



Deep Learning Side-Channel Collision Attack

Marvin Staib  and Amir Moradi 

Ruhr University Bochum, Horst Görtz Institute for IT Security, Bochum, Germany
firstname.lastname@rub.de

Abstract. With the breakthrough of Deep Neural Networks, many fields benefited from its enormously increasing performance. Although there is an increasing trend to utilize Deep Learning (DL) for Side-Channel Analysis (SCA) attacks, previous works made specific assumptions for the attack to work. Especially the concept of template attacks is widely adapted while not much attention was paid to other attack strategies. In this work, we present a new methodology, that is able to exploit side-channel collisions in a black-box setting. In particular, our attack is performed in a non-profiled setting and requires neither a hypothetical power model (or let’s say a many-to-one function) nor details about the underlying implementation. While the existing non-profiled DL attacks utilize training metrics to distinguish the correct key, our attack is more efficient by training a model that can be applied to recover multiple key portions, e.g., bytes. In order to perform our attack on raw traces instead of pre-selected samples, we further introduce a DL-based technique that can localize input-dependent leakages in masked implementations, e.g., the leakages associated to one byte of the cipher state in case of AES. We validated our approach by targeting several publicly available power consumption datasets measured from implementations protected by different masking schemes. As a concrete example, we demonstrate how to successfully recover the key bytes of the ASCAD dataset with only a single trained model in a non-profiled setting.

Keywords: Deep-Learning · Side-Channel Analysis · Side-Channel Collision Attack · Non-Profiled Attack · Masking

1 Introduction

Since 1999, it has been known that the security of a device can be compromised by analyzing its power consumption during sensitive calculations [Koc96]. Compared to cryptanalytical attacks, the so-called Side-Channel Analysis (SCA) attacks are also able to reveal the secret of devices that implement mathematically secure cryptographic primitives, e.g., block ciphers. However, the success of an SCA attack mainly depends on the implementation details, especially with respect to the employed countermeasures. These countermeasures are continuously being improved, leading to new kinds of attacks as well.

There are two main SCA attack categories, covering different attacker models. Profiled attacks assume an adversary, who not only has access to the actual target device, but also to a similar device under his control. This allows him to model the physical behavior of the device for all possible secret intermediates during a so-called profiling phase. Subsequently, he can compare these models with the measurements collected from the target device (i.e., the attack phase). Typical examples for such an attack are Template Attacks [CRR02] and Machine Learning (ML)-based attacks [HGM⁺11]. Non-profiled attacks, on the other hand, deal only with the target device itself and usually require additional information or assumptions, e.g., a hypothetical power model or details of the actual implementation. Common attacks are Correlation Power Analysis (CPA) [BCO04] and collision SCA attacks [SWP03].

During the past years, Deep Learning (DL) has found its way to the SCA community, where it proves to be very powerful. DL allows to automatically detect complex dependencies of SCA measurements to sensitive values and takes over the task of extracting important features, e.g., points of interest [MOP07]. Due to the full connection of multiple layers in the network architecture, the combination of arbitrary points is considered, hence, the network is able to find dependencies that are spread over different points in time. During the learning process, the model outputs are compared with the given labels and the network's internal parameters are updated. This makes these attacks especially promising against implementation of higher-order masking schemes with multivariate leakages. Typically, the concept of Template Attacks is adapted by modelling the physical observations for each intermediate value during the training phase. Afterwards, the secret intermediate is determined by a prediction over the measurements collected from the target device.

1.1 Motivation and Related Works

A well established and popular SCA attack on power traces was published in 2004 [BCO04]. The authors modelled the power consumption of a device for all possible key candidates and determined the correct key through a correlation between the modeled and actual power values. This method has shown to be very powerful, but requires details on the leakage behaviour of the underlying implementation. However, it has been observed in practice that the power consumption of unprotected (i.e., not masked) implementations can be relatively easily predicted by Hamming weight (HW) and Hamming distance (HD) models.

A combination of SCA attacks and collision attacks has been introduced in [SWP03] by making use of SCA information for detecting collisions in the internal state of an AES implementation. This benefits from the the advantage of not relying on any power model. While this type of attacks originally covered only single collisions between two portions of SCA traces, they have been improved later in [MME10] and [MS16] to cover all possible collisions in the whole measurement set.

The application of ML for SCA was first shown in [HGM⁺11] and the follow-up works adapted the concept of Template Attacks for ML approaches. For example, the authors of [MPP16] modelled the traces with DL in a profiled setting and the authors of [CDP17] utilized the translation-invariance property of Convolutional Neural Networks (CNNs) against misaligned traces.

These successful DL attacks initially required to have access to a profiling device, i.e., to substitute Template Attacks. Alternatively, Timon [Tim19] demonstrated how to utilize DL in a non-profiled setting by adapting the concept of Differential Power Analysis (DPA). Instead of training only a single model, this attack trains a model for every possible key candidate and determines the correct key as the one with the best training metrics. The authors of [KHK22] later reduced the complexity of Timon's attack by combining multiple networks and share common layers. Although this approach successfully recovers the key of some implementations, it relies on information about the position of exploitable leakages, and needs to be done considering a part of the targeted intermediate value (e.g., a single bit like DPA) or a hypothetical power model (e.g., HW model like CPA), in short a many-to-one function.

In the ML community, it is common to evaluate the performance of neural networks by means of publicly available datasets. The authors of [PSB⁺18] aimed to provide a new SCA dataset, i.e., ASCAD, as a reference for testing ML approaches. To improve the efficiency of the attacks, Prouff et al. provided a limited ASCAD dataset that only contains the relevant sample points for attacking a certain key byte of an AES implementation. Most attacks focused on this dataset. To the best of our knowledge, the only work covering the full dataset is the one published at CHES 2021 [LZC⁺21]. While the first ASCAD dataset contains only measurements with a fixed key (ASCAD fix), another dataset with a variable key (ASCAD variable) was published in 2019. The authors of [EST⁺22] presented

a detailed analysis on both datasets, where they determined the leakages for all cipher state bytes. Afterwards, they performed a cross-byte analysis by recovering several key bytes with a single trained model still in a profiled setting.

We would like to highlight that most of the relevant research focused on minimizing the number of required traces for recovering the secret key. To compare the performance of different approaches, publicly available datasets are used, which are often tailored to cover only the sensitive operations, e.g., ASCAD dataset as stated above. A black-box adversary has neither access to a profiling device nor information on the operations that are performed at a specific point in time. Hence, existing DL SCA attacks may not be practical when considering such a black-box adversary model.

1.2 Our Contributions

In this work we present a new and efficient attack on masked implementations by exploiting side-channel collisions with DL. Our method does not require a profiling phase and can be performed without any prior implementation details. More precisely, we consider a black-box scenario where the attacker has only access to raw power traces and their corresponding inputs, e.g., plaintexts. By combining the advantages of collision attacks with the power of neural networks, our approach does not rely on any power model and is able to exploit multivariate higher-order leakages without pre-processing the traces. More importantly, compared to the other non-profiled DL SCA attacks [Tim19, KHK22], our approach relaxes the requirements by avoiding any needs for a many-to-one function. To identify potentially colliding sections in the raw traces, we show a method that is able to localize the leakage associated to input-dependent intermediate values, e.g., SBox inputs. Again, no knowledge of secret parameters is required and having access to traces and corresponding plaintexts suffice to mount the attack. We demonstrate the power of our technique by successfully attacking the ASCAD dataset with only a single model in non-profiled settings. Due to its underlying relaxed attacker model, our attack is easily applicable to other implementations with little adjustment and adaptation.

1.3 Outline

The paper is organized as follows. In Section 2, we provide the reader with general information regarding SCA attacks and countermeasures, that are necessary to follow the rest of the paper. We also give a brief introduction to the concepts of DL before explaining how to apply them to our attack in Section 3. We put our methodology into practice and present the results in Section 4, and finally conclude our research in Section 5.

2 Preliminaries

2.1 Masking

To protect cryptographic implementations against SCA attacks, a common technique is to cut the relation between sensitive intermediate values and the measurable physical properties. Masking has become the most adopted countermeasure due to its sound mathematical basis. Since its introduction in 1999 [CJRR99], various improvements on the efficiency and effectiveness of masking have been made [ISW03, GPQ11, BDGN13]. The concept of masking is based on *secret sharing*, where a sensitive intermediate is split into several individually independent and random shares. Boolean masking is the most common masking scheme and can be easily applied to linear functions L as $L(a) \oplus L(b) = L(a \oplus b)$ with \oplus being addition in $\text{GF}(2)^n$ for some integer n . Non-linear operations do not provide this property which makes their masking more challenging. The simplest Boolean sharing of a binary random variable x leading to (x_1, x_2) requires a random r such that $x_1 \oplus x_2 = x$ with $x_1 = r$ and $x_2 = x \oplus r$. Note that r , which is as large as x , should be taken from a

uniform distribution at random. Further, this is trivially extended to higher orders, i.e., a higher number of shares, requiring $d - 1$ random and independent variables r_1, \dots, r_{d-1} to represent x with d shares x_1, \dots, x_d .

All linear cryptographic operations are then performed on each share independently, resulting in randomized intermediate values. This ensures that SCA leakages cannot be correlated with the sensitive values and hence preventing first-order SCA. Following the concept of secret sharing, having access to all shares is necessary to reconstruct x . In a d^{th} -order secure implementation, any combination of d intermediate values should be independent of any sensitive variable. This naturally leads to a minimum of $d + 1$ shares. In other words, a combination of $d + 1$ leakage points (in line with either univariate or multivariate higher-order attacks [Mes00, WW04]) is expected to exploit the leakage and recover the secrets of a d^{th} order secure implementation. Note that finding suitable leakage points to combine is not a trivial task, especially with no or limited knowledge on the implementation under attack. It is noteworthy that the number of required traces in higher order attacks increases exponentially with d [PRB09] but only if the Signal-to-Noise Ratio (SNR) is low enough. Otherwise, a successful attack might be not possible using a feasible number of traces.

While application of Boolean masking to linear functions is trivial, the majority of relevant research activities focused on providing an efficient protection scheme for non-linear functions like the SBox of block ciphers. A comparably easy technique is called *table re-computation* where a masked version of an SBox is pre-calculated before the actual round functions are called. When being limited to Boolean operations, such a pre-computation is done as follow.

$$\forall x, \quad S^*(x) = S(x \oplus r) \oplus r', \quad (1)$$

with $S(x)$ being the SBox result of x , where r and r' stand for the input and output masks, respectively. Afterwards, if the SBox input x is masked with r , $S^*(x)$ can be used to derive the SBox output masked with r' .

2.2 Side-Channel Collision Attack

Classical SCA attacks exploit the relation between actual SCA measurements and some hypothetical models, e.g., HW power model. In contrast, SCA collision attacks do not require prior knowledge on the leakage behavior of the underlying implementation. The concept is based on the assumption that similar operations on the same data lead to similar physical behaviour, e.g., similar power consumption patterns. To illustrate the underlying concept, consider the `SubBytes` operation of the AES, where the same SBox is applied on all cipher state bytes. Observing the same power consumption pattern for two different SBox operations indicates that the same data have been processed. If this assumption is true and the underlying SBox is a bijection, one can write:

$$\begin{aligned} S(p_1 \oplus k_1) &= S(p_2 \oplus k_2) \\ \iff p_1 \oplus k_1 &= p_2 \oplus k_2 \\ \iff p_1 \oplus p_2 &= k_1 \oplus k_2 \end{aligned} \quad (2)$$

Known as linear collision attack [Bog07], this can lead to the linear difference between two key portions k_1 and k_2 . This attack is usually performed in a chosen-plaintext setting and is strongly affected by noise, since collision between two distinct power patterns associated to two serially-processed SBoxes should be detected.

However, other forms of SCA collision attacks are presented in [MME10] and [MS16], where the whole measurement set is used to find all possible collisions. More precisely, such attacks predict the key difference $\Delta_{k_1, k_2} = k_1 \oplus k_2$ and classify the traces into 2^n classes (with n being the SBox input size) once based on the first plaintext portion p_1 , and once more based on $p_2 \oplus \Delta_{k_1, k_2}$. The next, some statistical features of each class, e.g.,

various statistical moments like mean, variance, skewness, etc., are predicted. The most probable key difference Δ_{k_1, k_2} should lead to the most similarity between the extracted features. These attacks – unlike classical SCA collision attacks – can be applied at higher orders and can exploit the leakage of parallelized hardware implementations, i.e., when SBoxes are performed simultaneously.

2.3 Machine Learning / Deep Learning

2.3.1 Overview

ML has gained a substantial amount of popularity over the past few decades, and is applied in a wide range of fields, e.g., speech recognition, self-driving cars, or targeted advertising. In general, ML can be divided into unsupervised and supervised learning. Unsupervised learning processes rely on the raw input data x (also called *features*) while no further inputs are required. In contrast, supervised methods rely on additional labels y during the training process which already contain the correct answer to the problem that is to be solved. These labels can be difficult to obtain, since they require a human supervisor. The training process ends with a model that can be then used to either predict or classify new data. In this work, we focus on multi-class classifications, where the output of a classifier provides a probability for each discrete class, for example, for all possible key values associated to an SCA trace.

Before the model can be trained, the dataset needs to be split into training, validation and test sets to make the learning process more effective. The *training set* usually is the largest subset and used to actually learn the hidden patterns of the data. To evaluate the model's performance, the *validation set* is used to calculate the correctness of the current parameters and how they should be adjusted to improve the prediction accuracy. Both steps take place repeatedly and during each repetition, also called *epoch*, the whole datasets are iterated. The training data is divided into smaller *batches*, such that the network parameters can be updated multiple times during a single epoch. This ensures a faster training process and lower memory requirements, as only a part of the whole dataset is evaluated at the same time.

Finding the optimal values for a model that generalizes well on new data is a challenging task, and two scenarios may occur. The learning algorithm may start to memorize random fluctuations and noise when being trained for too many epochs or when the model is overly complex. This results in a model which has a high accuracy on the training set but is not able to handle unseen data that do not have exactly the same characteristics as the learned data. This is referred to as *overfitting*, whereas *underfitting* results from a model that is not able to learn the relevant features of a dataset and hence shows a poor performance on the training data as well. After the learning process is finished, the final performance of the model can be evaluated by feeding it with the data coming from the *test set*. These data have not been seen by the model yet, hence giving a realistic indication on the future accuracy.

Although the parameters get optimized while the model is trained with data, the so-called *hyper-parameters* fully depend on the user's choice and need to be set in advance. Examples include the general structure of a neural network, the function used to determine the model's performance, or to what extent the model should be adjusted after each step.

The goal of a classification function \mathcal{F} is to optimize the parameters $\theta \in \Theta$, such that $\hat{y} = y$ for $\hat{y} = \mathcal{F}(x; \theta)$. To quantify the correctness of the predictions, a loss function \mathcal{L} is used during the training process, giving an indication about the quality of the chosen parameters. Finding the optimal set of parameters can be done by finding $\hat{\theta}$ that leads to a minimal loss:

$$\hat{\theta} = \arg \min_{\theta} \mathcal{L}(\mathcal{F}(\cdot; \theta), x, y) \quad \forall \theta \in \Theta, x \in X, y \in Y, \quad (3)$$

with X being all inputs of the dataset and Y the corresponding correct labels. A commonly used loss function is the categorical crossentropy between the correct label and the output

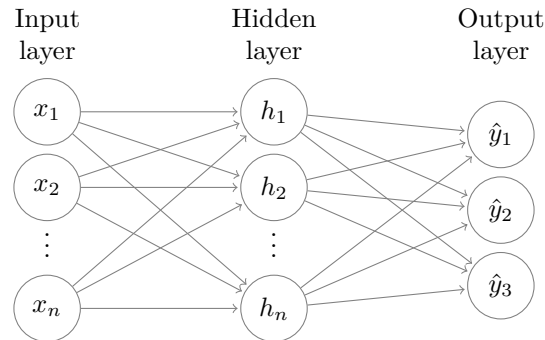


Figure 1: 3-class MLP with a single hidden layer.

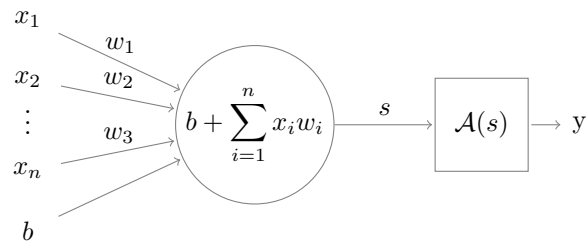


Figure 2: Schematic view of a perceptron.

of \mathcal{F} . Calculating this metric requires the label to be *one-hot* encoded, meaning that it needs to be converted to a bit-vector t of length $|\mathcal{C}|$ where \mathcal{C} is the set of possible labels. For label y in multi-class problems, this vector contains only a single 1 at bit position t_y . For example, when the correct class of a rolled dice is 4, the corresponding one-hot encoded label would be 001000. The categorical crossentropy is defined as

$$\mathcal{L}(t, \hat{y}) = - \sum_{i=1}^{|\mathcal{C}|} t_i \log \hat{y}_i, \quad (4)$$

and averaged over all elements of the validation set to make a statement on the network's performance when instantiated with θ .

2.3.2 Multi-Layer Perceptron

Multi-Layer Perceptrons (MLPs) as a subset of DL are based on neural networks which are inspired by the structure of the human brain. When scientists and industry started to utilize neural networks almost 30 years ago, due to the low computational power of former computers only simple mathematical structures were possible to employ. Classifiers became much more accurate, when more complex networks in the form of MLP were used. These networks not only increase the prediction performance but also allow automatic feature extraction such that less human interventions are required.

MLPs are built by *perceptrons* connected to each other. They are arranged in multiple layers whereas each layer is fully connected to its successor, as shown in **Figure 1**. Each perceptron i has an associated trainable bias b_i , and each connection between two perceptrons i and j has a trainable weight $w_{i,j}$. With these parameters, the output function is calculated as a weighted sum followed by an activation function \mathcal{A} , which is shown in **Figure 2**. \mathcal{A} is a non-linear mapping that decides, to what extent the neurons input influences the prediction process. Typical examples for activation functions proven to be suitable for many tasks are *ReLU* for hidden layers and *softmax* for the output layer.

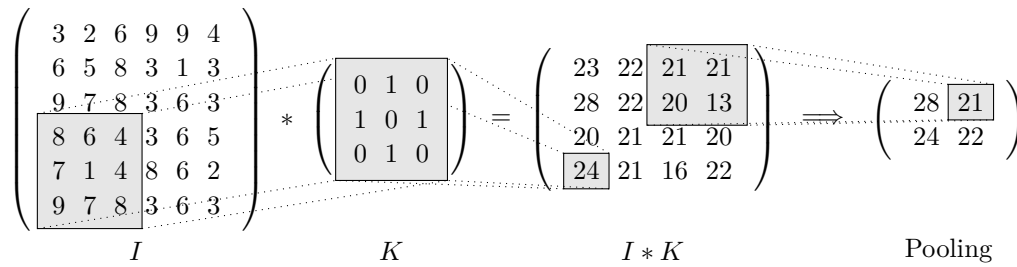


Figure 3: Concept of CNN. Here, the pooling layer is realised by taking the maximum value within each window.

2.3.3 Convolutional Neural Networks

CNNs were initially used to increase the performance of image classifiers, but have become widely applicable to all data with spatial and temporal dependencies. They are a subset of DL and consist of a *convolutional layer* and a *pooling layer*. First, one or several n -dimensional kernels K are shifted over the input data I , and a convolution operation is applied. There are many hyper-parameters the user has to define, e.g., the number and size of the kernel, step size that is used to iterate over the input data (also called *stride*), and the kernel itself. After filtering the data, a non-linear *pooling* operation is performed on the new feature map to reduce its dimension and to summarize the features within a region. An example showing how both layers work is given in Figure 3. Although initially used for higher-dimension inputs such as images, CNNs have shown to be efficient in the context of SCA as well and enable feature detection even for misaligned data.

2.3.4 More Basic Blocks

In the growing field of ML, there exist countless additional layers that can improve the accuracy of a learned model. In this work, we try to keep the underlying model as simple as possible to understand the ongoing processes. Some commonly used basic blocks which are also utilized for our models are summarized as follows.

Dropout can be used during the training phase which disables a random share of connections between two layers [SHK⁺14]. It increases the network’s generalization performance, and reduces overfitting by simulating many different networks without performing the actual training procedure over and over again.

While Dropout is only used during training, and all connections stay untouched after the final model is learned, **Monte-Carlo Dropout (MC-Dropout)** [GG16] applies the same concept to the prediction phase. Instead of getting a single prediction per input, MC-Dropout provides one prediction for each simulated model, such that their average output reduces possible uncertainties a prediction might have.

To make the training of a Neural Network faster and more stable, a technique called **Batch-Normalization** can be applied. Before the input is fed into the next layer, a batch-wise normalization is performed to adjust the values. Hence, each output neuron follows the same distribution such that an issue called *internal covariate shift* is reduced.

Before training a model, the whole dataset is randomly divided into different training, evaluation and test subsets, as explained earlier. To evaluate the model’s performance on new data, the test set is evaluated, but due to the non-determinism when training a model, this result contains a bias. This is due to the random initialization of all weights that might lead to better or worse prediction accuracies. **Cross-Validation** repeatedly splits up the dataset into the corresponding subsets while performing a new training every time. Then, all resulting scores are averaged to give a more general hint on the generalization capabilities of the model.

2.4 ASCAD Dataset

The ASCAD databases were released by ANSSI to allow easy comparisons between SCA attacks. Our targeted dataset is referred to as **ASCADv1** [PSB⁺18] which is publicly available through GitHub¹. It contains 60,000 power measurements of the first round of an SCA-protected AES implementation. In particular, Boolean masking is used for the linear parts and table-recomputation for the **SubBytes** layer, while the implementation was running on an 8-bit AVR ATmega2815 micro-controller. To perform preliminary tests on the dataset, the first two cipher state bytes are only protected with table-recomputation, such that the corresponding cipher state bytes are only weakly protected. Although the implementation is supposed to achieve first-order security, the work presented in [EST⁺22] has shown the existence of first-order leakage for almost all cipher state bytes. To reduce the attack complexity for other researchers, ANSSI already analyzed the leakage behaviour of the third cipher state byte and provided a reduced dataset containing only 700 sample points per trace. Hence, most works on the ASCAD dataset focused on such a reduced set and omitted to find a suitable window of sample points for learning their models.

For our experiments, we use the full dataset with 100,000 samples per trace and a fixed encryption key for all measurements. Note that there also exist two other ASCAD datasets with variable encryption keys following a slightly different measurement scenario. Those sets are not suitable for our attacks, as we assume a non-profiling adversary model, who can only collect SCA measurements with a fixed, but unknown, key.

3 Methodology

3.1 General Idea

In our proposed attack, a DL model M_i is trained with associated information containing the processing of the i -th plaintext portion, e.g., a byte. These information are extracted from the set of power traces $T = \{T^1, \dots, T^{(N)}\}$ (with N being the number of collected traces, containing S sample points each) while only a certain range of sample points ξ_i ($|\xi_i| \ll S$) is involved in the learning process. A suitable target is the calculation of functions, which are repetitively used for processing the cipher state portions. For many block ciphers like AES, the **SubBytes** layer of the first round fulfils this condition. The identification of those functions is performed in multiple steps, and requires to find the ranges ξ_i'' and ξ_i' first, while $|\xi_i| < |\xi_i'| < |\xi_i''|$. More precisely, every ξ_i describes the range, in which the important leakages can be found, with ξ_i' and ξ_i'' denoting coarse-grained ranges before the precise range ξ_i can be determined. In Section 3.2 we explain, how to find these ranges. Since we are in a non-profiled setting, the key-byte k_i is not known. Hence, for training the neural network we use the plaintexts p_i as the label. The trained model is then utilized to predict labels \hat{y} for a different range of sample points ξ_j that come from the same set of traces T (when j -th plaintext portion/byte is processed), such that $\hat{y} = M_i(T[\xi_j])$. As the ranges ξ were chosen, such that the underlying operations are similar, we assume that $\hat{y} \oplus k_i = p_j \oplus k_j$ being an internal (linear) collision. When taking the knowledge of the known labels p_j into account, we are able to recover $\Delta k_{i,j} = k_i \oplus k_j = p_i \oplus p_j$. The concept is visualized in Figure 4.

In total, our approach requires three different neural networks. The first two differ only slightly (see Section 4.3) and are used to identify the leaking ranges ξ_i'' , ξ_i' , and ξ_i . The third network, denoted as M_i , is then used to train an actual model based on the determined range ξ_i .

¹<https://github.com/ANSSI-FR/ASCAD>

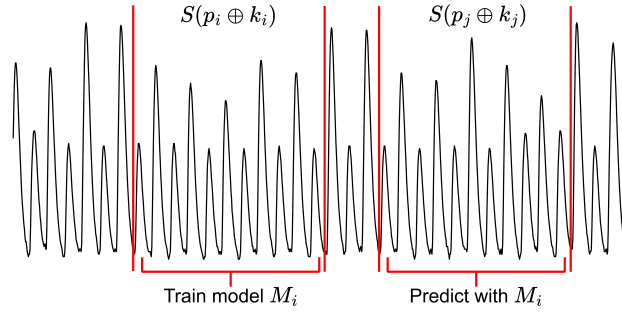


Figure 4: After learning the processing of the first SBox, the model is used to predict labels for another SBox.

3.2 Leakage Identification

A common approach to identify the point in time, where a specific plaintext portion is processed, is to calculate the SNR [MOP07]. With only the plaintexts available, this method works well for implementations with first-order univariate leakage. For properly protected implementations, it requires knowledge of the utilized random masks (in masked implementations), hence, making this metric more suitable for a designer than for an attacker. To find leakage locations without any prior knowledge, we adapted the Sensitivity Analysis (SA) expressed in [Tim19] such that only the plaintexts are needed to be known. The plaintext in masked implementations is usually only processed when being initially loaded and when the random masks m are applied, but not during the actual calculation of the cipher functions. DL can automatically combine the relevant sample points of different shares, e.g., the processing of the masked state $p \oplus m$ and the mask m itself. Hence, the neural network can reveal multivariate leakages that depend on the masked plaintext. Note that a similar dependency between the traces and plaintext can be seen when the SBox is calculated. More precisely, since the key addition (XOR with a secret but constant key) is a one-to-one function, for every plaintext, there is a unique value for the SBox input and its output (assuming the Sbox is a bijection, e.g., AES). Therefore, all leakages relevant to the SBox input (and SBox output) are associated to the plaintext as well.

By performing SA, we want to know, which input features x contribute most to the classification. Therefore, the partial derivatives $f'(T)$ of the loss with regards to the network input is calculated. Formally, this is denoted as

$$f'(T_l^{(n)}) = \frac{\partial \mathcal{L}_{T^{(n)}}}{\partial x_l} \quad \forall l \in \xi'', \forall n \in \{1, \dots, N\}. \quad (5)$$

Subsequently, the derivatives for every sample l are accumulated over the number of traces to obtain the sensitivity. As we do not know, how many epochs the neural network requires until the relevant leakage area is found, the sensitivity values over multiple epochs are accumulated as well. Training the network for too long leads to features being learned that are less relevant for our analysis and potentially result from overfitting. We denote S_i^ξ as the resulting sensitivity when $T[\xi_i]$ is used as the network's input.

To find the optimal hyperparameters, we analyze the influence of the number and size of the hidden layers. Therefore, we perform a grid search with up to four hidden layers with sizes between 10 and 1600. Each hyperparameter-combination is tested three times to reduce the bias caused by the network's random initialization. To assess which set of hyperparameters is the best, manual inspection of the sensitivity results is required. This is due to the non-deterministic behaviour of neural networks which prevents automatic analysis based on maximum sensitivity. Additionally, our investigations showed that there is no correlation between training accuracy and sensitivity result. Evaluating all possible hyperparameter-combinations leads to the conclusion that the neural network's exact

structure is not as important as its general complexity. Networks with either many neurons in the first hidden layers, or a few neurons in the last hidden layer performed equally well.

When training the network, we do not use the full acquisition window, but focus on a smaller range ξ_i'' to reduce the computational complexity. This range must cover the calculation of the target plaintext portion i . Otherwise, the leakage analysis will not show any significant peak, and another range has to be checked. For an implementation without shuffling countermeasures, finding the correct range of each plaintext portion should lead to strictly distinct ranges, as they are processed in a fixed order, e.g., ascending. However, some implementations process the portions in an unknown order, making it more difficult to find the correct ranges.

After obtaining the leakage area ξ_i'' , the experiment is conducted again, but with a smaller range ξ_i' . This gives a more precise result and the final range ξ_i can be extracted by visual inspection. Note that using this approach we do not get any quantitative value about the leakages, but only information on how much specific features contribute to the decision that is made by a model. More specifically, only the most important features related to the given label are used, such that less important features might be ignored by the network.

3.3 Training

After the sensitive leakage area is found, the corresponding sample range ξ_i is trained by another neural network. Again, the known plaintexts are used to label the traces such that $p_i^{(n)}$ is the label for $T^{(n)}[\xi_i]$ with i being the target plaintext portion and n the trace index. We refrain a deeper hyperparameter analysis for this network, and instead use the architectures presented in [RWPP21]. The authors used reinforcement learning for hyperparameter tuning and targeted the ASCAD dataset as well, but in a profiled setting. Their technique was able to find powerful neural networks with a relatively low number of neurons and layers. We tested several of their presented hyperparameters and identified the best results for the network denoted as *ID CNN(RS)* on the ASCAD fixed key dataset without desynchronization. Note that the authors performed their experiments on the reduced ASCAD dataset (see Section 2.4) and focused on recovering the third key-byte only. For other implementations (or even for other targeted bytes of the same implementation) the leakage characteristic might be different, leading to other hyperparameters having an even better performance. Even though the network is biased due to the optimization for a single byte of one specific implementation, we used it throughout all our experiments. We observed that small changes on the network architecture do not significantly change the attack results, and the influence of randomly initializing the internal weights has a much higher impact. Larger differences in the leakage characteristics can, to some extent, also be compensated by increasing or decreasing the number of training iterations, which is confirmed by our experiments. Using a single architecture for all targeted values, we waive the computational effort for hyperparameter tuning on individual plaintext portions and show the general applicability of the chosen network.

In profiled scenarios, the label is usually an intermediate value of the implementation derived by utilizing the associated secret information, e.g., key and mask values. This simplifies the learning process as the relation between intermediate value and power consumption is more obvious. For non-profiled attacks, the attacker does not have access to those information and needs to get along with only the plaintext. Consequently, the network has to learn a more difficult relation, leading to a higher number of iterations to be performed. The training process is stopped as soon as the validation accuracy surpasses the threshold of random guessing, which is $1/2^8$ when data are processed bitwise. By applying this early-stopping approach, we ensure that the network will not be trained longer than required, hence preventing overfitting.

3.4 Attack

For the attack phase, we use the same set of traces T as for the training phase, but with a different range of sample points. We consider two scenarios. The first one assumes a fixed or known offset between the sample ranges associated to different plaintext portions ξ_j . This might be revealed by visual inspection of the traces or by having access to the implementation's source code. During the attack phase, we can then simply shift the leakage range ξ_i to obtain the leaking areas of the other portions. The second scenario follows a black-box setting, where the attacker does not have this knowledge, and the cipher state portions could also be processed at an arbitrary order. Consequently, the leakage analysis must be performed for each plaintext portion separately to find the corresponding ranges for a collision attack. All ranges should cover the same operations, which is difficult to achieve by visual inspection only. Even slightly incorrect (non-compatible) ranges would potentially lead to an unsuccessful attack, although the use of CNNs diminishes this hard requirement. To obtain appropriate ranges more analytically, we estimate the correlation ρ between the sensitivity values $S_i^{\xi'}$ and $b = S_j^{\xi'+\tau}$ for all possible displacements τ as

$$C(\tau) = \rho(S_i^{\xi'}, S_j^{\xi'+\tau}). \quad (6)$$

Subsequently, ξ_j is determined by τ that maximizes C .

As already pointed out, the attack phase utilizes the trained model M_i to make predictions on sample points corresponding to the processing of plaintext portion j . Consequently, the model will output a vector of probabilities $P_v^{(n)}$ for an input trace $T^{(n)}[\xi_j]$, indicating the similarities to all possible plaintext values v ($0, \dots, 255$ in case of AES) that were trained before. We can then obtain

$$\hat{y} = \arg \max_v \left(P_v^{(n)} \right) \quad (7)$$

and predict $\Delta k_{i,j} = \hat{y} \oplus p_j^{(n)}$. As stated in Equation (2), this is only correct when \hat{y} equals $p_i^{(n)}$. In most cases, this condition is not met when only a single trace is considered, but when the entire set of traces is used. This is why we accumulate the resulting probabilities for all given traces and directly adjust the indexing to obtain the following result vector Y

$$Y_v = \sum_{n=1}^N P_{v \oplus p_j^{(n)}}^{(n)}. \quad (8)$$

Then, the key difference can be predicted as

$$k_i \oplus k_j = \arg \max_v (Y_v). \quad (9)$$

Algorithm 1 summarizes all steps of our attack. A common metric to quantify the success rate of a DL-based SCA is the key-rank, which denotes the index of the correct key in a sorted vector Y . The key-rank gives a good estimation about the remaining complexity of a complete key recovery when using key enumeration, especially when the correct key is not ranked first. Another approach is the application of MC-Dropout to quantify the certainty of our result. We run every prediction multiple times and when the same key-difference is obtained for most of the predictions, we assume the correct difference is successfully recovered. This approach does not require knowledge of the correct key and only gives us information about the keys, that have the highest probability in the sorted vector Y . We approved this method with our knowledge of the actual key.

3.5 Advantages Compared to Other Non-Profiled DL Attacks

Most DL SCA attacks adapt the concept of template attacks, which has the disadvantage of requiring a profiling device. The existing non-profiled DL attacks, on the other hand,

Algorithm 1 DL Collision Attack on AES

Input: Traces T , Plaintext-bytes p , Training index i , Deep-Learning architecture DL
Output: Key-differences Δk

- 1: Obtain sensitivity $S_i^{\xi'} \leftarrow \text{LeakageAnalysis}(T, p_i)$
- 2: Determine range ξ_i from $S_i^{\xi'}$ by visual inspection
- 3: Train model $M_i \leftarrow \text{DL}(T[\xi_i], p_i)$
- 4: **for** $j \leftarrow 0$ to 15 **do**
- 5: **if** $j \neq i$ **then**
- 6: Obtain sensitivity $S_j^{\xi'} \leftarrow \text{LeakageAnalysis}(T, p_j)$
- 7: Estimate cross-correlation $C(\tau) \leftarrow \rho(S_i^{\xi'}, S_j^{\xi'+\tau})$
- 8: Determine optimal offset $\theta \leftarrow \arg \max_{\tau} (C(\tau))$
- 9: $Y \leftarrow \emptyset$
- 10: **for** $n \leftarrow 1$ to N **do**
- 11: Predict $P \leftarrow M_i(T^{(n)}[\xi_j + \theta])$
- 12: **for** $v \leftarrow 0$ to 255 **do**
- 13: $Y_v \leftarrow Y_v + P_{v \oplus p_j}$
- 14: **end for**
- 15: **end for**
- 16: $\Delta k_{i,j} \leftarrow \arg \max_v (Y)$
- 17: **end if**
- 18: **end for**
- 19: **return** $\langle \Delta k_{i,0}, \Delta k_{i,1}, \dots, \Delta k_{i,15} \rangle$

are based on a power model [Tim19, KHK22]. More precisely, they need to utilize a many-to-one function, e.g., the HW model. Considering no model, i.e., the identity function (e.g., the 8-bit output of the AES SBox), would result in no success, as all key candidates would become equally likely. Our approach, on the other hand, makes use of no hypothetical model, and deals only with the known (non-secret) information, e.g., plaintexts.

They further only focused on pre-selected sample ranges that were determined by a preceding leakage analysis utilizing secret values [Tim19, KHK22]. This approach might be useful for comparing the performance of different model architectures, but in a black-box scenario the adversary has no access to these information. Our methodology includes a technique how to find leaking areas in masked implementations using only the raw traces and their associated public inputs.

DL typically consists of a training phase and a subsequent prediction phase, where the trained model is applied. While other works deviated from this concept and instead used the training metrics as a distinguisher to find the correct key, we bring back the classical train-then-predict approach. We show that the trained model can be applied not only to recover a single portion of the key, but almost for full key-recovery. More precisely, the method by Timon [Tim19] requires 256 networks to be trained for recovering a single AES key-byte. Although the authors of [KHK22] improved the attack by merging multiple networks into a single neural network, its applicability is still limited to a single key-byte at the time. Furthermore, their architecture can only handle power models that are based on a single bit. Our approach has neither of these requirements as we are, for the first time, exploiting SCA collisions with DL.

4 Results

In our experiments, we made use of the well-studied ASCAD dataset (see Section 2.4), but only used 20,000 traces for all steps. More specifically, we targeted the `SubBytes`

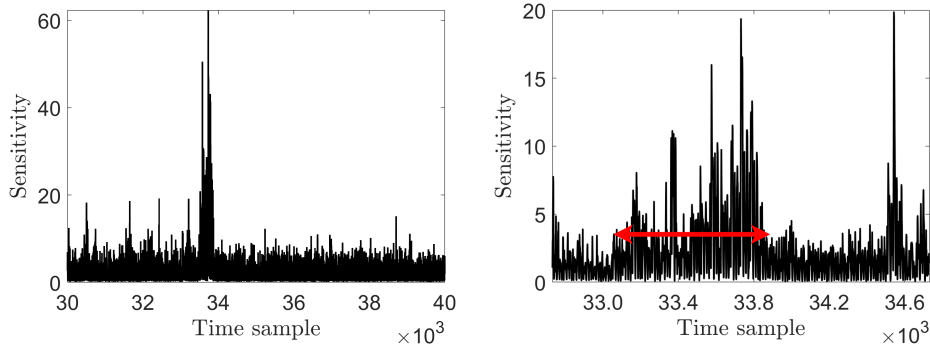


Figure 5: Leakage analysis with identified range ξ_3'' (left) and more detailed analysis with range ξ_3' (right). The determined range ξ_3 is marked by a red arrow.

layer during the first round of AES, since it is used repeatedly for all 16 status bytes. Furthermore, the processing depends directly on the (known) plaintexts used, which are only interposed by the key addition layer.

4.1 Leakage Identification

Training a neural network is highly computationally intensive, especially when the dimensionality of the inputs is large. For the coarse-grained leakage analysis, we trained our network with only 10,000 sample points, such that several ranges had to be checked until a suitable range ξ'' was found. Because the cipher state bytes were not processed in an ascending/descending order, we were not able to infer from one obtained range to the other; hence, all ranges needed to be checked individually. Based on the sensitivity analysis results, a smaller range ξ' of 2,000 points around the peak was selected, and the same procedure was re-performed. Finally, we derived the range ξ by visual inspection. An example of the resulting sensitivity analyses can be seen in Figure 5. The observed locations then indicate, in which order the bytes are processed. Interestingly, for two plaintext bytes, the sensitivity result $S^{\xi'}$ did not show a peak anymore and we had to perform the analysis again by shifting the range ξ' . We assume, the leakage at the initial position depends on other operations within the larger sampling range ξ'' that are not covered by the smaller range ξ'_i . Our goal is to cover the associated leakages within a range ξ_i (as small as possible) to reduce the computational complexity of our neural network, and to ensure that the same type of leakage is covered among all bytes.

4.2 Success Rate

For both, the known offset scenario and the correlation approach (see Section 3.4), we performed an extensive analysis. In Figure 6, we show the key-ranks when Byte 10 was trained and all other bytes being attacked. Our analysis shows that the correct key is either ranked first after a maximum of 600 attack traces, or the attack failed completely. Note that the key-rank does not give any information about the number of required traces for a successful attack, as we always utilize (the same) fixed number of traces during the training phase. From an attacker’s perspective, this metric is not able to detect the cases, where a wrong key was recovered, which is why we focused on the application of MC-Dropout.

We trained the leaking range ξ of every plaintext byte five times, resulting in 70 models. By means of each of them, we attacked all other bytes and averaged the number of successful key recovery attempts. With this approach, we want to overcome the random initialization of each model, leading to different attack results. All networks were trained for up to 500 epochs, but for most bytes the training already stopped earlier due to a sufficiently high validation accuracy. Our experiments clearly show that the accuracy is not

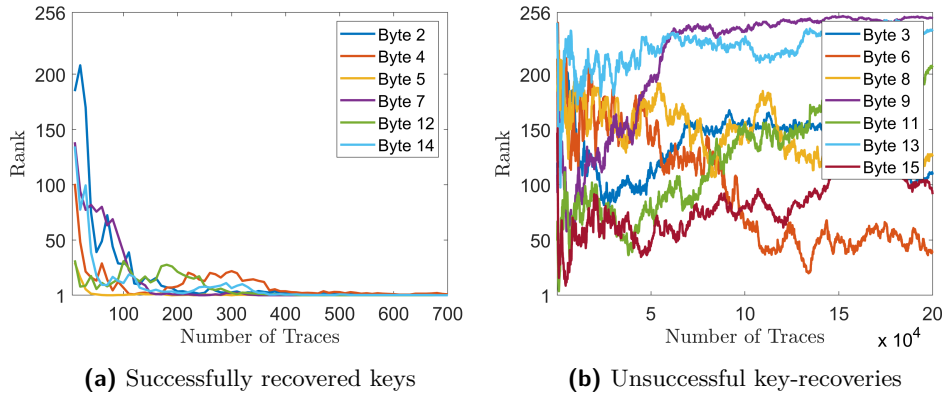


Figure 6: Key-ranks when byte 10 was used for training and all other bytes were targeted with our correlation method.

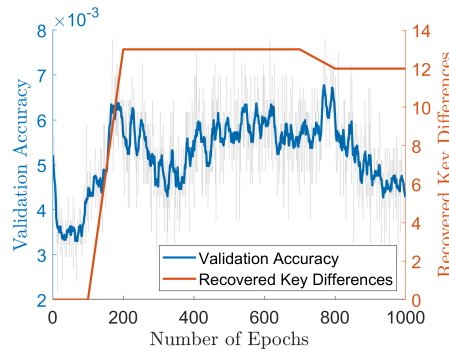


Figure 7: Validation accuracy and number of recovered key differences over the number of trained epochs for byte 10. The blue line shows the moving average of the actual accuracies (grey).

significantly increasing after a certain number of epochs. When the network is still being trained any longer, the number of recoverable key differences start to decrease again, which can be seen exemplary for one trained model in Figure 7. For this analysis, we stored the intermediate model every 100 epochs and subsequently performed the attack. The results show the consequences of overfitting and confirm the advantage of our early-stopping approach. We want to highlight that we recovered the maximum number of key differences in this example, as we did not target the two not-correctly masked state bytes (see our relevant explanation in Section 2.4).

We applied MC-Dropout and obtained three predictions for each attack by randomly dropping 20% of the connections between the fully-connected layers. A model was identified as successful, when two out of three predictions revealed the same key. In Figure 8, we visualize the proportion of trained models that were able recover the correct key differences for all attacked bytes (based on the five trained models per byte). However, in a real-world attack the adversary does not need to recover all key differences, but only the relevant ones², such that a success rate of 7% can already be enough for recovering the whole key. As expected, the results when using known offsets between different bytes were significantly better compared to our correlation method (53% and 25% overall success rate, respectively). In half of the cases of the first scenario, almost all key differences can be recovered with only a single trained model. Five models, on the other hand, were not able to recover any useful key differences. An explanation for this behaviour is given in Section 4.4.

²In such SCA collision attacks on AES-128, at most 15 independent equations (key differences) are required to reduce the key space from 2^{128} to 2^8 .

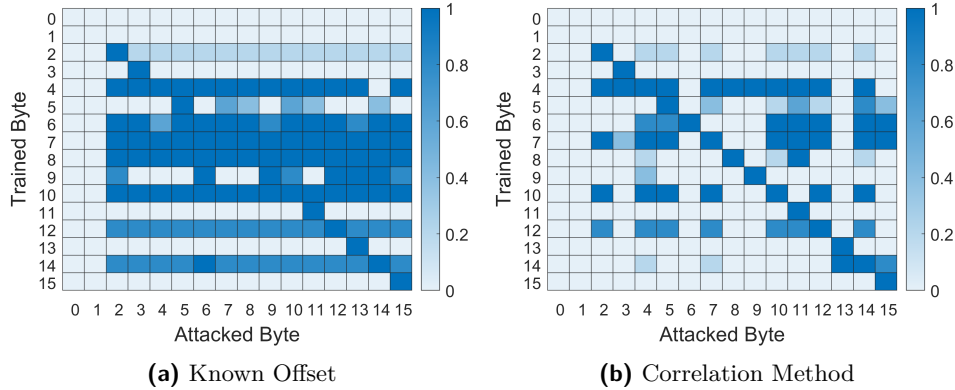


Figure 8: Success rate after the attack phase for all (masked) byte combinations.

With the correlation method, we were also able to recover all key differences, although only the model for byte 4 showed a good overall performance. A minimum of three models is required to get all relevant key differences, although it is unlikely that an attacker chooses the optimal three bytes for training a model. Both scenarios only differ during the attack phase, but use the same trained models. Hence, the models showing a bad performance in the known-offset case also perform bad when using our correlation method. We assume, the relevant leakage features were not represented correctly by the model, which might be caused by choosing wrong ranges ξ . Alternatively, this may happen if the implementation does not execute exact the same operations for every cipher state byte. When calculating the ranges based on a known offset, we can be sure to cover the same operations, and – in an ideal case – cover the same leakage characteristics for all bytes. This makes the attack of one byte independent of its corresponding leakage analysis result. In contrast, the trained and attacked leakages might be different when using the correlation method. Although it can – up to some extent – be compensated by the underlying CNNs, the attack loses its efficiency if the executed operations are fully different, hence no exploitable collisions.

We also performed our attack directly on the ranges ξ without correlating the sensitivity analysis results. This approach relies on visual inspection for all bytes, such that an additional uncertainty is added to the attack phase. Our results confirm this hypothesis, as the success rates were notably worse.

4.3 Complexity

We performed all our experiments on a server equipped with four NVIDIA GeForce RTX 2080Ti GPUs and an Intel Xeon W-3223 CPU. The training process did not perform significantly faster when utilizing multiple GPUs. Hence, we parallelized the attack on multiple bytes. In the following, we show the execution times for all steps while using only a single GPU and 20,000 traces.

- **Identify range ξ'' and ξ' :** Both steps consist of the training itself and calculating the partial derivatives afterwards. To identify ξ'' , the network is trained with a limited number of 10,000 sample points for 20 epochs and a batchsize of 1000, which takes approximately 60 seconds. This step might need to be performed up to 10 times with different ranges until a significant peak in the sensitivity analysis result is found. Range ξ' is identified after 45 seconds when training the corresponding network for 10 epochs and a batchsize of 100.
- **Training phase:** Training the network with traces from range ξ takes up to 14 minutes for 500 epochs. Note that due to our early-stopping approach, only 200-400 epochs are required for most bytes.
- **Attack phase:** We repeated the prediction for each key difference 3 times and

Table 1: Performance comparison between our DL collision attack and the non-profiled DDLA by Timon [Tim19].

	Setting	Traces	Samples	Recovered Keys	Total Run Time
DL collision	Black-Box	20,000	Full Set	13	1h 57min 17s
DL collision	Known Offset	20,000	Full Set	13	14min 46s
DDLA [Tim19]	Known Range	20,000	700	13	1h 20min 10s ³

determined the correct key by majority voting. This took about 25 seconds for each attacked byte.

In a best-case scenario, recovering all 13 key differences takes 15 minutes when byte 10 is selected for training. The corresponding model is able to recover all key differences, i.e., training using only this byte is enough. Additionally, the training is stopped after only 200 epochs due to sufficiently high training accuracy. We assume that the order of the cipher state bytes as well as their offsets to each other are known. Hence, the identification step only needs to be performed once, and all key bytes can be recovered with only five training runs (three models for determining ξ'' since the leakage of byte 10 occurs after approximately 28,000 sample points, one model for finding ξ' and the last model for training ξ). Again, we want to highlight that no additional training is required during the attack phase, as only the predictions for the given ranges of sample points are queried. In a worst-case scenario, the leakage analysis needs to be performed for all cipher state bytes, and the ranges necessary for the attack phase are obtained using our correlation approach. We assume that our attack needs to be performed on five trained models until enough key differences are recovered. Approximately 100 training runs are necessary to recover all relevant key differences, which is done within 1 hour and 57 minutes. In Table 1, we summarize the complexities and compare them with the attack presented in [Tim19]. Note that Timon used the extracted ASCAD dataset, which only contains the processing of the third cipher state byte. His approach is also applicable to other keybytes of the full dataset as long as the corresponding ranges are known.

Compared to the non-profiled DL approach by Timon [Tim19], our models are more complex and require more time to be trained. In return, the number of models is reduced significantly, such that a single model can be sufficient to recover all key differences. More precisely, his method requires 256 trainings to recover a single keybyte without handling the problem of finding suitable leakage areas.

4.4 Accuracy of Leakage Detection

In order to estimate the accuracy of our leakage detection analysis, we compare the retrieved ranges with SNR results. Additionally, we show the ranges found by the authors of [EST⁺22], who used a method based on correlation to find suitable ranges. Using their ranges in our attack leads to the best success rate for all trained bytes, hence, we consider them to be optimal. Our SNR results cover the following two leakage models

$$\begin{aligned} l_1 &= S(p \oplus k) \oplus r_{out}, \\ l_2 &= S(p \oplus k) \oplus r, \end{aligned} \tag{10}$$

where r_{out} denotes the SBox output mask for table re-computation and r is the mask applied to linear operations of the cipher. All results are visualized in Figure 9, whereas the numbers denote the processed byte indices, and the colors are used to differentiate

³Note that the author reported a runtime of 1min 54s per targeted key byte. For a better comparison, we performed their attack with identical parameters on our setup leading to 6min 10s for every 8-bit key recovery attack.

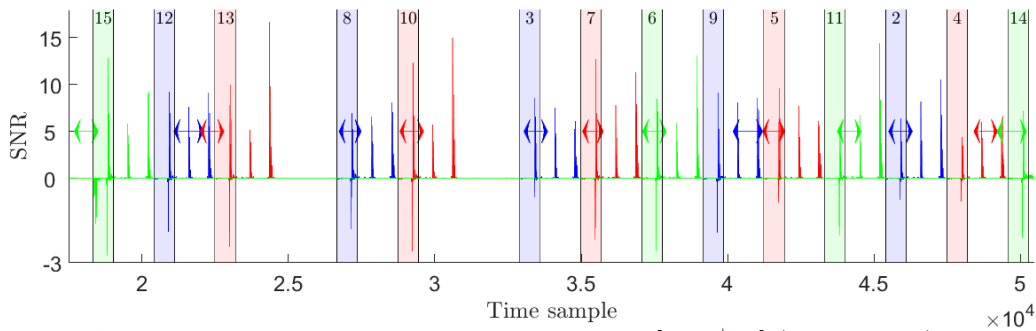


Figure 9: SNR results compared to ranges reported in [EST+22] (shaded area) and our leakage analysis approach (arrows). Positive SNR curves are associated to l_2 and the negative ones to l_1 (see Equation (10)).

the targeted bytes. For a better visibility, we showed the negative of SNR for l_1 to make the difference between the two leakage models more clear. While the ranges reported in [EST+22] perfectly covers the leakages associated to l_1 for all bytes, our approach covers this area only in eight cases. Three bytes match the leakages associated to l_2 , while the analysis of the remaining three bytes match the area identified by neither l_1 nor l_2 . For those models, the success rate for recovering the key differences was 0% for both the correlation approach and the known offset scenario. In some cases, our attack was even unsuccessful although the discovered range matches the actual leakage. We assume that the corresponding models were not able to learn the important leakage features for those bytes and instead learned other features within the same range. Recovering the key differences for the bytes, whose leakage could not be modelled correctly, was only possible using them as the attacked byte with another byte being trained. If the used mask values are known, the SNR is able to precisely capture the leakage of different intermediate values (here, l_1 and l_2). Our method, on the other hand, is performed in a black-box setting where only the plaintexts are known. As we do not know which sample points are combined within the neural network, we cannot make a statement about the type of leakage that is detected. However, the success rate during the attack phase seems to be independent of whether our range matched l_1 or l_2 as we achieved similar results in both cases.

4.5 Comparison with Classical Side-Channel Collision Attacks

For classical SCA attacks, we can usually differentiate between univariate and multivariate attacks. Univariate attacks analyze every point in time individually, such that only leakages evoked from one specific point in time can be detected. In contrast, multivariate attacks combine the information of different sample points in time such that more complex dependencies can be detected, e.g., when time sample t_1 leaks about the mask r and t_2 leaks the masked SBox $S(p \oplus k) \oplus r$. However, performing such an attack requires much effort since (1) correct sample points t_1 and t_2 should be found, and (2) the combination of the leakage points need to be performed by (mean-free) multiplication prior to the actual attack. When there is no information about the leakage distribution available, the attacker would need to test all possible combinations of sample points. Our approach utilizes DL, benefiting from automatically detecting complex dependencies by combining different time samples within the fully-connected layers. We refer to Figure 1, where a simple MLP was shown. The input neurons – referring to sample points – are connected to all neurons of the subsequent layer. When several of such layers are combined, the network can identify which samples points should be combined. Note that to exploit higher-order multivariate leakages the leakages at the corresponding sample points should be multiplied (ideally mean-free product). This indeed happens by the (non-linear) activation function applied at every layer of the network.

To show the power of our collision attack, we also performed a regular (univariate) SCA collision attack. More specifically, we conducted an Moments-Correlating DPA (MC-DPA) [MS16]. To make the attack more efficient, it is beneficial to know the offsets between the processing of different bytes. While we showed a method how to obtain these information without any prior knowledge of sensitive values, we used the ranges reported in [EST⁺22] for MC-DPA as well. We performed first- and second-order MC-DPA attack between all key bytes on the full dataset. Only the latter was able to successfully recover the keys while the first-order attack failed for all masked byte combinations. An example can be seen in Figure 10. These results demonstrate the effectiveness of our DL-based collision attack, which can handle higher-order leakages automatically.

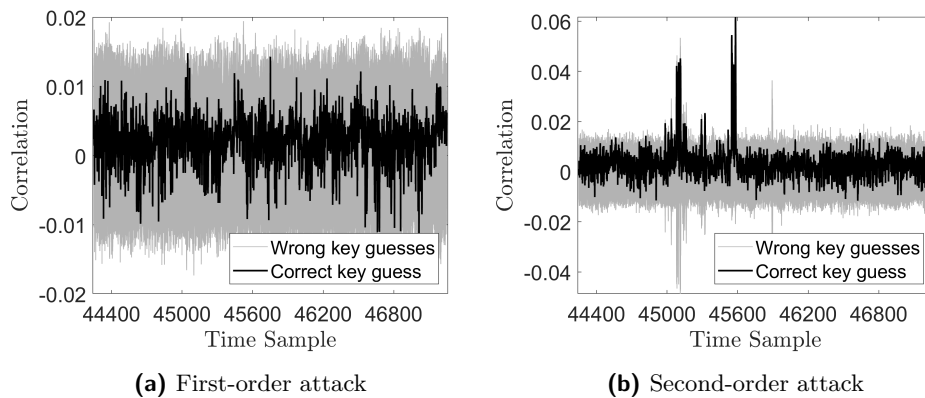


Figure 10: Results of MC-DPA between Byte 2 and Byte 4.

4.6 Applicability to Other Datasets

Our methodology can be easily adapted to other datasets, requiring only slight changes. First, we want to highlight that the neural network M_i for training the leakage area ξ_i does not require any modifications except for the number of training iterations. As we suggest to use our proposed early-stopping approach, the optimal number of epochs can be determined automatically. Compared to many classical attacks, we do not have to cope with different power models as our attack does not require a many-to-one function to successfully exploit leakages. For only weakly protected implementations, it is easier for the SA to find the leaking areas. Consequently, the sample range to analyze can be larger and the SA is still able to detect those areas. This results in smaller ranges that should be tested when the exploitable leakage is expected to be low (e.g., when the implementation is appropriately masked), such that the overall complexity for testing all possible ranges is increased. Additionally, we observed that longer traces benefit from a larger batchsize and vice versa.

ASCAD Variable Key. One year after publishing the ASCAD fixed-key dataset, ANSSI published another dataset with a variable key. It targets the same implementation on the same microcontroller, but with a different measurement setup. While the main goal of the fixed-key dataset was to get a clean signal, a more noisy setup was used for the variable-key dataset, making attacks more challenging⁴. The traces were recorded with a higher sampling rate of 500 MS/s and cover the first two AES encryption rounds. Consequently, the number of sample points per trace is significantly larger than for the first dataset, namely 250,000. In total, the measurement campaign consists of 300,000 traces, whereof 200,000 are intended to be used for profiling and the remaining 100,000 for attacking. These 100,000 traces belong to encryption runs with a fixed key, and hence can be used in

⁴<https://github.com/ANSSI-FR/ASCAD/issues/13>

our approach.

To the best of our knowledge, we are the first attacking this dataset in a non-profiled setting. For reference, we also performed classical MC-DPA and CPA with HW power model. All first-order MC-DPA attacks failed while only key byte 5 was recoverable through CPA. The second-order univariate leakage was successfully exploitable by both attacks, requiring between 10,000 and 30,000 traces depending on the targeted key byte.

We used the same ranges reported in [EST⁺22] to examine the performance of our DL-based collision attack. Again, we did not change any hyperparameters and successfully attacked the implementation. When using all 100,000 traces, the overall success rate was 85% with 11 learned bytes being suitable to recover all key differences. With only 20,000 traces, the success rate shrunk down to 40% with 4 models allowing full key recovery.

DPA Contest v4. The traces from DPA Contest v4 were measured from an Atmel ATmega-163 smart-card executing a masked AES-256 implementation⁵. More precisely, a masking technique called Rotating Sbox Masking (RSM) is applied [NSGD12], which is supposed to be more efficient than classical masking while offering the same level of security. In total, the dataset contains 100,000 traces, each with 435,000 sample points, whereas only the first AES round and the beginning of the second round is covered.

In [MGH14], the authors successfully recovered the secret key with a correlation-collision attack after approximately 2,500 traces of this dataset. For the CPA, classical power models (e.g., HW of SBox input/output) failed while the bit-wise HD between SBox input and output revealed the key after 500 traces. The authors decreased the number of traces even further after some statistical analysis and using a more complex and customized power model. Performing the attack on the full trace would take a considerable amount of time, which is why the authors performed a leakage analysis on all available traces before. The authors of [BBD⁺14] summarized all submitted attacks to the contest and state that the best non-profiled attack requires only 14 traces without giving information about implementation details.

Due to the higher number of sample points compared to the ASCAD dataset, we increased the range ξ'' from 10,000 to 100,000 sample points. Consequently, in a worst-case scenario only 5 ranges had to be tested instead of 44 for the first step of our leakage analysis. To make the learning process more efficient, we down-sampled the given traces for the leakage analysis by discarding every second point. Each clock cycle was then covered by approximately 31 sample points. Compared to our previous experiments, the sensitivity analysis showed a much clearer peak and the range ξ' was chosen to cover 2,000 points. We then determined the final range of 1,000 points ξ by visual inspection. For the leakage analysis, we used batchsizes of 100 and 25 to determine ξ'' and ξ' respectively, and a total of 5,000 traces for training the networks.

The network architecture for training the leaking ranges ξ_i was the same as those used in our analysis of the ASCAD dataset. On average, the network was trained for 60 epochs per byte until the validation accuracy surpassed the threshold for early stopping. With 5,000 utilized traces, we achieved an overall success rate of 84% while 10 models were able to recover all key differences. While our leakage analysis successfully revealed the correct position of each byte with only 2,000 traces, the learning and attack phase required a minimum of 3,000 traces to achieve comparable success rates. As we did not perform any hyperparameter optimizations for this dataset, we assume that the number of required traces can be reduced even further.

DPA Contest v4.2 We also performed our attack on the improved version, i.e., DPA contest v4.2, in which several security flaws of the previous version were fixed [BBD⁺14]. In particular, the implementation utilizes shuffling and masks each state byte individually. The underlying implementation is a 128-bit AES that is executed on the same target as

⁵<https://www.dpacontest.org/v4/>

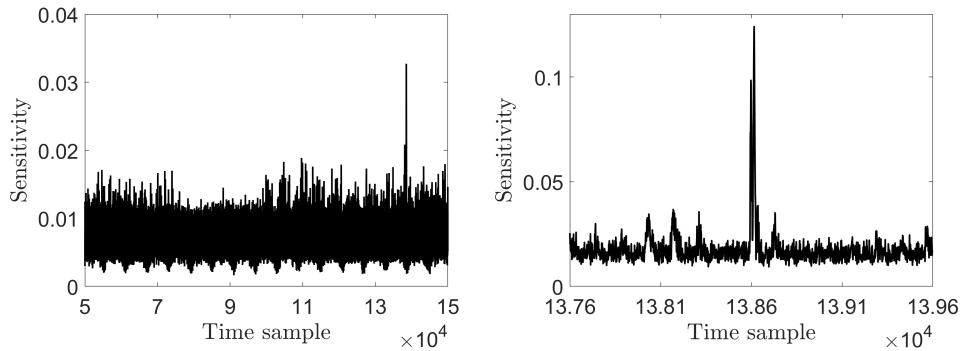


Figure 11: Leakage analysis on dataset from DPA Contest v4 with range of 100,000 sample points (left) and 2,000 sample points (right).

DPA contest v4. In the provided datasets, the secret key changes every 5,000 traces. As our attack relies on a fixed key for the whole dataset, we considered the plaintext XOR key as the input to the implementation, interpreting that the key is set to 0 (only for the first AES round). This allows us to combine more datasets and use more than 5,000 traces in our attack. Furthermore, we assume the shuffling of the byte order is known. Performing our attack on this combined dataset, we achieved an overall success rate of 44% when utilizing 10,000 traces, whereas the training of five bytes allowed to recover all key differences. The corresponding ranges where found by our leakage (sensitivity) analysis requiring approximately 20,000 traces to obtain clear results.

5 Conclusions

In this work, we introduced a novel attack strategy by exploiting SCA collisions using DL in a non-profiled setting. We showed how the adversary can successfully recover the linear difference between the key portions in masked implementations, even in a black-box scenario. More precisely, the attacker neither requires a profiling device (or a profiling dataset) nor implementation details, making the attack suitable for a wide range of real-world scenarios. Most of the former works dealing with DL in the SCA context focus on a small subset of sampling points that exactly cover the exploitable leakage without considering how to obtain these ranges from an attacker’s point of view. Our methodology includes a technique on how to determine the location of leakages in protected implementations without having access to any secret values or any profiling devices. This also solves one of the main disadvantages of SCA collision attacks, namely, the temporal offset between colliding intermediate values. Compared to previous works in the field of non-profiled DL SCA attacks, we significantly reduced the amount of required models for key recovery by successfully performing cross-byte analysis. For the sake of reproducibility, we provided our program code with all relevant parameters in the supplementary material of this submission. The code and the examples are made public through GitHub⁶.

Limitations and Future Works. By training the leakage position for a specific plaintext portion, we rely on the fact that the model is able to learn any plaintext dependent differences in the SCA traces. Adding dummy random operations would add an additional dependency that we do not have any control over and can make the training not as accurate as required. Although this countermeasure might not prevent the attack completely, it will definitely make it more difficult to perform and would lead to more required traces. When an implementation is protected by shuffling, our attack is also not directly applicable, and the adversary may need to reverse engineer the shuffling permutation first. More

⁶https://github.com/ChairImpSec/DL_Collision_Attack

precisely, prior to the collision attack, an independent training and prediction should be first conducted to recover the shuffling of each trace individually.

During our experiments, we only covered software implementations that consecutively process the portions of the cipher state. It is still an open work how such attacks perform on hardware implementations that usually have significantly shorter execution times and many operations (like several SBoxes) are performed in parallel. In hardware, a single round is often performed within one clock cycle, such that we cannot differentiate between the processing of individual portions of the cipher state. Adapting such attacks to exploit cross-round collisions would be an alternative approach to cope with this issue. Even for consecutively processed cipher state portions the attack might fail as the existence of collision leakage is the most important prerequisite. Generally, application of DL SCA attacks on parallelized (round-based) hardware implementations is among our planned works for the future.

Acknowledgments

The work described in this paper has been supported in part by the German Research Foundation (DFG) under Germany's Excellence Strategy - EXC 2092 CASA - 390781972, and by the Federal Ministry of Education and Research of Germany through the Project mINDFUL (16KIS1151).

References

- [BBD⁺14] Shivam Bhasin, Nicolas Bruneau, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. Analysis and Improvements of the DPA Contest v4 Implementation. In *Security, Privacy, and Applied Cryptography Engineering SPACE 2014*, volume 8804, pages 201–218, 2014.
- [BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *CHES 2004*, volume 3156 of *LNCS*, pages 16–29. Springer, 2004.
- [BDGN13] Shivam Bhasin, Jean-Luc Danger, Sylvain Guilley, and Zakaria Najm. A low-entropy first-degree secure provable masking scheme for resource-constrained devices. In *WESS 2013*, pages 7:1–7:10. ACM, 2013.
- [Bog07] Andrey Bogdanov. Improved Side-Channel Collision Attacks on AES. In *SAC 2007*, volume 4876 of *LNCS*, pages 84–95. Springer, 2007.
- [CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Pre-processing. In *CHES 2017*, volume 10529 of *LNCS*, pages 45–68. Springer, 2017.
- [CJRR99] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards Sound Approaches to Counteract Power-Analysis Attacks. In *CRYPTO 1999*, volume 1666 of *LNCS*, pages 398–412. Springer, 1999.
- [CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *CHES 2002*, volume 2523 of *LNCS*, pages 13–28. Springer, 2002.
- [EST⁺22] Maximilian Egger, Thomas Schamberger, Lars Tebelmann, Florian Lippert, and Georg Sigl. A Second Look at the ASCAD Databases. In *COSADE 2022*, volume 13211 of *LNCS*, pages 75–99. Springer, 2022.

- [GG16] Yarin Gal and Zoubin Ghahramani. Dropout as a Bayesian Approximation: Representing Model Uncertainty in Deep Learning. In *ICML 2016*, volume 48 of *JMLR*, pages 1050–1059. JMLR.org, 2016.
- [GPQ11] Laurie Genelle, Emmanuel Prouff, and Michaël Quisquater. Thwarting Higher-Order Side Channel Analysis with Additive and Multiplicative Maskings. In *CHES 2011*, volume 6917 of *LNCS*, pages 240–255. Springer, 2011.
- [HGM⁺11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptogr. Eng.*, 1(4):293–302, 2011.
- [ISW03] Yuval Ishai, Amit Sahai, and David A. Wagner. Private Circuits: Securing Hardware against Probing Attacks. In *CRYPTO 2003*, volume 2729 of *LNCS*, pages 463–481. Springer, 2003.
- [KHK22] Donggeun Kwon, Seokhie Hong, and Heeseok Kim. Optimizing Implementations of Non-Profiled Deep Learning-Based Side-Channel Attacks. *IEEE Access*, 10:5957–5967, 2022.
- [Koc96] Paul C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In *CRYPTO 1996*, volume 1109 of *LNCS*, pages 104–113. Springer, 1996.
- [LZC⁺21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay Attention to Raw Traces: A Deep Learning Architecture for End-to-End Profiling Attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(3):235–274, 2021.
- [Mes00] Thomas S. Messerges. Using Second-Order Power Analysis to Attack DPA Resistant Software. In *CHES 2000*, volume 1965 of *LNCS*, pages 238–251. Springer, 2000.
- [MGH14] Amir Moradi, Sylvain Guilley, and Annelie Heuser. Detecting Hidden Leakages. In *Applied Cryptography and Network Security, ACNS*, volume 8479, pages 324–342, 2014.
- [MME10] Amir Moradi, Oliver Mischke, and Thomas Eisenbarth. Correlation-Enhanced Power Analysis Collision Attack. In *CHES 2010*, volume 6225 of *LNCS*, pages 125–139. Springer, 2010.
- [MOP07] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks - revealing the secrets of smart cards*. Springer, 2007.
- [MPP16] Housseem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking Cryptographic Implementations Using Deep Learning Techniques. In *SPACE 2016*, volume 10076 of *LNCS*, pages 3–26. Springer, 2016.
- [MS16] Amir Moradi and François-Xavier Standaert. Moments-Correlating DPA. In *TIS @ CCS 2016*, pages 5–15. ACM, 2016.
- [NSGD12] Maxime Nassar, Youssef Souissi, Sylvain Guilley, and Jean-Luc Danger. RSM: A small and fast countermeasure for AES, secure against 1st and 2nd-order zero-offset SCAs. In *DATE 2012*, pages 1173–1178. IEEE, 2012.
- [PRB09] Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.

- [PSB⁺18] 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.
- [RWPP21] Jorai Rijdsdijk, 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.
- [SHK⁺14] Nitish Srivastava, Geoffrey E. Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, 2014.
- [SWP03] Kai Schramm, Thomas J. Wollinger, and Christof Paar. A New Class of Collision Attacks and Its Application to DES. In *FSE 2003*, volume 2887 of *LNCS*, pages 206–222. Springer, 2003.
- [Tim19] Benjamin Timon. Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):107–131, 2019.
- [WW04] Jason Waddle and David A. Wagner. Towards Efficient Second-Order Power Analysis. In *CHES 2004*, volume 3156 of *LNCS*, pages 1–15. Springer, 2004.