Non-Profiled Deep Learning-Based Side-Channel Attacks

Benjamin Timon

UL Transaction Security, Singapore
benjamin.timon@ul.com

Abstract. Deep Learning has recently been introduced as a new alternative to perform Side-Channel analysis [1]. Until now, studies have been focused on applying Deep Learning techniques to perform Profiled Side-Channel attacks where an attacker has a full control of a profiling device and is able to collect a large amount of traces for different key values in order to characterize the device leakage prior to the attack. In this paper we introduce a new method to apply Deep Learning techniques in a Non-Profiled context, where an attacker can only collect a limited number of side-channel traces for a fixed unknown key value from a closed device. We show that by combining key guesses with observations of Deep Learning metrics, it is possible to recover information about the secret key. The main interest of this method, is that it is possible to use the power of Deep Learning and Neural Networks in a Non-Profiled scenario. We show that it is possible to exploit the translation-invariance property of Convolutional Neural Networks [2] against de-synchronized traces and use Data Augmentation techniques also during Non-Profiled side-channel attacks. Additionally, the present work shows that in some conditions, this method can outperform classic Non-Profiled attacks as Correlation Power Analysis. We also highlight that it is possible to target masked implementations without leakages combination pre-preprocessing and with less assumptions than classic high-order attacks. To illustrate these properties, we present a series of experiments performed on simulated data and real traces collected from the ChipWhisperer board and from the ASCAD database [3]. The results of our experiments demonstrate the interests of this new method and show that this attack can be performed in practice.

Keywords: side-channel attacks, deep learning, machine learning, non-profiled attacks, profiled attacks

1 Introduction

Side-Channel attacks, introduced in 1996 by P. Kocher [4], exploit side-channel leakages such as power consumption from a device to extract secret information. Side-Channel attacks can be classified into two classes:

– *Profiled* Attacks such as Template Attacks [5], Stochastic attacks [6,7] or Machine-Learning-based attacks [8,9,10].
– *Non-Profiled* Attacks such as Differential Power Analysis (DPA) [11], Correlation Power Analysis (CPA) [12], or Mutual Information Analysis (MIA) [13].

To mount a Profiled Side-Channel attack, one needs to have access to a pair of identical devices: One closed *target* device, with limited control and running a cryptographic operation with a fixed key value $\boldsymbol{k}^* \in \mathcal{K}$, where $\mathcal{K}$ is the set of possible key values. One *profiling* device, with full knowledge and control of the inputs and keys. In such a context, a Profiled Attack is performed in two steps:

1. A profiling phase, while the leakage of the targeted cryptographic operation is profiled for all possible key values $k \in \mathcal{K}$ using side-channel traces collected from the profiling device.
2. An attack phase, where traces collected from the target device are classified based on the leakage profiling in order to recover the secret key value $\boldsymbol{k}^*$.

During the profiling phase, a set of side-channel traces is collected for each possible key value $k \in \mathcal{K}$. Usually, a divide-an-conquer strategy is applied, and one has for instance $\mathcal{K} = \{0, \ldots, 255\}$ meaning that 256 sets of traces are collected to perform the profiling. For Template Attacks, during the profiling phase, one computes a multivariate gaussian model M_k for each possible key value which characterizes the leakage of the cryptographic operation when the key k is used. During the attack phase, we select the model which best fits the side-channel traces collected from the target device to reveal the correct key $\boldsymbol{k}^*$. Profiled attacks are considered as the most powerful form of side-channel attacks as the attacker is able to characterize the side-channel leakage of the device prior to the attack. However, the profiling phase requires to have access to a profiling device, which is a strong assumption that cannot be always met in practice.

Indeed, for *closed* products (for example smart cards running banking applications) an attacker does not have control of the keys and is usually limited by a transaction counter which caps the number of side-channel traces that can be collected. In such a context, Profiled attacks cannot be performed. However, Non-Profiled attacks such as DPA, CPA, or MIA can still threaten the device. The only assumption for Non-Profiled attacks is that the attacker is able to collect several side-channel traces of a cryptographic operation with a fixed unknown key value $\boldsymbol{k}^* \in \mathcal{K}$ and known

random inputs (or outputs) from the targeted device. The attacker then combines key hypotheses with the use of statistical distinguishers such as Pearson's Correlation or Mutual Information to infer information about the secret $\boldsymbol{k}^*$ from the side-channel traces.

Our contribution Recently, Deep Learning has been introduced as an interesting alternative to perform Profiled Side-Channel attacks [1,2]. However, the studies only focused on applying Deep Learning to perform Profiled Side-Channel attacks. As mentioned previously, mounting a Profiled attack requires to have access to a profiling device, which is a strong assumption. In this paper, we present a new side-channel attack method to apply Deep Learning techniques in a Non-Profiled scenario. The main interest of this method, is that it is possible to use Deep Learning techniques and Neural Networks even when profiling is not possible. We show for instance that — as in the Profiled context [2] — it is possible to use the translation-invariance property of Convolutional Neural Networks, as well as Data Augmentation even in a Non-Profiled context. This leads to results showing that in some cases, this attack method can outperform some classic Non-Profiled attacks as CPA. We also study the efficiency of this attack against High-Order protected implementations, and show that this method is able to break masked implementations with a reasonable number of traces and without leakages combination preprocessing. All these points are supported by experiments performed on simulated data and traces from the ASCAD database and collected from the ChipWhisperer-Lite board [14].

Outline The paper is organized as follows: In Section 2, Deep Learning and Deep Learning-based Side-Channel attacks are described. In Section 3, we present our method to apply Deep Learning techniques in a Non-Profiled scenario with illustrations and examples. In Section 4, we give more detailed results from experiments performed on simulated data and traces collected from the ChipWhisperer board and from the ASCAD database. Finally, in Section 5 we conclude and summarize the interests of this new attack.

2 Deep Learning-Based Side-Channel attacks

2.1 Deep Learning

Deep Learning is a branch of Machine Learning which uses deep Neural Networks (NN) and which has been applied to many fields such as image classification, speech recognition or genomics. [15,16,17]. In this section, we give a brief description of Deep Learning for data classification. In such a case, the objective is to classify some data $x \in \mathbb{R}^D$ based on their labels $z(x) \in \mathcal{Z}$, where D is the dimension of the data to classify and $\mathcal{Z}$ is the set of classification labels. For simplicity's sake, we can consider $\mathcal{Z} = \{0, 1, \ldots, U-1\}$ with U is the number of classification labels. We also define the function $L : \mathbb{R}^D \longrightarrow \mathbb{R}^{|\mathcal{Z}|}$ as:

$$L(x)[i] = \begin{cases} 1 & \text{if } i = z(x) \\ 0 & \text{otherwise} \end{cases}$$

which can be seen as a vector representation of the label $z(x)$. A Neural Network is a function $\mathtt{N} : \mathbb{R}^D \longrightarrow \mathbb{R}^{|\mathcal{Z}|}$ which takes as input a data to classify $x \in \mathbb{R}^D$, and outputs a score vector $y = \mathtt{N}(x) \in \mathbb{R}^{|\mathcal{Z}|}$. Additionally, one can define an error function $\Delta : \mathbb{R}^D \longrightarrow \mathbb{R}$ for instance as the Euclidean distance[1] between the output of the Neural Network and the vector representation of the label:

$$\Delta(x) = \left(\sum_{i=1}^{|\mathcal{Z}|} (L(x)[i] - \mathtt{N}(x)[i])^2 \right)^{\frac{1}{2}}.$$

Deep Learning can be used to *train* neural networks to classify data based on their label. We consider we have a set of M training samples $X = (x_i)_{1 \leqslant i \leqslant M}$ whose the corresponding labels are known. To quantify the classification error of the network, one can use a so called *loss* function. The loss can be defined for instance as the average error[1] over all the training samples x_i:

$$loss(X) = \frac{1}{M} \sum_{i=1}^{M} \Delta(x_i).$$

Based on these definitions, data classification using Deep Learning is composed of two steps:

[1] The error and loss functions presented here are given only as examples. There exist actually many different error/loss functions which can be used when performing Deep Learning.

– A training/learning phase: for a chosen number of iterations called *epochs*, the DL method processes the set of training samples X and automatically tunes the network internal parameters by applying the Stochastic Gradient Descent algorithm [18] to minimize the loss function output.
– A classification phase: to classify a data x whose the corresponding label is unknown, one computes $\ell = \underset{j \in \mathcal{K}}{\text{argmax}}\ \mathtt{N}(x)[j]$. The classification is successful if $\ell = z(x)$.

Multi Layer Perceptron A Multi Layer Perceptron (MLP) is a type of Neural Network which is composed of several perceptron units [19]. As shown on Fig. 1, a perceptron $\mathtt{P} : \mathbb{R}^n \longrightarrow \mathbb{R}$ takes as input a vector $x \in \mathbb{R}^n$ and outputs a weighted sum evaluated through an *activation function* denoted A:

$$\mathtt{P}(x) = A\big(\sum_{i=1}^{n} w_i x_i\big).$$

Common activation functions are for instance the Rectified Linear function (relu), or the Hyperbolic Tangent function (tanh).

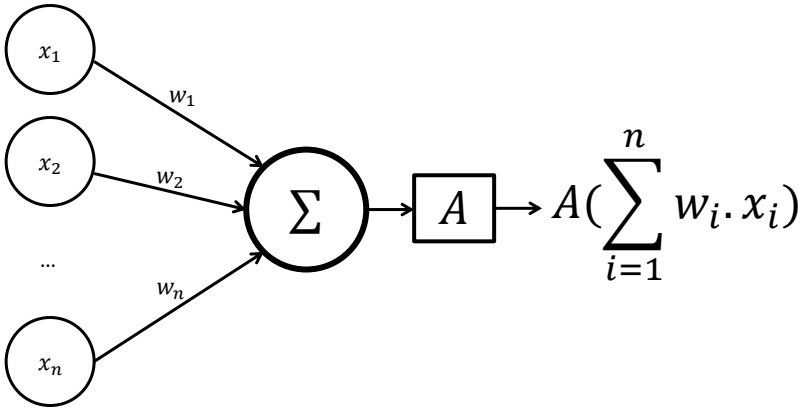

Fig. 1: Perceptron unit diagram.

A Multi Layer Perceptron is a Neural Network which is a combination of many perceptron units organized in layers as shown in Fig. 2. Each

perceptron output of one layer is connected to each perceptron of the next layer. A MLP is composed of an input layer, and output layer and a series of intermediate layers called *hidden layers*. Each layer is composed of one or several perceptron units. During the training of a MLP, the weights of the network are tuned in order to minimize the loss function.

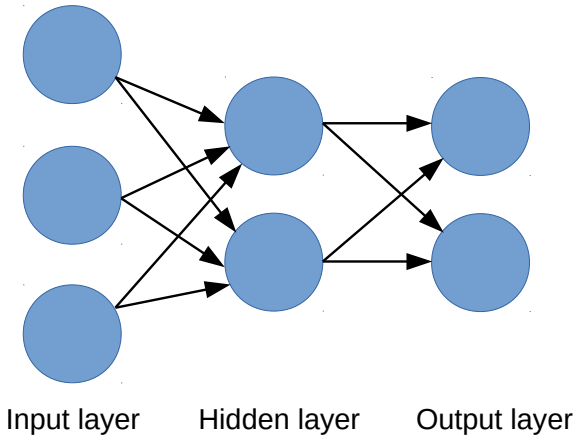

Fig. 2: Multi Layer Perceptron with 1 hidden layer.

CNN Convolutional Neural Networks is a family of deep Neural Networks which combines two types of layers called *Convolutional* layers and *Pooling* layers and has shown good results specially in the field of image recognition [20,21]. Convolutional layers apply convolution operations to the input by sliding a set of *filters* along the traces as shown on Fig. 3. The filters weights are shared across the space which allows to learn translation-invariant features. During the training of a CNN, the filters weights are tuned in order to minimize the loss function.

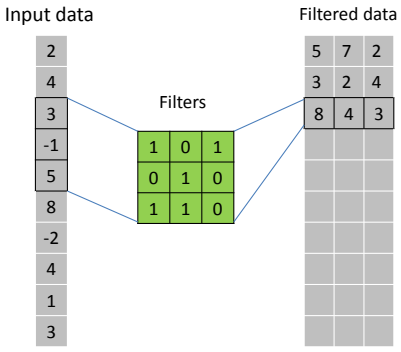

Fig. 3: Convolution operation using 3 filters of size 3.

The pooling layers are non-linear layers used to reduce the spatial size of the data. It therefore reduces the amount of computation in the network. It first partitions the input into a set of non-overlapping areas of same dimensions. For each area, the pooling operation outputs a single value which summarizes the input data in this area. The most common types of pooling operations are the *Max Pooling* operation which outputs the max value of each area and the *Average Pooling* operation which outputs the average of each area. Fig. 4 gives an example of Max Pooling operation. The input matrix is divided into non-overlapping areas of size (2×2) and the pooling operation outputs the max values of each area.

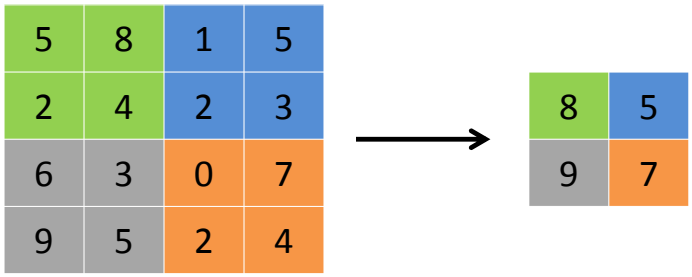

Fig. 4: Max Pooling operation example.

The CNN architecture has a natural translation-invariance property due to the use of pooling operations and shared weights applied across space during the convolution operations. Therefore, CNN is particularly interesting when dealing with de-synchronized side-channel traces as it is able to learn and detect features even if the traces are not perfectly aligned [2].

2.2 Deep Learning in practice

Many open-source Deep Learning frameworks such as Keras [22] or TensorFlow provide user friendly API to build, train and test Deep Learning architectures. For this paper we decided to use Keras and all results presented in the paper were obtained using this framework. After defining the Neural Network architecture the user only needs to provide the training data set and labels to the framework in order to perform the DL training which is managed automatically by the framework.

Loss and Accuracy During a Deep Learning training, it is possible to monitor the *loss* and *accuracy* of the training. As introduced in Section 2.1, the loss quantifies the classification error of the network. The accuracy at a given epoch can be defined for instance as the proportion of training samples that are correctly classified by the Neural Network. Both metrics give information about the evolution of the training. A decreasing loss and increasing accuracy usually indicate that the network is properly learning the data, except in case of *overfitting* where the Neural Network actually *memorizes* the data instead of learning the targeted features. In the rest of the paper, we denote as $loss \in \mathbb{R}^{n_e}$ and $acc \in \mathbb{R}^{n_e}$ the loss and accuracy over the epochs of a Deep Learning training, where n_e is the number of epochs used during the training.

Keras offers several options to compute the loss and the accuracy during a Deep Learning training [22]. For the loss function it is for instance possible to use the *mean squared error* or the *categorical crossentropy*. For the accuracy one can use for example the *categorical accuracy* or the *top-k categorical accuracy*. The choice of the loss function can have a big impact on the learning efficiency. For all results presented in Section 3 and Section 4, we used the *mean squared error* loss function and the *categorical accuracy* as we observed that this combination provided good results during our experiments.

Validation data In addition to training samples and labels, it is also possible to provide validation samples and labels to the DL training method.

The validation set is usually a separated set of data which is only used to monitor the training efficiency but is not used for the training itself. It is important that none of the validation samples are part of the training set. When validation samples are used, it is then possible to monitor the validation loss $loss_{val} \in \mathbb{R}^{n_e}$ and validation accuracy $acc_{val} \in \mathbb{R}^{n_e}$ over the epochs which are computed using the validation data. Observing the validation metrics $loss_{val}$ and acc_{val} in addition to the metrics $loss$ and acc gives a better overview of the training evolution. If for instance, acc increases but acc_{val} stays stable or decreases, it could mean that overfitting happens and that the network only memorizes the training samples and is not able to classify data outside the training set. On the other hand, if both acc and acc_{val} increase, and both $loss$ and $loss_{val}$ decrease over the epochs, it indicates that the network learns properly the targeted features and that there is no overfitting.

In the rest of the paper we denote by:

$$acc, loss = \texttt{DL}(X, Y, n_e)$$

a Deep Learning training over n_e epochs, using the set X as training samples and the set Y as training labels. The accuracy $acc \in \mathbb{R}^{n_e}$ and loss $loss \in \mathbb{R}^{n_e}$ of the training are returned as output of this process. In a similar way, we denote by:

$$acc, loss, acc_{val}, loss_{val} = \texttt{DL}(X, Y, X_{val}, Y_{val}, n_e)$$

a Deep Learning training using validation samples and labels X_{val} and Y_{val}, and returns as output the corresponding training and validation losses and accuracies.

2.3 Profiled Deep Learning Side-Channel attacks

In this section, we remind how Deep Learning can be applied to perform Profiled Side-Channel attacks [1,2,3]. We consider that the attacker has access to a pair of identical devices: a target device running a cryptographic operation with a fixed unknown key $\boldsymbol{k^*} \in \mathcal{K}$ and a profiling device with knowledge and control of the keys and inputs. We consider that a divide-and-conquer strategy is applied and that $\mathcal{K} = \{0, 1, \ldots, 255\}$. The goal of the attack is to recover the secret key byte $\boldsymbol{k^*}$. The method proposed in [1] is to perform a Profiled attack similar to a Template Attack [5], but using Deep Learning training as a profiling method instead of using Multivariate gaussian profiling as in Template Attacks.

Profiling phase For the profiling phase, a set of N traces $T_k = \{T_{i,k} \mid i = 1, \ldots, N\}$ is collected from the profiling device for each key $k \in \mathcal{K}$ leading to a set X of $(N \times 256)$ training traces:

$$X = \bigcup_{k=0}^{255} T_k \ .$$

The set of training labels Y is defined as the set of keys $z(T_{i,k}) = k$ corresponding to the training traces. To profile the leakage, a Deep Learning training $\mathtt{DL}(X, Y, n_e)$ is performed using the Side-Channel traces as training data in order to build a Neural Network $\mathtt{N}$ able to classify the side-channel traces based on their corresponding key values.

Attack phase To recover the secret key value $\boldsymbol{k^*} \in \mathcal{K}$ using M side-channel traces $(T_i)_{1 \leqslant i \leqslant M}$ collected from the target device, one first evaluates each trace T_i using the trained Neural Network to get M score vectors $y_i = \mathtt{N}(T_i) \in \mathbb{R}^{|\mathcal{K}|}$. One can then selects the key k which leads to the highest summed score:

$$k = \underset{j \in \mathcal{K}}{\operatorname{argmax}} \ \Big(\sum_{i=1}^{M} y_i\Big)[j].$$

The attack is successful if $k = \boldsymbol{k^*}$.

Interests Previous publications studied the interest of using Deep Learning to perform Profiled Side-Channel attacks. In [1], Maghrebi *et al.* showed that different type of Neural Networks such as CNN and MLP can outperform other Profiled attacks such as Template Attacks in some cases. In [2], the authors showed that the translation-invariance property of CNN networks can be used against de-synchronized traces to improve the attack results. Moreover, they showed that Data Augmentation can also be applied to artificially increase the size of the training set and improve the learning phase and attack results. However, all these studies focused only on applying Deep Learning to perform Profiled attacks.

3 Non-Profiled Deep Learning Side-Channel attacks

In this section we present our attack method to apply Deep Learning techniques in a Non-Profiled context. In the first subsection, we describe the principle of the attack. In the following subsections, we further discuss about some specific points of the attack and provide illustrations. More advanced experiments of the attack are presented in Section 4.

3.1 Deep Learning Power Analysis

For the rest of the paper, we consider a Non-Profiled Side-Channel attack scenario. In such context, an attacker collects N side-channel traces $(T_i)_{1 \leqslant i \leqslant N}$ corresponding to the manipulation of a sensitive value $F(d_i, \boldsymbol{k^*})$ where $(d_i)_{1 \leqslant i \leqslant N}$ are known random values and $\boldsymbol{k^*} \in \mathcal{K}$ is the fixed secret value. Usually such an attack is performed following a divide-an-conquer strategy, and one has for instance $|\mathcal{K}| = 256$ with d_i and $\boldsymbol{k^*}$ 8-bit values. For the rest of the paper we focus on the AES algorithm. In this case, the target function F can be chosen as the AES Sbox function, meaning that $F(d_i, \boldsymbol{k^*}) = Sbox(d_i \oplus \boldsymbol{k^*})$.

For a classic Non-Profiled attack such as CPA, for each key hypothesis $k \in \mathcal{K}$ the attacker computes a series of hypothetical intermediate values $(V_{i,k})_{1 \leqslant i \leqslant N}$ such that $V_{i,k} = F(d_i, k)$ and then applies a leakage model, for instance the Hamming Weight (HW) leakage model to get series of hypothetical power consumption values $(H_{i,k})_{1 \leqslant i \leqslant N}$ such that $H_{i,k} = HW(V_{i,k})$. The attacker then computes the correlation between the series $(H_{i,k})_{1 \leqslant i \leqslant N}$ and the collected side-channel traces. For the correct key value $\boldsymbol{k^*}$, the series of intermediate values will be correctly guessed, and it will lead to a high correlation if the leakage model is well-chosen. For all the other key candidates, the guessed intermediate values will be wrong and therefore not correlated to the traces, leading to low correlation values. The attacker is able to discriminate the correct key value by selecting the key leading to the highest correlation.

To apply Deep Learning in a Non-Profiled context, our idea is to combine CPA-like hypotheses with Deep Learning trainings. For each key hypothesis $k \in \mathcal{K}$ the attacker computes the series $(V_{i,k})_{1 \leqslant i \leqslant N}$ and $(H_{i,k})_{1 \leqslant i \leqslant N}$. He then performs a Deep Learning training using the traces $(T_i)_{1 \leqslant i \leqslant N}$ as training data, and the series $(H_{i,k})_{1 \leqslant i \leqslant N}$ as the corresponding labels. As for a CPA attack, for the correct key value $\boldsymbol{k^*}$, the series of intermediate values will be correctly guessed, and therefore the labels used for the Deep Learning training will be correct. If the DL architecture is able to learn the targeted features, this should lead to a successful training and to good training metrics such as a decreasing loss and increasing accuracy over the epochs when the correct key $\boldsymbol{k^*}$ is used to compute the labels. On the other hand, for all the other key candidates, the series of intermediate values will be incorrect, and this should lead to unsuccessful trainings. The attacker can then discriminate the correct key value from the other candidates by selecting the key leading to the best Deep Learning training metrics. A description of different Deep Learning

metrics which can be used is given in Section 3.2. We will use the acronym DLPA (Deep Learning Power Analysis) for this new attack method. Algorithm 1 summarizes the DLPA procedure to perform a Non-Profiled attack using Deep Learning:

Algorithm 1 DLPA

Inputs: N traces $(T_i)_{1 \leqslant i \leqslant N}$ and corresponding plaintexts $(d_i)_{1 \leqslant i \leqslant N}$. Number of epochs n_e.

1: Set training data as $X = (T_i)_{1 \leqslant i \leqslant N}$
2: **for** $k \in \mathcal{K}$ **do**
3: Compute the series of hypothetical values $(H_{i,k})_{1 \leqslant i \leqslant N}$
4: Set training labels as $Y = (H_{i,k})_{1 \leqslant i \leqslant N}$
5: Perform DL training: $acc, loss = \texttt{DL}(X, Y, n_e)$
6: **end for**
7: **return** key k which leads to the best DL training metrics

It is important to note that the DLPA attack method is not limited to a specific type of Neural Network. In the next section, we study some metrics such as the loss and accuracy which can be used to perform DLPA with any type of Neural Networks. This provides many possibilities when performing the attacks as the attacker can adapt the architecture based on the targeted implementation and device. In this paper, we focus on two variants of DLPA, using MLP and CNN architectures as underlying neural networks. To distinguish both variants, MLP-DLPA will refer to a DLPA process using MLP as Neural Network architecture and CNN-DLPA will refer to a DLPA attack using CNN.

3.2 Metrics

In this section we study different metrics that can be used to reveal the correct key value during a DLPA attack. To illustrate how the different metrics can lead to the recovery of the secret key, we present some results obtained from a simulation data set. We generated $N = 5,000$ simulated traces as follows:

- 50 samples per trace.
- Sbox leakage set at time sample $t = 25$ and defined as $Sbox(d_i \oplus \boldsymbol{k^*}) + \mathcal{N}(0, 1)$ with d_i a known randomized byte and $\boldsymbol{k^*}$ a fixed key byte. $\mathcal{N}(0, 1)$ corresponds to a Gaussian noise of mean $\mu = 0$ and standard deviation $\sigma = 1$.
- All other points on the traces are chosen as random values in $[0; 255]$.

The purpose of this simulation is only to illustrate how some Deep Learning metrics can be used to discriminate the correct key from the other candidates. More detailed simulations and results are presented in Section 4. Using this simulation data, we performed the attack as defined in Algorithm 1 and observed the following metrics.

Layers weights The first hidden layer of a Multi Layer Perceptron takes as input the n samples of a side-channel trace. By definition of the MLP, each time sample t of the traces is paired with R weights $(W_{t,j})_{1 \leqslant j \leqslant R}$ where R is the number of neurons in the first hidden layer. Therefore, the first hidden layer weights can be seen as a $(n \times R)$ matrix W where $W_{i,j}$ is the weight of the connection between the i^{th} sample of the trace and the j^{th} neuron of the first hidden layer. Our observations revealed that the Deep Learning process will tend to put higher weights on the time samples corresponding of the side-channel leakage than on the other samples when the correct key is used to compute the labels. To discriminate the correct key guess it is then interesting to observe the following metrics for each time sample t after the Deep Learning training :

$$S[t] = \sum_{j=1}^{R} (W_{t,j})^2 \quad \textbf{(sum of squared weights)} \ ,$$

$$P[t] = \prod_{j=1}^{R} |W_{t,j}| \quad \textbf{(product of weights)}.$$

We performed a MLP-DLPA attack as described in Algorithm 1 using our simulation traces with $n_e = 50$ epochs per guess. The two weights metrics obtained for all the 256 key guesses can be observed in Fig. 5.

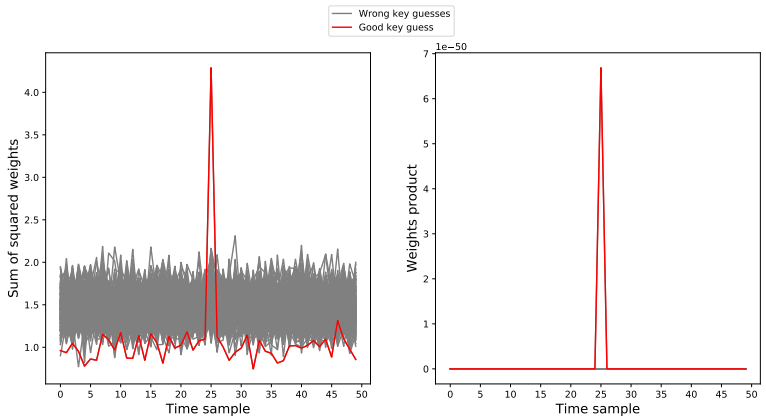

Fig. 5: MLP-DLPA attack using weights metrics. Left: sum of squared weights. Right: product of weights.

We can observe that the correct key guess clearly leads to higher values for the two metrics, at the time sample $t = 25$ which corresponds to the location of the Sbox leakage. This means that both metrics can be used to reveal the correct key candidate by selecting the key guess leading to the highest values. Additionally, as the highest values are located precisely at $t = 25$, this method is also able to reveal the leakage location. Therefore, DLPA using weights metrics can also be used to reveal points of interest (we further discuss this application in Section 4.1 when targeting masked implementations).

However, such metrics are limited to only specific Neural Network architectures such as MLP. The following section introduces generic metrics that can be used for any type of Neural Network.

Loss and accuracy As presented in Section 2.2, the two main metrics that can be observed to monitor a Deep Learning training are the loss and the accuracy of the training over the epochs. In this section we show that these two metrics can also be used to discriminate the correct key value when performing a DLPA attack. As illustration, we present in Fig. 6 the losses and accuracies obtained when performing a MLP-DLPA attack with our simulation data set with $n_e = 50$ epochs per guess. The figure clearly shows that the training using the correct key value leads to a higher accuracy and lower loss compared to the trainings for the other candidates. These metrics can therefore be used to reveal the correct key

by selecting the guess leading to the highest accuracy or lowest loss values. Compared to the weights metrics introduced in the previous section, the loss and accuracy are generic metrics that can be used with any type of Neural Networks.

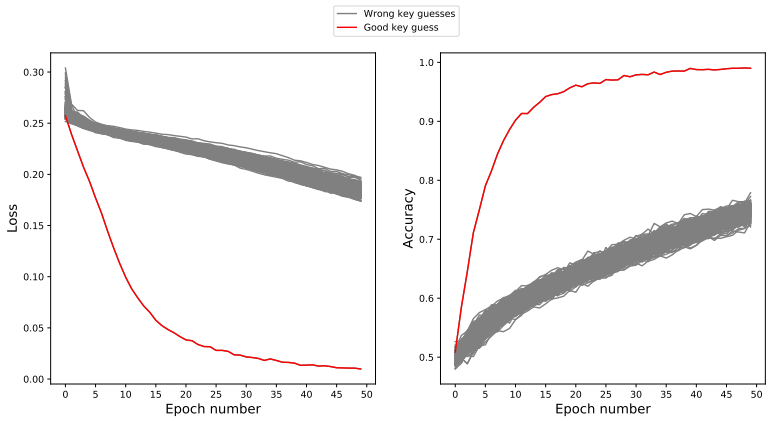

Fig. 6: Loss (left) and accuracy (right) over the training epochs for all the key guesses when applying DLPA.

Using validation data We observed that using directly loss and accuracy metrics from the training traces can sometimes lead to poor results. Indeed, in some cases, the neural network is able to lower the loss function and increase the accuracy even for the wrong key candidates. In such cases, it is then interesting to use a set of validation traces as explained in Section 2.2 and to observe the loss or accuracy obtained from this validation set during the training. The set of N attack traces can be split into respectively one set of N_T training traces, and one set of N_V validation traces with $N_T + N_V = N$. Algorithm 2 summarizes this variant of DLPA using validation data which can lead to better results in some cases.

Algorithm 2 DLPA with validation data

Inputs: N traces $(T_i)_{1 \leqslant i \leqslant N}$ and corresponding plaintexts $(d_i)_{1 \leqslant i \leqslant N}$. Number of epochs n_e.

Choose N_T and N_V integers such that $N = N_T + N_V$ with $N_T > N_V$
Set training data as $X = (T_i)_{1 \leqslant i \leqslant N_T}$
Set validation data as $X_{val} = (T_i)_{N_T+1 \leqslant i \leqslant N}$
for $k \in \mathcal{K}$ **do**
    Compute the series of hypothetical values $(H_{i,k})_{1 \leqslant i \leqslant N}$
    Set training labels as $Y = (H_{i,k})_{1 \leqslant i \leqslant N_T}$
    Set validation labels as $Y_{val} = (H_{i,k})_{N_T+1 \leqslant i \leqslant N}$
    Perform DL training: $acc, loss, acc_{val}, loss_{val} = \texttt{DL}(X, Y, X_{val}, Y_{val}, n_e)$
end for
return key k which leads to the best DL metrics

There is no rule concerning the choice of N_T and N_V but we usually wants that $N_T > N_V$. It is also important to choose a good trade off between training and validation data. If N_V is too high, not enough traces are available for the training. If N_V is too low, then the attack may fails as not enough validation traces are available to reveal the good key value.

To simulate conditions where validation data is useful, we generated a set of simulation traces similar as previously, but this time composed of $N = 8,000$ traces of $n = 1,000$ points and we added a slight artificial de-synchronization to the traces. We then applied a MLP-DLPA attack with and without validation data with $n_e = 150$ epochs per guess and compared the results. When using validation data, we split the $8,000$ attack traces into $N_T = 6,500$ training traces and $N_V = 1,500$ validation traces. Results are presented in Fig. 7.

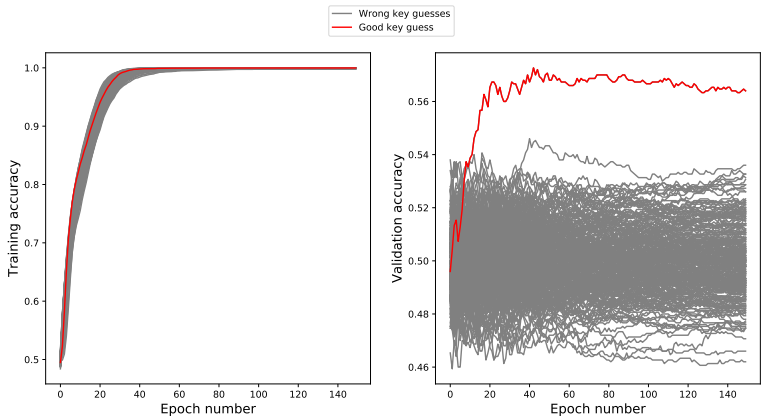

Fig. 7: Comparison of DLPA with and without validation data. Left: Training accuracies with no validation data. Right: Validation accuracies.

We can observe that when no validation data is used, all accuracies tend towards 1 and that it is not possible to distinguish the correct key value. On the other hand, when validation data is used, the validation accuracy reveals the correct key value after a few epochs. It confirms that in some cases it can be useful to use some of the attack traces as validation data and study the corresponding validation metrics to reveal the key.

Summary In this section we presented different metrics which can be used to reveal the correct key value when performing DLPA. Some metrics, such as the first layer weights of MLP networks are specific to some architectures and some metrics like the loss and accuracy are generic metrics which can be used with any type of networks. We showed with our examples that all these metrics can be used to reveal the correct key. For the rest of the paper, we decided to use the accuracy as the main metric to determine if the attack is successful. It means that we will consider that a DLPA attack is successful when the training with the good key guess leads to the highest accuracy.

3.3 Labels

For a Profiled Deep Learning attack, it is possible to use directly the Sbox output values $Sbox(d \oplus k)$ as labels for the Deep Learning training without applying any additional function to the Sbox output. However, for a Non-Profiled Deep Learning attack, it is not possible to use directly the values

$Sbox(d_i \oplus k)$ as labels for the attack. Indeed, if one uses the identity labeling $H_{i,k} = Sbox(d_i \oplus k)$, the partition of the attack traces derived from this labeling method will be equivalent for all the key guesses. In other words, the partition of the traces

$$P_k = \{E_u^{(k)} \mid u \in \{0, \ldots, 255\}\}$$

defined by the sets

$$E_u^{(k)} = \{T_i \in (T_i)_{1 \leqslant i \leqslant N} \mid Sbox(d_i \oplus k) = u\}$$

is the same for all key guesses k. This means that from one key guess to another, there is no difference in the partition of the attack traces, and that only the labels are permuted which does not impact the training metrics. It means that using the identity labeling $H_{i,k} = Sbox(d_i \oplus k)$ will naturally lead to similar Deep Learning metrics for all the key candidates, making it impossible to discriminate the correct key value. That is why it is necessary to apply a non-injective function to the Sbox output to compute the labels so that the partition of the attack traces is different from one guess to another. We propose hereafter two methods:

Hamming Weight labeling One solution is to use labels based on the Hamming Weight of the guessed value:

$$H_{i,k} = HW(V_{i,k}).$$

Binary labeling During our experiments, we noticed that using only two labels derived from the guessed values is also a good alternative. We propose two binary labeling methods to perform DLPA based on the Least Significant bit (LSB) and Most Significant bit (MSB) of the guessed value:

$$H_{i,k} = \begin{cases} 0 & \text{if } V_{i,k} < 127 \\ 1 & \text{otherwise} \end{cases} \qquad \textbf{(MSB labeling)} \ ,$$

$$H_{i,k} = V_{i,k} \ mod \ 2 \quad \textbf{(LSB labeling)}.$$

To illustrate the importance of the labeling method, we performed the same attack as in Section 3.2 but using the identity labeling ($H_{i,k} = Sbox(d_i \oplus k)$). The comparison of accuracies obtained when using the identity labeling and binary labeling is presented in Fig. 8.

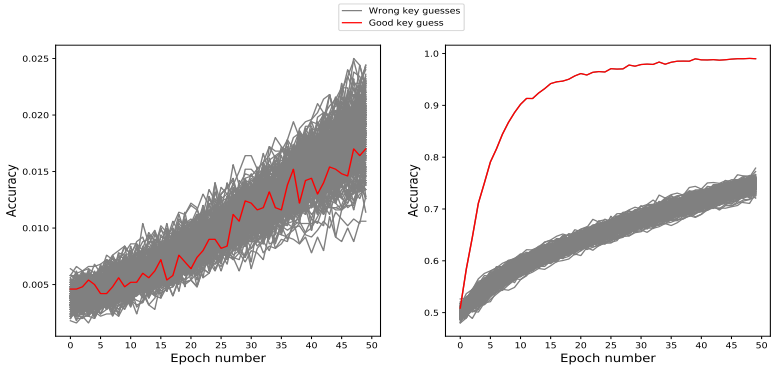

Fig. 8: MLP-DLPA using two different labeling methods. Left: Identity labeling . Right: Binary labeling (MSB).

As expected, the left graph shows that all key guesses lead to similar accuracies when using the identity labeling. All accuracies are not perfectly identical even when using the identity labeling as the Deep Learning training is not a deterministic process. Indeed, the training always depends on the weights initialization as well as the shuffling of the input data during the different epochs, which explains the slight differences between the accuracies even though the identity labeling is used. However, using the identity labeling will always lead to similar accuracies making it impossible to distinguish the correct key value. That is why it is necessary to use other labeling methods, such as the Hamming Weight or Binary labeling methods. During our experiments, the binary labeling usually provided better results than using Hamming Weight labels. For the rest of the paper, all the results presented were obtained using the binary labeling.

3.4 High-Order attacks

High-Order CPA attacks A common countermeasure to protect cryptographic implementations against Profiled and Non-Profiled attacks is to conceal the sensitive intermediate values with *masks*. In the following, we focus on Boolean masking, which is commonly used to protect symmetric algorithms like AES [23]. In the case of Boolean masking, a sensitive intermediate value, for instance the AES Sbox output, is never manipulated in plain, but instead, is represented as a XOR of $s + 1$ *shares* as follows:

$$S = Sbox(d \oplus k) \oplus m_1 \oplus \cdots \oplus m_s$$

The values $m_1, \ldots, m_s$ are called the *masks* and S is called the *masked value*. Each mask m_i is generated as a random value for each execution of the algorithm, making the leakages uncorrelated to the sensitive values. However, *High-Order* attacks such as High-Order CPA have been developed to target such implementations [24,25,26,27]. A High-Order attack is usually composed of two steps:

- A pre-processing phase: the leakages of the masks are combined with the leakage of the masked value using combination functions such as the absolute difference or centered product [27].
- The attack phase: a statistical distinguisher, for instance the Pearson's Correlation is used to extract information from the combined leakages traces.

In the rest of the paper, we focus on two practical cases, where an Sbox operation is protected respectively by 1 and 2 random masks as follows:

$$S = Sbox(d \oplus k) \oplus m_1 \ ,$$

$$S = Sbox(d \oplus k) \oplus m_1 \oplus m_2.$$

A high-order attack targeting a 1-mask protected implementation is called a *second order* attack, and a *third order* attack corresponds to a high-order attack targeting an implementation protected with 2 masks.

Leakages combination For a second order attack, one needs to combine the leakage of the mask m_1 with the leakage of the masked Sbox value $Sbox(d \oplus k) \oplus m_1$. If the locations of the mask and masked value are known, then one only needs to combine these two leakage locations together. If the locations of the mask and masked value leakages are unknown, a solution is to combine all the possible couples of points in the trace together. If the traces are of size n, such processing will lead to combined traces of size $\frac{n \times (n-1)}{2}$. Therefore, for large traces, such processing can become too complex and not practical.

High-Order DLPA In [1] and [3], the authors successfully attacked first order protected AES implementations, showing that it is possible to break 1-mask protected implementations using CNN and MLP networks in a Profiled attack context. Our experiments presented in Section

4 show that it is possible to break implementations protected with 1 and 2 masks using Deep Learning also in a Non-Profiled context with a reasonable number of traces. In comparison with High-Order CPA, DLPA does not requires to combine the leakages prior to the attack. Moreover, we show in Section 4.1 that MLP-DLPA can also be used to highlight areas of interest, for instance masks leakages locations, by studying the MLP weights as introduced in Section 3.2.

3.5 CNN and Data Augmentation

In [2], Cagli *et al.* highlighted that due to its translation-invariance property, the CNN architecture is naturally able to extract information even from de-synchronized traces. Additionally, they showed that Data Augmentation can also be performed to artificially increase the training set size which can lead to more efficient trainings and better results during the attack.

The results presented in Section 4 demonstrate that these properties are also applicable when performing DLPA attacks in a Non-Profiled context: first, our experiments show that CNN-DLPA leads to good results against de-synchronized traces due to the translation-invariance property of the CNN architecture. Additionally, the results indicate that Data Augmentation can also significantly improve the results of MLP-DLPA and CNN-DLPA attacks. Using these properties, we show in Section 4 that DLPA can outperform CPA when attacking de-synchronized traces.

4 Experiments

In this section we further study the efficiency and interests of the DLPA attack method. We focus on the application of MLP-DLPA against masked implementations and the application of DLPA with Data Augmentation against de-synchronized traces. Additionally we present some comparisons of DLPA with CPA. We perform these experiments using 3 types of side-channel traces:

- Simulated traces.
- Traces collected from the ChipWhisperer-Lite board [14].
- Traces from the public side-channel database ASCAD [3].

4.1 Simulations

High-Order DLPA The simulation results presented in Section 3 already demonstrate the efficiency of DLPA against unprotected (non-masked) implementations. In this section we focus on attacking simulated

masked implementations. We study the efficiency of MLP-DLPA when targeting an AES Sbox output protected with 1 and 2 random masks. A similar procedure as in Section 3.2 was used to generate simulated traces. The only difference is that for this experiment, random masks values are added to the simulation traces. To simulate a 1-mask protected Sbox, we generated traces as follows:

- 50 samples per trace.
- Masked Sbox leakage set at $t = 25$ and defined as $Sbox(d_i \oplus \boldsymbol{k^*}) \oplus m_1 + \mathcal{N}(0, 1)$ with d_i and m_1 randomized bytes and $\boldsymbol{k^*}$ a fixed key byte.
- Mask leakage set at $t = 15$ and defined as $m_1 + \mathcal{N}(0, 1)$.
- All other points on the traces are chosen as random values in $[0; 255]$.

We applied MLP-DLPA as in Algorithm 1 with $n_e = 50$ epochs per guess on $N = 5,000$ simulated traces. In Fig. 9 we present the results of this attack:

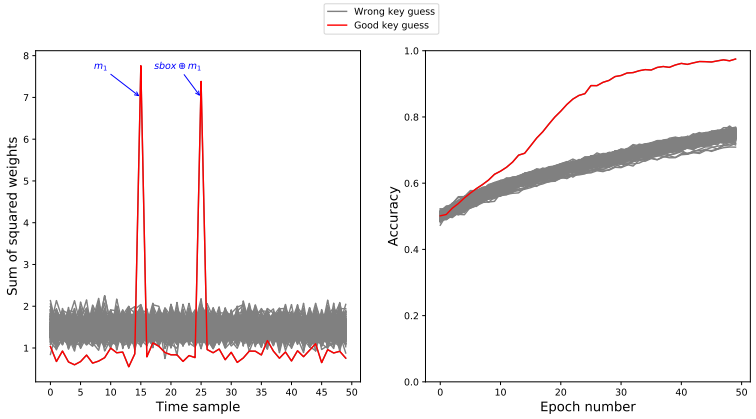

Fig. 9: MLP-DLPA applied to 1-mask protected Sbox. Left: Sum of squared weights. Right: accuracy over the epochs.

The attack is successful as the training with the good key value leads to the highest accuracy after less than 10 epochs per guess. Moreover, we can observe that the sum of squared weights metric reveals two distinct peaks, which correspond to the precise leakage locations of the mask and protected Sbox leakages.

We performed similar experiment on a 2-masks protected Sbox which we simulated as follows:

- 50 samples per trace
- Masked Sbox leakage set at $t = 25$ and defined as $Sbox(d_i \oplus \boldsymbol{k^*}) \oplus m_1 \oplus m_2 + \mathcal{N}(0, 1)$ with d_i, m_1 and m_2 randomized bytes and $\boldsymbol{k^*}$ a fixed key byte.
- First mask leakage set at $t = 15$ and defined as $m_1 + \mathcal{N}(0, 1)$.
- Second mask leakage set at $t = 5$ and defined as $m_2 + \mathcal{N}(0, 1)$.
- All other points on the traces are chosen as random values in $[0; 255]$.

We applied MLP-DLPA with $n_e = 50$ epochs per guess on $N = 10,000$ simulated traces. The results of the attack are presented in Fig. 10.

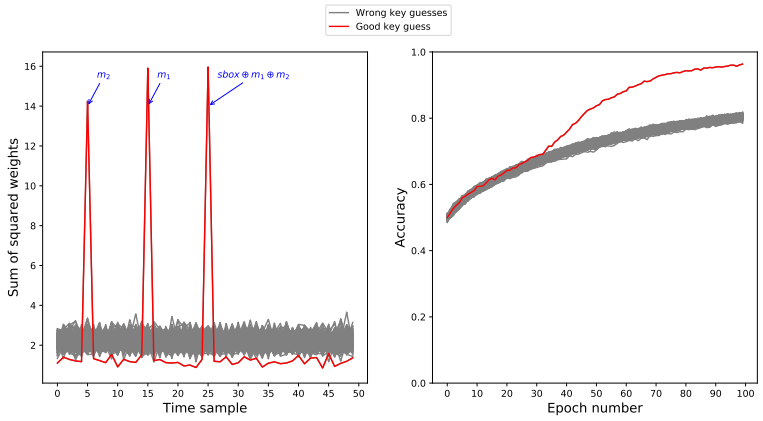

Fig. 10: MLP-DLPA applied to 2-masks protected Sbox. Left: Sum of squared weights. Right: accuracy over the epochs.

The attack is successful after around 40 epochs per guess. The first layer weights also reveal the location of the masks and protected Sbox leakages. As a conclusion, these two experiments demonstrate that it is possible to target masked implementations with DLPA. In comparison with high-order CPA attacks, this method does not require to combine the leakage points together. Additionally, it shows that applying MLP-DLPA can also reveal the locations of the masks and masked values leakages. This means that DLPA could also be applied as a reverse engineering method to reveal points of interests and to reveal information about the

implementation. DLPA could therefore be combined with other High-Order attacks: for instance the leakages locations revealed by the DLPA could be combined and attacked by a High-Order CPA attack.

CNN-DLPA and Data Augmentation In this section we study the efficiency of CNN-DLPA with and without Data Augmentation when applied on de-synchronized traces. We simulated unprotected Sbox traces similar as in Section 3.2, with only two differences:

- We introduced an artificial jitter in order to simulate de-synchronization. Each trace was shifted left or right by a random value chosen in the interval $[-6; 6]$.
- The noise level was increased to $\mathcal{N}(0, 20)$ instead of $\mathcal{N}(0, 1)$ previously.

We applied 3 attacks on $N = 2,000$ de-synchronized traces:

- A CPA attack, in order to evaluate the efficiency of a classic Non-Profiled attack against these de-synchronized traces. This will be our reference.
- A CNN-DLPA attack without Data Augmentation.
- A CNN-DLPA attack with Data Augmentation.

For the third attack we performed Data Augmentation using a method similar to the *Shifting Deformation* presented in [2]. We decided to fix the Data Augmentation factor to 10, meaning that after DA, the training set to perform the attack is composed of $N' = 2,000 \times 10 = 20,000$ traces. We performed each attack 50 times, each time with a new set of simulated traces, and studied the success rate of these attacks. The CNN-DLPA attacks were performed as in Algorithm 1, with $n_e = 50$ epochs per guess. For the CPA, the guess leading to the highest correlation was selected as the best candidate to compute the success rate over the iterations. For the CNN-MLP attacks, we selected the best candidate based on the highest accuracy and the lowest loss to compute the corresponding success rates. The results of the attacks are presented in Fig. 11.

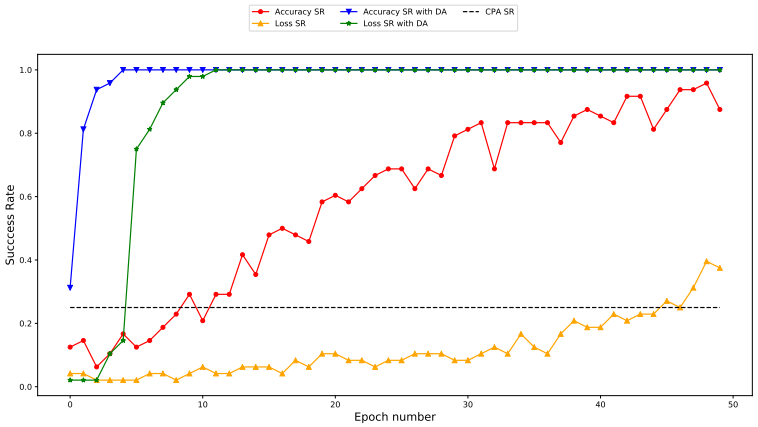

Fig. 11: Comparison of success rates between CNN-DLPA (with and without DA) and CPA against de-synchronized traces.

We observe that due to the de-synchronization of the Sbox leakage, the CPA only leads to a success rate of around 0.25. Without Data Augmentation, we can observe that CNN-DLPA leads to better results, specially when the accuracy metric is used. This is due to the translation-invariance property of the CNN architecture which compensates the effect of the traces de-synchronization. Moreover, when Data Augmentation is applied, the results are clearly improved, leading to success rates of 1 after only a few epochs per guess.

These results show that DLPA can outperform other Non-Profiled attacks as CPA in some cases, for example when applied on de-synchronized traces. In this case, it shows that the natural translation-invariance of the CNN architecture can be used in order to mount CNN-DLPA attacks that can provide successful results against de-synchronized traces. It also demonstrates that as for Profiled Deep Learning attacks [2], Data Augmentation can also be performed in Non-Profiled scenarios in order to improve attacks efficiency. Therefore, CNN-DLPA combined with Data Augmentation can be considered as an interesting alternative when performing Non-Profiled attacks on traces which cannot be perfectly re-synchronized.

4.2 ChipWhisperer

We used the ChipWhisperer-Lite board [14] in order to validate the efficiency of the DLPA method when applied on non-simulated traces with different levels of protections. First, we targeted an unprotected Sbox operation with and without de-synchronization. We studied the efficiency of an MLP-DLPA attack in this context, with and without Data Augmentation in comparison with a classic CPA attack. Then, we targeted two masked Sbox implementations protected respectively with 1 and 2 masks.

Implementations We implemented an AES Sbox operation $Sbox(d_i \oplus \boldsymbol{k^*})$ in C. For the masked implementations, we used a masked re-computed Sbox as described in [23], meaning that the Sbox value $Sbox(d_i \oplus \boldsymbol{k^*})$ is never manipulated in plain. For the 1-mask protected implementation, we collected traces containing the execution of the following operations:

- Copy of the mask m_1 in memory.
- Copy of $s = Sbox(d \oplus \boldsymbol{k^*}) \oplus m_1$ in memory.

Similarly, for the 2-masks protected implementation, we collected traces containing the execution of the following operations:

- Copy of the mask m_1 in memory.
- Copy of the mask m_2 in memory.
- Copy of $s = Sbox(d \oplus \boldsymbol{k^*}) \oplus m_1 \oplus m_2$ in memory.

For the tests against the unprotected implementation, we simply collected traces of the copy of $s = Sbox(d \oplus \boldsymbol{k^*})$ in memory. For the three implementations, we fixed a window of 500 points containing only the targeted operations. For the 1 and 2 masks protected implementation, a first order CPA was applied in order to validate that the implementations does not have first order leakages. Moreover, for the 2-masks protected implementation, we also performed a second order CPA attack to verify that the implementation does not have second order leakages.

Unprotected implementation without de-synchronization We collected $N = 3,000$ traces of the unprotected implementation from the ChipWhisperer-Lite. We then performed a first order CPA and a MLP-DLPA attack. The results are presented in Fig. 12.

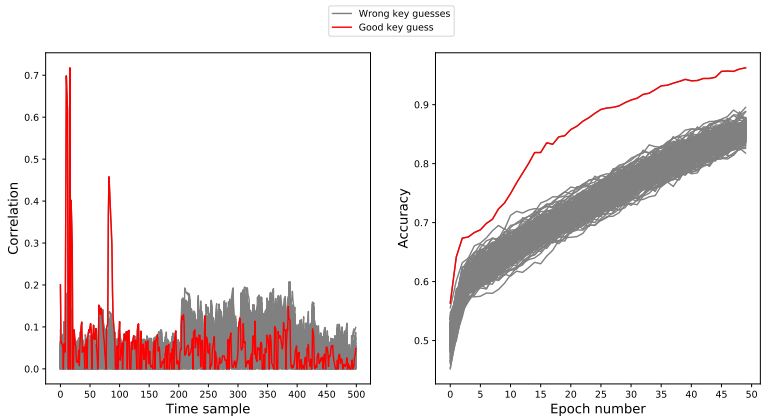

Fig. 12: Attack on CW unprotected implementation without de-synchronization. Left: First-Order CPA. Right: MLP-DLPA.

As expected, the CPA attack is successful as the targeted implementation is unprotected. We can observe that the DLPA attack is also successful with only $3,000$ traces which shows that the attack is feasible with a very reasonable number of traces even on non-simulated traces.

Unprotected implementation with de-synchronization As the ChipWhisperer traces are by default almost perfectly synchronized, it is necessary to add artificial de-synchronization in order to study the efficiency of DLPA against de-synchronized traces. We decided to add artificial de-synchronization after the collection of the traces by shifting each trace left or right by a random number of points chosen in the interval $[-15; 15]$. We applied this software de-synchronization to the $3,000$ traces from the previous experiment. We then performed three attacks against these $N = 3,000$ de-synchronized traces:

- A first order CPA attack
- A MLP-DLPA attack without Data Augmentation
- A MLP-DLPA attack with Data Augmentation

We used the same process as in section 4.1 to perform the Data Augmentation. This time, the Data Augmentation factor was set to 20, meaning that the Data Augmentation process led to $N' = 3,000 \times 20 = 60,000$ data augmented traces. The two DLPA attacks were performed using

$n_e = 50$ epochs per guess. The results of the 3 attacks are presented in Fig. 13.

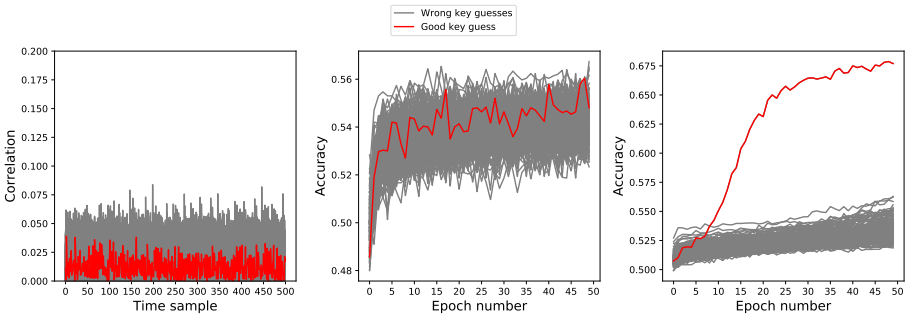

Fig. 13: Attack on CW unprotected implementation with de-synchronization. Left: CPA. Center: MLP-DLPA without DA. Right: MLP-DLPA with DA.

We can observe that even though the implementation is not protected, the CPA is not successful due to the de-synchronization of the traces. Similarly, the MLP-DLPA attack without Data Augmentation fails to reveal the correct key value. However, we can observe that when using Data Augmentation, the MLP-DLPA is successful and reveals the correct key after a few epochs per guess. It further confirms our results from simulation and shows that Data Augmentation can significantly improve the results of DLPA attacks in a Non-Profiled context using a very limited number of collected traces. Additionally, it also demonstrates that using Data Augmentation should not be limited to the CNN architecture, but can also be used successfully with MLP networks.

Masked implementations We then applied MLP-DLPA attacks against the 1 and 2 masks protected implementations. For the 1-mask protected implementation, we performed the attack on $N = 5,000$ traces collected from the ChipWhisperer with $n_e = 150$ epochs per guess. As mentioned previously, we also performed first order CPA to validate that the implementation does not have first-order leakages. The results are presented in Fig. 14.

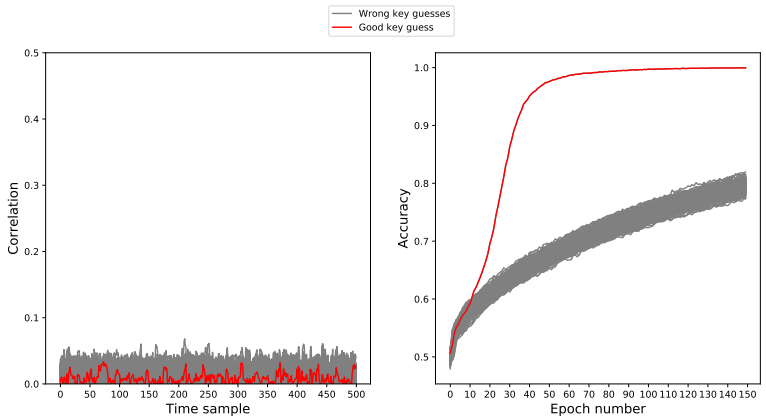

Fig. 14: Attack on CW 1-mask protected implementation. Left: First-Order CPA. Right: MLP-DLPA.

As expected, the first order CPA result confirms that the implementation does not have first order leakages. We can observe that the MLP-DLPA attack is successful and reveals the correct key value after only a few epoch per guess. It demonstrates that DLPA can break masked implementations using very reasonable number of traces collected from a device.

We then targeted the 2-masks protected Sbox implementation. We performed a first order and a second order CPA attack on the traces to validate that the implementation does not have first or second order leakages. The first order attack led to results similar to the left graph of Fig. 14 confirming that the implementation does not have first order leakages. To reduce the complexity of the second order attack and of the DLPA, we limited the number of points to 150 points per trace containing the leakages of the two masks as well as the leakage of the masked Sbox. We then combined all the possible couple points from this window using the absolute difference combination function leading to combined traces of size $\frac{150\times149}{2} = 11,175$ samples. We then performed a second order attack on these combined traces. The MLP-DLPA attack was performed without pre-processing, directly on the raw traces of 150 points with $n_e = 100$ epochs per guess. Both attacks were performed using $50,000$ traces collected from the ChipWhisperer-Lite. The results of the second order attack and of the MLP-DLPA attack are presented in Fig. 15.

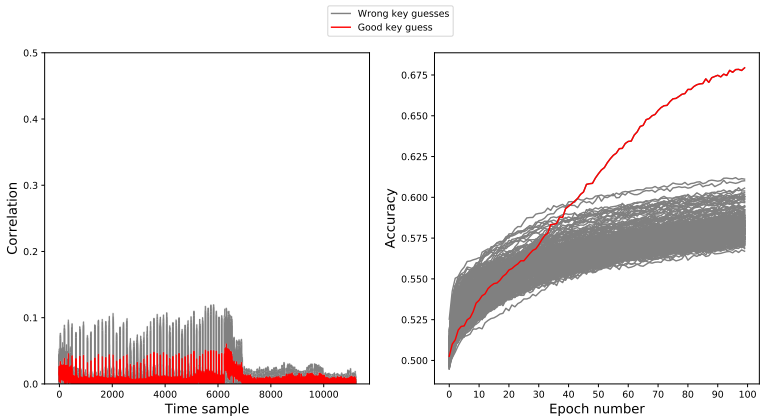

Fig. 15: Attack on CW 2-masks protected implementation. Left: Second-Order CPA. Right: MLP-DLPA.

It can be observed that the MLP-DLPA attack reveals the correct key value after around 50 epochs per guess. On the other hand, the second order CPA is not successful. The result of the second order CPA applied on $50,000$ traces from the ChipWhisperer gives a high level of confidence that the implementation does not have second order leakages. This shows that the MLP-DLPA method is able to combine the leakages of 3 different shares to reveal the secret key, even on non-simulated data, with a reasonable number of traces and without traces pre-processing.

4.3 ASCAD

ASCAD is a public database recently introduced by Prouff *et al.* in [3] to provide a common set of side-channel traces for research on Deep Learning-based Side-Channel attacks. The targeted implementation is a first order protected Software AES implementation running on an 8-bit ATMega8515 board. The main database `ASCAD.h5` is composed of two set of traces: a profiling set of $50,000$ traces to train Deep Learning architectures and an attack set of $10,000$ traces to test the efficiency of the trained Neural Networks. Each trace of the database is composed of 700 samples focusing on the processing of the third byte of the masked state $Sbox(p[3] \oplus k[3]) \oplus r[3]$ where p, k and r are respectively the plaintext, the key and the mask values. In [3] Prouff *et al.* focused on providing reference results for Profiled Deep Learning attacks using the profiling

and attack sets of the ASCAD database. In this section, we validate our Non-Profiled Deep Learning attack using this public database.

For both the profiling set and the attack set of `ASCAD.h5`, the same 16-byte fixed key is used while the plaintexts and masks are randomized. Therefore, as the key is always fixed, both the attack set and profiling set can be considered as traces obtained from a closed device to perform a Non-Profiled attack. We decided to use the profiling set to perform our experiment as it contains more traces than the attack set. We applied a MLP-DLPA attack on the first $20,000$ traces of the profiling set of `ASCAD.h5` with $n_e = 50$ epochs per guess. The result of this attack is presented in Fig. 16.

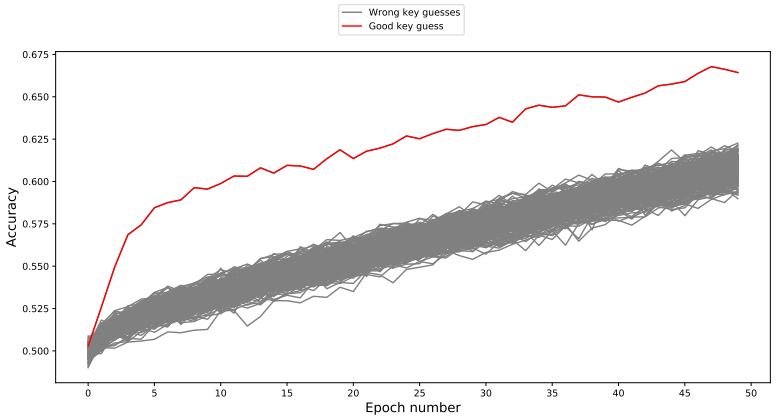

Fig. 16: MLP-DLPA attack on ASCAD.

The results show a clear success of the attack with $20,000$ traces from the `ASCAD.h5` database. It further confirms the validity of the method as well as its interest. The execution times of this attack and of previous attacks are given in the next section.

4.4 Complexity

One drawback of DLPA is that it is necessary to perform a Deep Learning training for each key guess. When using 8-bit key guesses, it means that 256 trainings are necessary. This suggests that most of the time

DLPA will be slower than first order CPA, due to the multiple trainings needed for DLPA. However, for high order attacks, we showed that DLPA does not require to combine leakage points together before the attack. As an example, to perform a second order attack on the ASCAD traces, as the location of the leakages are unknown, one would need to perform a leakage combination on 700 points for each trace. This means that a total $\frac{700\times699}{2} = 244,650$ points per traces must be processed during pre-processing and during the attack phase which would lead to a high complexity. On the other hand, DLPA does not requires any pre-processing and it able to reveal the secret key after only a few epochs. If we limit our attack on ASCAD to only 5 epochs per guess, the attack only requires around 10 minutes on our setup and is still successful. For comparison, we performed a second order CPA on the ASCAD traces by combining all the $244,650$ couples of point for each trace. The execution times of this CPA attack and of different DLPA attacks are summarized in Table. 1. We recorded these values when running the attacks in Python on our personal computer with 32 GB of RAM, a GeForce GTX 1080 GPU and a Intel Xeon E5-2687W CPU.

Implementation	Attack	Nb traces	Nb samples	Nb epochs	Time (hours)
CW 1-mask	MLP-DLPA	5,000	500	150	0.92
CW 2-masks	MLP-DLPA	50,000	150	100	5.14
ASCAD	MLP-DLPA	20,000	700	50	1.19
ASCAD	MLP-DLPA	20,000	700	5	0.17
ASCAD	2nd order CPA	1,000	700	N/A	2.16

Table 1: Execution times comparison.

The table shows that DLPA attacks can be performed in reasonable time and are therefore practical. All experiments were performed using many epochs per guess, but it can be observed that most of the time, only a few epochs were needed to reveal the correct key value. It means that these attacks can be performed faster than the results of the table by reducing the number of epochs per guess. For the results on ASCAD, the comparison with the second order CPA shows that the DLPA attack is faster than the second order CPA as no leakages combination is needed. The second order CPA applied on only $1,000$ traces takes more than 2 hours due to the high number of points during and after pre-processing, while only 10 minutes are needed to reveal the correct key value with

DLPA. This further confirms that DLPA can be an interesting alternative for instance when performing Non-Profiled High-Order attacks.

5 Conclusion

In this paper we introduced a new attack method to apply Deep Learning techniques in a Non-Profiled context. We showed that it is possible to use Deep Learning and Neural Networks even when attacking a closed device where no profiling is possible. The possibility to use Deep Learning for Non-Profiled attacks offers several benefits: we showed that even in a Non-Profiled context, the translation-invariance property of Convolutional Neural Networks can be exploited against de-synchronized traces and that it is possible to perform Data Augmentation with MLP and CNN architectures to improve the results of Non-Profiled attacks. Using these properties, we obtained results where Non-Profiled Deep Learning-based attacks can outperform classic Non-Profiled attacks as CPA. This new attack method can also be applied to break high-order protected implementation, without applying any leakage combination pre-processing. The complexity snapshot that we provide, shows that this attack method is practical and may even be more efficient than some existing attacks in some cases, for instance when attacking masked implementations. Finally, this attack can be implemented in a few lines of code using Deep Learning frameworks such as Keras.

References

1. H. Maghrebi, T. Portigliatti, and E. Prouff, "Breaking cryptographic implementations using deep learning techniques." Cryptology ePrint Archive, Report 2016/921, 2016. https://eprint.iacr.org/2016/921.
2. E. Cagli, C. Dumas, and E. Prouff, *Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures*, pp. 45–68. Cham: Springer International Publishing, 2017.
3. Emmanuel Prouff and Remi Strullu and Ryad Benadjila and Eleonora Cagli and Cecile Dumas, "Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database." Cryptology ePrint Archive, Report 2018/053, 2018. https://eprint.iacr.org/2018/053.
4. P. C. Kocher, "Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems.," in *Advances in Cryptology - CRYPTO '96* (N. Koblitz, ed.), vol. 1109 of *Lecture Notes in Computer Science*, pp. 104–113, Springer, 1996.
5. S. Chari, J. R. Rao, and P. Rohatgi, *Template Attacks*, pp. 13–28. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.
6. J. Doget, E. Prouff, M. Rivain, and F.-X. Standaert, "Univariate side channel attacks and leakage modeling," *Journal of Cryptographic Engineering*, vol. 1, p. 123, Aug 2011.

7. W. Schindler, K. Lemke, and C. Paar, "A stochastic model for differential side channel cryptanalysis," in *Cryptographic Hardware and Embedded Systems – CHES 2005* (J. R. Rao and B. Sunar, eds.), (Berlin, Heidelberg), pp. 30–46, Springer Berlin Heidelberg, 2005.
8. G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle, "Machine learning in side-channel analysis: a first study," *Journal of Cryptographic Engineering*, vol. 1, p. 293, Oct 2011.
9. L. Lerman, R. Poussier, G. Bontempi, O. Markowitch, and F.-X. Standaert, "Template attacks vs. machine learning revisited and the curse of dimensionality in side-channel analysis," in *Revised Selected Papers of the 6th International Workshop on Constructive Side-Channel Analysis and Secure Design - Volume 9064*, COSADE 2015, (New York, NY, USA), pp. 20–33, Springer-Verlag New York, Inc., 2015.
10. L. Lerman, G. Bontempi, and O. Markowitch, "A machine learning approach against a masked aes," *Journal of Cryptographic Engineering*, vol. 5, pp. 123–139, Jun 2015.
11. P. C. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Advances in Cryptology - CRYPTO '99* (M. J. Wiener, ed.), vol. 1666 of *Lecture Notes in Computer Science*, pp. 388–397, Springer, 1999.
12. E. Brier, C. Clavier, and F. Olivier, "Correlation Power Analysis with a Leakage Model," in *CHES*, pp. 16–29, 2004.
13. B. Gierlichs, L. Batina, P. Tuyls, and B. Preneel, *Mutual Information Analysis*, pp. 426–442. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.
14. "ChipWhisperer Website." https://newae.com/tools/chipwhisperer/.
15. C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.
16. Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.
17. "Deep Learning website." http://deeplearning.net/tutorial/tutorial.
18. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.
19. C. M. Bishop, *Neural Networks for Pattern Recognition*. New York, NY, USA: Oxford University Press, Inc., 1995.
20. Y. Lecun and Y. Bengio, *Convolutional networks for images, speech, and time-series*. MIT Press, 1995.
21. K. O'Shea and R. Nash, "An introduction to convolutional neural networks," 11 2015.
22. "Keras framework." http://www.keras.io.
23. M.-L. Akkar and C. Giraud, "An Implementation of DES and AES, Secure against Some Attacks," in *CHES* (Çetin Kaya Koç, D. Naccache, and C. Paar, eds.), vol. 2162 of *Lecture Notes in Computer Science*, pp. 309–318, Springer, 2001.
24. T. S. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software," pp. 238–251.
25. M. Joye, P. Paillier, and B. Schoenmakers, "On Second-Order Differential Power Analysis.," in *Cryptographic Hardware and Embedded Systems - CHES 2005* (J. R. Rao and B. Sunar, eds.), vol. 3659 of *Lecture Notes in Computer Science*, pp. 293–308, Springer, 2005.
26. J. Waddle and D. Wagner, "Towards Efficient Second-Order Power Analysis," in *CHES*, pp. 1–15, 2004.
27. E. Prouff, M. Rivain, and R. Bevan, "Statistical Analysis of Second Order Differential Power Analysis," *IEEE Transactions on Computers*, vol. 58, pp. 799–811, June 2009.