL https://doi.org/10.1007/978-3-030-05072-6_10.

[48] Emmanuel Prouff, Rémi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptol. ePrint Arch.*, page 53, 2018. URL http://eprint.iacr.org/2018/053.

[49] Nurul Amirah Abdul Rahman, I Sairi, NAM Zizi, and Fariza Khalid. The importance of cybersecurity education in school. *International Journal of Information and Education Technology*, 10(5):378–382, 2020.

[50] Christian Rechberger and Elisabeth Oswald. Practical template attacks. In *International Workshop on Information Security Applications*, pages 440–456. Springer, 2004.

[51] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):677–707, 2021. doi: 10.46586/tches.v2021.i3.677-707. URL https://doi.org/10.46586/tches.v2021.i3.677-707.

[52] Matthieu Rivain. On the exact success rate of side channel analysis in the gaussian model. In *International Workshop on Selected Areas in Cryptography*, pages 165–183. Springer, 2008.

[53] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 413–427. Springer, 2010.

[54] Edward F Schaefer. A simplified data encryption standard algorithm. *Cryptologia*, 20(1): 77–84, 1996.

[55] Kai Schramm and Christof Paar. Higher order masking of the aes. In *Cryptographers' track at the RSA conference*, pages 208–225. Springer, 2006.

[56] Sagar Sharma, Simone Sharma, and Anidhya Athaiya. Activation functions in neural networks. *towards data science*, 6(12):310–316, 2017.

[57] Dinggang Shen, Guorong Wu, and Heung-Il Suk. Deep learning in medical image analysis. *Annual review of biomedical engineering*, 19:221–248, 2017.

[58] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 443–461. Springer, 2009.

[59] O-X Standaert, Eric Peeters, Gaël Rouvroy, and J-J Quisquater. An overview of power analysis attacks against field programmable gate arrays. *Proceedings of the IEEE*, 94 (2):383–394, 2006.

[60] Biaoshuai Tao and Hongjun Wu. Improving the biclique cryptanalysis of aes. In *Australasian Conference on Information Security and Privacy*, pages 39–56. Springer, 2015.

[61] Jasper G. J. van Woudenberg, Marc F. Witteman, and Bram Bakker. Improving differential power analysis by elastic alignment. In Aggelos Kiayias, editor, *Topics in Cryptology - CT-RSA 2011 - The Cryptographers' Track at the RSA Conference 2011, San Francisco, CA, USA, February 14-18, 2011. Proceedings*, volume 6558 of *Lecture Notes in Computer Science*, pages 104–119. Springer, 2011. doi: 10.1007/978-3-642-19074-2_8. URL https://doi.org/10.1007/978-3-642-19074-2_8.

[62] Yoo-Seung Won, Xiaolu Hou, Dirmanto Jap, Jakub Breier, and Shivam Bhasin. Back to the basics: Seamless integration of side-channel pre-processing in deep neural networks. *IEEE Trans. Inf. Forensics Secur.*, 16:3215–3227, 2021. doi: 10.1109/TIFS.2021.3076928. URL https://doi.org/10.1109/TIFS.2021.3076928.

[63] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(3):147–168, 2020. doi: 10.13154/tches.v2020.i3.147-168. URL https://doi.org/10.13154/tches.v2020.i3.147-168.

[64] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IACR Cryptol. ePrint Arch.*, page 1293, 2020. URL https://eprint.iacr.org/2020/1293.

[65] Lichao Wu, Léo Weissbart, Marina Krček, Huimin Li, Guilherme Perin, Lejla Batina, and Stjepan Picek. On the attack evaluation and the generalization ability in profiling side-channel analysis. Cryptology ePrint Archive, Report 2020/899, 2020. https://ia.cr/2020/899.

[66] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014*, pages 719–732. USENIX Association, 2014. URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom.

[67] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):1–36, 2020. doi: 10.13154/tches.v2020.i1.1-36. URL https://doi.org/10.13154/tches.v2020.i1.1-36.

[68] Zixing Zhang, Jürgen T. Geiger, Jouni Pohjalainen, Amr El-Desoky Mousa, Wenyu Jin, and Björn W. Schuller. Deep learning for environmentally robust speech recognition: An overview of recent developments. *ACM Trans. Intell. Syst. Technol.*, 9(5):49:1–49:28, 2018. doi: 10.1145/3178115. URL https://doi.org/10.1145/3178115.

[69] Ziyue Zhang, A Adam Ding, and Yunsi Fei. A fast and accurate guessing entropy estimation algorithm for full-key recovery. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 26–48, 2020.

[70] Yuanyuan Zhou and François-Xavier Standaert. Deep learning mitigates but does not annihilate the need of aligned traces and a generalized resnet model for side-channel attacks. *J. Cryptogr. Eng.*, 10(1):85–95, 2020. doi: 10.1007/s13389-019-00209-3. URL https://doi.org/10.1007/s13389-019-00209-3.