Deep Learning based Side Channel Attacks in Practice

Houssem Maghrebi

Underwriters Laboratories, France
houssem.maghrebi@ul.com

Abstract. A recent line of research has investigated a new profiling technique based on deep learning as an alternative to the well-known template attack. The advantage of this new profiling approach is twofold: (1) the approximation of the information leakage by a multivariate Gaussian distribution is relaxed (leading to a more generic approach) and (2) the pre-processing phases such as the traces realignment or the selection of the Points of Interest (PoI) are no longer mandatory, in some cases, to succeed the key recovery (leading to a less complex security evaluation roadmap). The related published works have demonstrated that Deep Learning based Side-Channel Attacks (DL-SCA) are very efficient when targeting cryptographic implementations protected with the common side-channel countermeasures such as masking, jitter and random delays insertion. In this paper, we assess the efficiency of this new profiling attack under different realistic and practical scenarios. First, we study the impact of the intrinsic characteristics of the manipulated data-set (*i.e.* distance in time samples between the PoI, the dimensionality of the area of interest and the pre-processing of the data) on the robustness of the attack. We demonstrate that the deep learning techniques are sensitive to these parameters and we suggest some practical recommendations that can be followed to enhance the profiling and the key recovery phases. Second, we discuss the tolerance of DL-SCA with respect to a deviation from the idealized leakage models and provide a comparison with the well-known stochastic attack. Our results show that DL-SCA are still efficient in such a context. Then, we target a more complex masking scheme based on Shamir's secret sharing and prove that this new profiling approach is still performing well. Finally, we conduct a security evaluation of a batch of several combinations of side-channel protections using simulations and real traces captured on the ChipWhisperer board. The experimental results obtained confirm that DL-SCA are very efficient even when a cryptographic implementation combines several side-channel countermeasures.

Keywords: Deep Learning based Side-Channel Attacks, Data Dimensionality, Data Scaling, Artificial Noise, Side-Channel Countermeasures, Shamir's Secret Sharing, Combination of Countermeasures.

1 Introduction

1.1 Profiling Side-Channel Attacks

Side Channel Attacks (SCA) are nowadays well known and most designers of secure embedded systems are aware of them. Among the SCA, profiling attacks play a fundamental role in the context of the security evaluation of cryptographic implementations. Indeed, the profiling attacks provide a security assessment in the worst-case scenario. That is, the adversary has full-knowledge access to a copy of the target device and uses it to characterize the physical leakage. Besides, the profiling attacks consist of two steps: (1) a profiling step during which the adversary estimates and characterizes the distribution of the leakage function and (2) an attack phase during which he performs a key recovery attack on the target device.

Several profiling approaches have been introduced in the literature. A common profiling side channel attack is the template attack proposed in [11] which is based on the Gaussian assumption[1]. It is known as the most powerful type of profiling in SCA context when (1) the Gaussian assumption is verified and (2) the size of the leakage observations is small (typically smaller than $1,000$).

When the Gaussian assumption is relaxed, several profiling side-channel attacks have been suggested including techniques based on Machine Learning (ML). Actually, ML models make no assumption on the probability density function of the data. For example, random forest model builds a set of decision trees that classifies the data-set based on a voting system [24] and Support Vector Machine (SVM)-based attack discriminates data-set using hyper-plane clustering [20]. Besides, one of the main drawbacks of the template attacks is their high data complexity [12] as opposed to the ML-based attacks which are generally useful when dealing with high-dimensional data [24].

Following the current trend in ML area, recent works have investigated the use of Deep Learning (DL) models as an alternative to the existing profiling SCA [7, 9, 27, 36]. The related practical results have demonstrated that these techniques are very efficient to conduct security evaluations of embedded systems even when some well-known countermeasures are involved to ensure protection against SCA. In the following, we provide an overview of deep learning techniques and then we describe the results derived from the recent investigations on the use of DL in side-channel context.

1.2 Classification of Deep Learning Techniques

Among DL models, three classes may be distinguished:

- the *fully connected networks*: are the basic type of neural networks. The major advantage of fully connected networks is that they are "structure-agnostic". That is, no special assumptions need to be made about the input

[1] The Gaussian assumption stipulates that the distribution of the leakage when the algorithm inputs are fixed is well estimated by a Gaussian Law.

data. A fully connected network can be described as a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ such that each output depends on the n inputs. The simplest fully connected neural network is the *perceptron* [5]. It is a linear classifier that uses a learning algorithm to tune its weights and minimize a so-called *loss function*. It is possible to connect several perceptrons between each other to build a classifier for more complex data-sets. The resulting fully connected network is called MultiLayer Perceptron (MLP) [5].

- The *features extractor networks*: are often used in image recognition and classification. The goal is to learn higher-level and deep features in data that are the most useful for the classification and/or target detection. This can be done via computing a convolution between the data and some filters followed by a down-sampling operation to keep only the most informative features. A typical example of features extractor networks is the Convolutional Neural Network (CNN) [23, 31]. Typically, a CNN is composed of alternating layers of (1) locally connected convolutional filters and (2) down-sampling, followed by a fully connected layer that works as a classifier (*a.k.a. SoftMax* layer).
- The *time dependency networks*: are a set of neural networks that differs from the other ones in their ability to process information shared over several time-steps. Indeed, in a traditional neural network, we assume that all inputs (and outputs) are mutually independent. However, for some applications, this assumption is unrealistic[2]. So, the core idea of this type of networks is that each neuron will infer its output from both the current input and the output of previous neurons. This feature is quite interesting in side-channel context since the leakage is spread over several time samples. The Recurrent Neural Networks (RNN) [18] and especially the Long-and-Short-Term-Memory units (LSTM) [21] are the most suitable time dependency neural networks.

1.3 Existing Works on Deep Learning based Side-Channel Attacks

Several works have investigated the application of DL techniques to conduct security evaluations of cryptographic implementations. These contributions have focused mainly on:

- **defeating both unprotected and protected symmetric cryptographic implementations.** In the seminal work on the use of DL techniques in SCA context [27], Maghrebi *et al.* demonstrated that the Deep Learning based SCA (DL-SCA) are very efficient to break both unprotected and masked AES implementations. The authors experienced several types of DL models (MLP, CNN, LSTM and stacked Auto-Encoders [28]) and the obtained results highlighted the overwhelming advantage of this profiling technique compared to the well-known template attack. Later, Cagli *et al.* proposed an end-to-end profiling approach based on CNN that is efficient in the presence of trace misalignment [7]. This property is of a great practical interest

[2] For instance, if we want to predict the next word in a sentence then the previous words are required and hence there is a need to remember them.

since it helps to streamline the evaluation process as no pre-processing of the traces is needed. Recently, Prouff *et al.* revisited different methodologies to select the most suitable *hyper-parameters*, *i.e.* the parameters that define a DL architecture (*e.g.* number of layers, number of epochs, *etc.*), for the CNN and MLP DL models [36]. More interestingly, the authors published an open database, named ASCAD, that contains electromagnetic traces of a masked AES implementation along with the source code of the used neural network architectures. Nowadays, this database is serving as a common basis for the side-channel community to progress on this DL-SCA topic.

- **Defeating secure asymmetric cryptographic implementations.** In [9], authors presented several profiling SCA against a secure implementation of the RSA algorithm. Indeed, the targeted implementation relies on a certified EAL4+ arithmetic co-processor and is protected with the classical side-channel countermeasures (blinding of the message, blinding of the exponent and blinding of the modulus). Through their practical experiments, the authors pinpointed the high potential of deep learning attacks (and in particular the CNN models) against secure RSA implementations.
- **Using the DL-SCA in non-profiling context.** Timon suggested in [42] a methodology to apply DL as a partition based SCA [40]. The core idea consists in partitioning the collected traces along with their labels according to a selection function that depends on the key hypotheses. Then, for each key hypothesis, a DL training (based on CNN or MLP models) is performed to evaluate the consistency of the obtained partitions. Finally, to recover the good key value, the author proposed several kinds of metrics that are based either on the used DL network input layers (*i.e.* the obtained weights on the first layer) or the DL training outcomes (*i.e.* the *accuracy* and the *loss*). The different reported experiments have shown the efficiency of this approach against higher-order masking implementations compared to the classical non-profiling SCA (*i.e.* higher-order Correlation Power Analysis [34]).
- **Using DL as a Point of Interest (PoI) selection method.** In several works [19, 29, 42], researchers tried to answer the question of whether the DL can be used as a leakage assessment method? Indeed, the question was answered positively and several methodologies based on different strategies were suggested: the analysis of the gradient of the loss function used during the training [29], the application of the well-known *attribution methods* as suggested in [19] and the exploitation of the *sensitivity analysis* techniques [42]. The different obtained results have shown that DL based PoI selection method is at least as good as the state-of-the-art leakage assessment methods,*e.g.* the Signal to Noise Ratio (SNR).

1.4 Contributions

The aforementioned investigations have shown that DL-SCA are an interesting research avenue to explore. Following this current trend, we propose in this work to assess the efficiency of DL-SCA under different realistic and practical scenarios. Mainly, the contributions of this paper are to study the:

- **impact of the distance between the PoI:** in Sec. 3, we discuss the impact of the distance in time samples between the leakages of the mask and the masked data on the efficiency of DL-SCA. We show that some architectures, especially the CNN, are sensitive to this practical issue and the corresponding hyper-parameters should be carefully tuned to achieve an efficient key recovery.
- **Impact of the dimensionality of the data:** despite their ability to process high-dimensional data, we demonstrate in Sec. 4 that targeting a large area of interest could negatively affects the efficiency of DL-SCA. Moreover, we provide some practical hints on how to overcome this issue.
- **Scaling of the data in DL context:** in Sec. 5, we prove through practical experiment the great interest of scaling the data before performing a DL-SCA evaluation. Indeed, performing this data pre-processing step would lead to a good trade-off between computation time and attack efficiency.
- **Impact of adding artificial noise:** although counter-intuitive, we prove in Sec. 6 that adding artificial noise can help DL-SCA (based on several DL models) to avoid *over-fitting*, *i.e.* the DL architecture perfectly fits the training data-set but is not able to generalize its predictions to other data-sets, and enhance the key recovery phase results. This artificial noise addition during the training can be seen as a regularization layer similar to the data augmentation method studied in [7].
- **Impact of the leakage model:** we discuss in Sec. 7 the efficiency of DL-SCA when the leakage model deviates from the idealized models (Hamming weight and value leakage models). The obtained results demonstrate that DL-SCA are not sensitive to this practical issue. Moreover, the comparison with stochastic models, when considering unprotected and masked implementations, shows that both attacks achieve similar results.
- **DL-SCA Versus Shamir's secret sharing:** in Sec. 8, we perform DL-SCA on an implementation of the Shamir's secret sharing. The obtained results demonstrate that this countermeasure is vulnerable to DL-SCA and hence a security designer has to increase the order of sharing to strengthen the resistance.
- **DL-SCA Versus common SCA countermeasures:** we assess in Sec. 9 two side-channel countermeasures (shuffling and 1-amongst-N) against DL-SCA. The simulation and practical experiments prove, as expected, that these countermeasures are vulnerable to these profiling attacks. Besides, we discuss the better DL models to consider depending of the effect of the implemented countermeasure.
- **DL-SCA Versus combined SCA countermeasures:** in Sec. 10, the DL-SCA are evaluated in a more realistic context where several countermeasures are combined to ensure protection. Our practical experiments highlight that DL-SCA are still efficient in such a context.

Moreover, for all our experiments, along with the commonly used DL models in profiling SCA context (*i.e.* MLP and CNN), we consider the LSTM and demonstrate that this model could be a very interesting alternative in some scenarios.

To enable the reproducibility of our results, we provide in Appendix A a detailed description of the used DL architectures. Furthermore, the implementations used for our practical experiments are publicaly available on GitHub [3].

2 Notations and Evaluation Methodology

2.1 Notations

Throughout this paper, we use the upper-case letter X to denote random variables and $\mathbf{X}$ in bold for random vectors. The corresponding lower-case letter x is used to denote realizations of X.

2.2 Attacker Profile

Since we are dealing with profiling attacks, we assume an attacker who has full control of a training device during the profiling phase and is able to measure the physical leakage during the execution of a cryptographic algorithm (*a.k.a.* the training data-set). Then, during the attack phase, the adversary aims at recovering the unknown secret key, processed by the same device, by collecting a new set of the physical leakage (*a.k.a.* the attack data-set). In addition, the adversary collects an extra data-set called *validation* data-set (different from the attack data-set). Indeed, it is worthy to highlight that having a validation data-set is essential when dealing with DL [17] as it allows the user to detect if there is an over-fitting effect.

2.3 Deep Learning Architectures Used

For our experiments, we target the CNN and the MLP models which are the most often used DL models by the SCA community. For the sake of comparison, we consider the LSTM model as well due to its ability to process information shared over several time-steps which is very suitable in SCA context. For the CNN, we use two different architectures: the first one consists of 2 convolutional layers, denoted by CNN_2LAYERS, while the second one consists of 3 convolutional layers[3], CNN_3LAYERS. The used MLP and LSTM architectures are denoted receptively MLP and LSTM.

For each experiment, we provide the hyper-parameters of each used DL model to ease the reproducibility of our results. Regarding the *gradient descent optimization* method (also called *optimizer*), we use the *adam* approach and the *categorical cross-entropy* as a loss function for all the experiments performed in this work. This choice is motivated by the fact that these functions provide good results in terms of classification and matching.

Finally, our implementations of DL models have been developed with the Keras library [2] (version 2.2.4) and we run the training using a PC equipped with $128GB$ of RAM and a gamer market GPUs Nvidia GTX 1080 Ti.

[3] For our experiments, when these two architectures achieve similar results, we only illustrate one of the two results for clarity reasons on our figures.

2.4 Targeted Operation

Regarding the targeted operation, we consider for all the experiments, one or several AES Sbox outputs of the first round: $Z = Sbox[P \oplus k^*]$ where P and k^* respectively denote the plaintext and the secret key. We motivate our choice towards targeting this non-linear operation by the facts that it is a common target in side channel analysis and that it has a high level of confusion.

2.5 Evaluation Metric

For the different experiments (performed using simulations or on real devices) presented in the sequel, we consider a fixed attack setup. In fact, each attack is conducted on 100 different sets of traces. Then, we compute the average rank of the correct key among all key hypotheses (*a.k.a.* the *guessing entropy metric* [41]).

3 Impact of the Distance between Masking Leakage and Masked Data Leakage on the Efficiency of DL-SCA

3.1 Context and Experimental Set-up

In this section, we consider a first-order masked implementation leaking the value of the manipulated data. Our goal is twofold: (1) assess the efficiency of the DL-SCA with respect to the distance between the leakages of the mask and the masked data and (2) suggest some hints on how to tune the hyper-parameters of the used DL models to enhance the key recovery results in this context.

To do so, we define hereafter our set-up to generate L simulated traces $(\mathbf{T_i})_{1 \leq i \leq L}$, of $S = 100$ time samples each, corresponding to the manipulation of the mask M and the masked sensitive data $Z \oplus M$:

$$\mathbf{T_i}[s] = \begin{cases} M + \mathcal{N}(0, \sigma) & \text{if } s = 1, \\ Z \oplus M + \mathcal{N}(0, \sigma) & \text{if } s = \text{the masked data index}, \\ R + \mathcal{N}(0, \sigma) & \text{otherwise}, \end{cases}$$

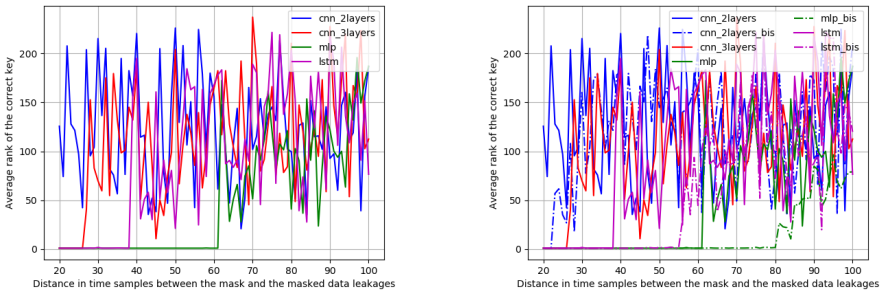
where $\mathcal{N}(0, \sigma)$ denotes a white Gaussian noise of null mean and standard deviation $\sigma = 0.5$ and R denotes a random integer in $[0, 255]$. That is, the mask is always leaking on the first time sample of each trace. Then, for each possible index of the masked data leakage (in $[20, 100]$), we do the following:

1. generate $L = 10,000$ traces for the profiling phase, $L = 100$ traces for the validation phase and $L = 500$ traces for the attack phase following the above-mentioned simulation set-up.
2. Train the different DL models.
3. Perform the key recovery and compute the average rank of the correct key.

The different hyper-parameters of the considered DL models are provided in Tab. 2 in Appendix A. We stress the fact that the hyper-parameters are kept fixed for the whole experiment.

3.2 Experimental Results

The obtained results are shown in Fig. 1 (a). They prove that the distance between both leakages has an impact on the efficiency of the key recovery results[4]. Indeed, the farther the distance is, the worse the key recovery is. In the meantime, one can conclude that the MLP in this context is outperforming the other DL models. This could be explained by the fact that the MLP is a fully connected network and hence all the time samples (including the interesting ones *i.e.* the leakages of the mask and the masked data) are combined between each other which conceal the masking effect. Regarding the CNN, the key recovery efficiency highly depends on the length of the convolutional filter used with respect to the distance between the leakages. For instance, the CNN_2LAYERS architecture contains a convolutional filter of length 16 on the first layer (see Tab. 2 in Appendix A). Now, since the minimum distance between the mask and the masked data leakages is 20 time samples, these leakages are never combined together and as a consequence, the key recovery will fail as confirmed in Fig. 1 (a).



(a) Basic DL architectures.

(b) Enhanced DL architectures.

Fig. 1: Evolution of the correct key rank (y-axis) when increasing the distance between the mask and the masked data leakages (x-axis).

To confirm our claim for the CNN and to investigate which are the hyper-parameters that can enhance the key recovery results for the MLP and the LSTM, we repeat the experiment using the same generated traces while tuning the hyper-parameters of the DL architectures as follow:

- increase the length of the convolutional filter on the first layer for the CNN_2LAYERS (from 16 to 32), and
- increase the number of neurons (*a.k.a.* units) in LSTM and MLP.

[4] and also on the accuracy and the loss metrics. The corresponding values are not reported in this paper for clarity reasons.

The new DL architectures are denoted respectively CNN_2LAYERS_BIS, LSTM_BIS, and MLP_BIS and the complete description of their respective hyper-parameters is detailed in Tab. 4 in Appendix A. The new obtained results are shown in Fig. 1 (*b*). They confirm our expectations. Indeed, the key recovery is enhanced for the new used DL architectures compared with the previous ones[5]. For instance, when using the CNN_2LAYERS_BIS it is possible to recover the good value of the key when the distance between both leakages is at most 23 time samples.

3.3 Takeaway Messages

In this section, we provide some practical hints to consider to enhance either the attack efficiency or the resistance of a cryptographic implementation.

From an Adversary's Perspective. Regarding the CNN architecture, the convolutional filter (in particular on the first layer) should be designed such that its length covers the most interesting PoI to enable the combination of the corresponding leakages. Regarding MLP and LSTM, increasing the number of units can enhance the training and hence the key recovery phase. However, this may lead to an over-fitting effect and/or a heavy computation burden during the training phase. So, an adversary has to tune carefully this parameter to mitigate these issues. Another approach would consist in increasing either the number of traces used for the profiling or the number of epochs.

From a Security Developer's Perspective. To strengthen the resistance of masked implementations against DL attacks, one solution would consist in spacing out the leakages of the mask and the masked data. For instance, this can be done by inserting some fake (or genuine) computations that do not involve the usage of the mask and/or by adding some random delays.

4 Impact of the Data Dimensionality on the Efficiency of DL-SCA

4.1 Context and Experimental Set-up

In this section, our goal is to answer the question of whether the dimensionality of the collected traces has an impact on the efficiency of DL-SCA. In fact, it has been demonstrated in several papers (*e.g.* [7, 27]) that one of the main interest of DL-SCA is their ability to process high-dimensional data without any application of the well-known dimensionality reduction techniques (PCA, LDA, KDA [8]) to select a small portion of PoI. To evaluate the tolerance of DL-SCA to the data dimensionality, we consider a first-order masked implementation leaking

[5] Please note also that the accuracy using these new DL architectures is enhanced as well.

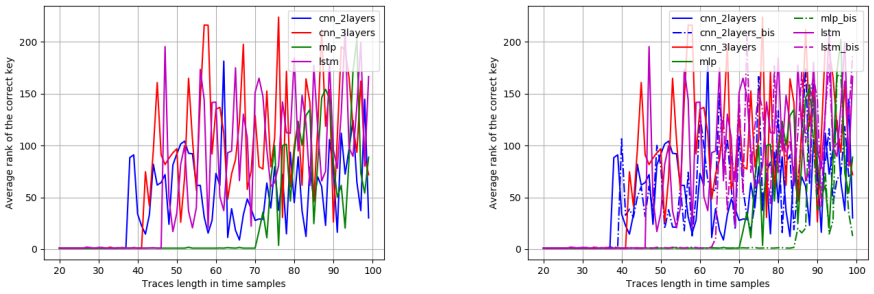
the value of the manipulated data. For this experiment, to generate L simulated traces $(\mathbf{T_i})_{1 \leq i \leq L}$, of S time samples each, we use the following protocol:

$$\mathbf{T_i}[s] = \begin{cases} M + \mathcal{N}(0, \sigma) & \text{if } s = 1, \\ Z \oplus M + \mathcal{N}(0, \sigma) & \text{if } s = 4, \\ R + \mathcal{N}(0, \sigma) & \text{otherwise,} \end{cases}$$

where $\mathcal{N}(0, \sigma)$ denotes a white Gaussian noise of null mean and standard deviation $\sigma = 0.5$ and R denotes a random integer in $[0, 255]$. That is, the mask and the masked data are always leaking at the same time samples (the first and the fourth time samples respectively). Now, the idea is to increase the length of the traces S in $[20, 100]$ and for each length value we perform the following:

1. generate $L = 10,000$ traces for the profiling phase, $L = 100$ traces for the validation phase and $L = 500$ traces for the attack phase following the above-mentioned simulation protocol.
2. Train the different DL models.
3. Perform the key recovery and compute the average rank of the correct key.

The different hyper-parameters of the considered DL models are provided in Tab. 2 in Appendix A.



(a) Basic DL architectures.

(b) Enhanced DL architectures.

Fig. 2: Evolution of the correct key rank (y-axis) when increasing the traces length.

4.2 Experimental Results

The obtained results are shown in Fig. 2 (a). The used DL models have different tolerance to the data dimensionality increase. Indeed, one can see that the MLP is less sensitive compared to the CNN and the LSTM. However, when the number of the non-informative points increases (*i.e.* when the trace length is greater than 75 time samples with only two informative time samples), the key recovery

fails for all the considered DL models. This leads us to answer positively the question regarding the tolerance of DL-SCA to the data dimensionality. Indeed, the shorter the traces are, the better the attack results are.

There is now the new question arising of whether it is possible to tune the hyper-parameters of the DL architectures to enhance the key recovery results even in the presence of high-dimensional data? To answer this question, we consider the same enhanced DL architectures described in the previous section and whose hyper-parameters are described in Tab. 4 in Appendix A. That is, we re-perform the experiment when using a CNN network with a bigger convolutional filer (CNN_2LAYERS_BIS) and an MLP and LSTM (LSTM_BIS and MLP_BIS) with more neural units. The new results are plotted in Fig. 2 (*b*).

Regarding the CNN architecture, increasing the length of the convolutional filter slightly increases the attack efficiency. This is expected since, in this context, increasing the convolutional filter size will add more non-informative samples in the processing which do not help the training and the key recovery phases. However, increasing the number of units (as done for the MLP and the LSTM architectures) allows increasing the attack efficiency when the number of the non-informative points increase. Indeed, these enhanced architectures allow a better analysis of the processed data.

4.3 Takeaway Messages

Following our experiments, we describe hereafter some practical recommendations that should be followed to either enhance the DL-SCA results or to strengthen the resistance of a cryptographic implementation against these attacks.

From an Adversary's Perspective. Despite the ability of DL-SCA to process high-dimensional data, the adversary has to target a reduced area of interest for the training and the attack. This actually can be done through a good inspection of the traces shape to identify the most relevant patterns. In addition, the knowledge of the implementation source code, when available, is of great interest since it allows the adversary to precisely locate the suitable window for the attack. When the adversary cannot select a small area of interest, another option is to wisely increase the number of units (and/or layers) used for the DL architectures to avoid over-fitting effects and/or heavy computation time. In addition, applying a PoI selection method can be emphasized in this case.

From a Security Developer's Perspective. To enhance the security of a cryptographic implementation, the developer has to hide the sensitive operations so that the adversary is unable to distinguish clearly the area of interest when inspecting the traces. This can be achieved for example by applying the well-known 1-amongst-N countermeasure, *i.e.* executing the sensitive computation randomly amongst $(N-1)$ fake computations (*a.k.a.* dummy computations) identical to the genuine one. In such a context, the adversary has to select a

large window to perform his attack. In Sec. 9.1, we assess the security of this countermeasure with respect to DL-SCA.

Another parameter which can affect the efficiency of DL-SCA is the sampling rate used during the acquisition of the data. Indeed, this parameter has a direct impact on the dimensionality of the data to be processed during the training and the attack phases. We keep the study of the sensitivity of DL-SCA with respect to a variation of the sampling rate as future work.

5 Importance of Data Scaling in DL Context

5.1 Context and Experimental Set-up

Data pre-processing is a crucial step for any data analysis, especially for deep learning analysis. In fact, this step can ease the difficulty of modeling and therefore enhance the outputted results. In several works and tutorials related to DL techniques (*e.g.* [43]), scaling the data-sets before feeding them to the DL architectures is highly recommended. Indeed, different scaling methods can be used [43]: the min-max scaling, the variance scaling, $\|.\|^2$ normalization, *etc.* The most common scaling method used in DL context is the min-max method, that is, having data whose values are within the range of 0 and 1. In this section, our goal is to provide some evidences that scaling the data can ease significantly the classification.

To do so, we consider the ASCAD database presented in [36]. Indeed, the authors in [36] have presented a large variety of benchmarks that have been performed to find the suitable hyper-parameters for a CNN and an MLP that guarantee a good trade-off between the SCA-efficiency and the training time. At the end of these benchmarks, authors end up with two architectures that are described in the following:

- the CNN_BEST is composed of 5 convolutional layers each followed by an average pooling layer, 2 dense layers (of $4,096$ units each) and a SoftMax layer.
- The MLP_BEST is composed of 5 dense layers (of 200 units each) and a SoftMax layer.

The attack results obtained when running the above architectures confirmed their efficiency in breaking a masked AES implementation (and even when a random delay is inserted to de-synchronize the traces). However, one can remark that the proposed architectures are too complex (many layers and many units are used) with respect to the obvious leakage detected by the authors when performing the SNR leakage assessment method. Moreover, when analyzing the different Python scripts provided by the authors in [4] to run the training and the attack phases, we notice that no data scaling was performed before experimenting their benchmarks.

Intuitively, we believe that these complex architectures are a direct consequence of not considering the scaling data step. To confirm this claim, we consider

the synchronized traces of ASCAD database (chosen for simplicity reasons) and we perform the following:

- select $1,000$ traces from the attack data-set to build our validation data-set and use the remaining ones for the attack.
- Scale the traces of the training, validation and attack data-sets. The applied method consists of removing the mean of the traces and then applying the min-max approach.
- Run the training and the key recovery when considering 3 "simple" DL architectures:
  - a CNN that is composed of 3 convolutional layers, one dense layer, and a SoftMax layer.
  - An MLP that consists of 2 dense layers (of 20 and 50 units respectively) and one SoftMax layer.
  - An LSTM that contains 2 LSTM layers (of 26 units each) and a SoftMax layer.

The complete description of the hyper-parameters of the aforementioned architectures is provided in Tab. 3 in Appendix A.

5.2 Experimental Results

The results of our investigation are plotted in Fig. 3. For the sake of comparison, we add the results obtained for the CNN_BEST and the MLP_BEST architectures when running the Python scripts provided in [4].

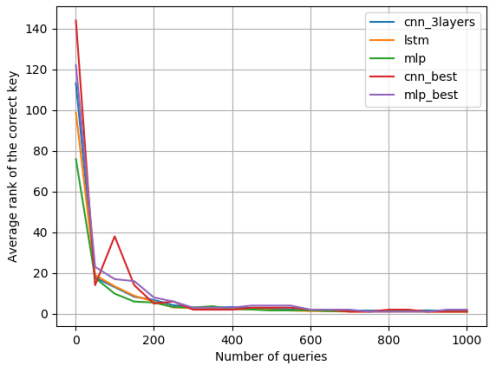


Fig. 3: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when targeting the ASCAD database.

From Fig. 3, one can conclude that when an appropriate scaling of the data is performed, the attack results obtained with our "simple" DL architectures

are as efficient as the ones obtained with the complex architectures provided in [36]. This result confirms that scaling the data allows to ease the classification and hence to avoid heavy computation time during the training while preserving efficient matching results.

We stress the fact that our goal was not to find the optimal "simple" DL architectures to consider for the ASCAD database but was more to pinpoint the importance of data scaling in DL context.

5.3 Takeaway Messages

From an adversary's perspective, it is crucial to scale the data as this would lead to a good trade-off between computation time and DL-SCA efficiency. This step is of great interest especially in the context of a *Common Criteria* security evaluation where the time required to perform the attack has an impact on its final rating [1].

6 Impact of the Noise on the Efficiency of DL-SCA

6.1 Context and Experimental Set-up

Recently, Kim *et al.* have demonstrated in [22] that adding artificial noise to the input data can be beneficial to the performance of the CNN network as it acts as a regularization term and hence prevents over-fitting. Doing so, the authors have enhanced their DL-SCA efficiency when targeting different data-sets (ASCAD, DPA contest v4). The idea consists in either adding a Gaussian noise tensor in the batch normalization layer applied on the input data during the training or adding it directly to the input data.

6.2 Simulation Set-up and Results

Following these investigations, we study in this section if this result can be generalized to other types of DL models (*i.e.* MLP and LSTM). To do so, we use exactly the same simulation set-up described in Sec. 4 with the following two major differences:

1. the length of the simulated traces is a fixed ($S = 25$ time samples) and,
2. we reduce the number of attack traces to $L = 10$.

The idea behind is to force the DL-SCA (based on the 4 selected DL architectures) to fail in recovering the correct value of the key. Then, by adding an artificial noise during the training, the goal is to confirm if this feature can enhance the key recovery phase.

To add the noise, we use the *GaussianNoise(stddev)* layer from Keras library [2], where *stddev* is the standard deviation of the artificial noise. This layer is described in [2] as a useful practice to mitigate over-fitting and can be seen as a form of the data augmentation method. For our experiment, this layer

is added after the first layer[6] of each used DL architectures as described in Tab. 5 in Appendix A. It is worthy to highlight that our approach is a different from the one followed in [22] in the sense that the noise is neither added after a batch normalization layer nor the traces are artificially modified; we simply use a noise layer provided by Keras library to handle this specific usage.

The experiment consists in increasing the artificial noise standard deviation *stddev* in $[0.1, 1]$ and then to perform the training of the DL architectures and the key recovery. The obtained results are depicted in Fig. 4.

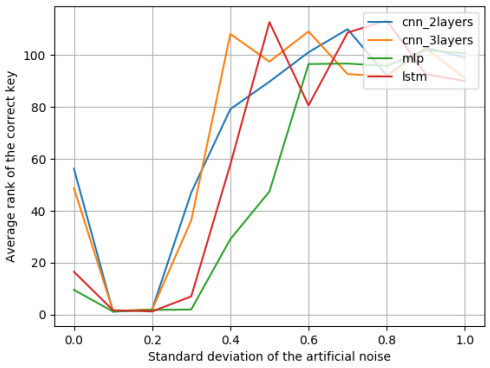


Fig. 4: Evolution of the correct key rank (y-axis) according to an increasing standard deviation of the added artificial noise.

The results prove that for some values of the artificial noise standard deviation *stddev* in $\{0.1, 0.2, 0.3\}$ the attack results are enhanced for the 4 DL architectures used. More interestingly, when $stddev = 0.1$, it is possible to break the masked implementation within 10 traces while when this feature is deactivated (*i.e.* $stddev = 0$) the key is not recovered. These results are not only in-line with those described in [22] but also demonstrate that adding artificial noise enhances the training and the key recovery outcomes of MLP and LSTM architectures as well.

6.3 Practical Set-up and Results

To validate the simulation results, we consider a real-world context by targeting the ASCAD database [36]. The idea consists in selecting some DL architectures for which either the attack results were not as efficient as the ones evaluated in Sec. 5 (whose results are illustrated in Fig. 3) or an over-fitting effect was

[6] Other options can be considerd regading the position of the noise layer in the used DL architectures. Our choice is motivated by simplicity reasons.

observed. Then, the goal is to enhance their efficiency when adding an artificial noise. The different hyper-parameters of the targeted DL architectures without and with artificial noise addition are available respectively in Tab. 6 and Tab. 7 in Appendix A. We stress the fact that the traces from the ASCAD database were scaled following the same method described in Sec. 5.

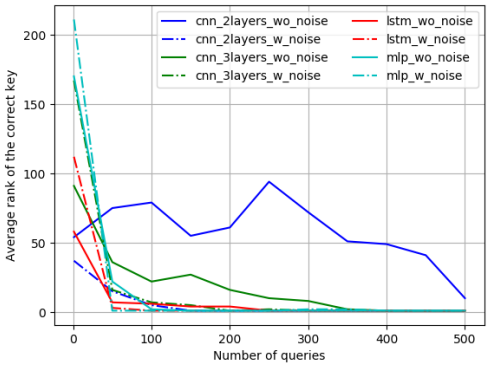


Fig. 5: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when targeting the ASCAD database.

The obtained DL-SCA results are plotted in Fig. 5. They demonstrate that adding artificial noise can significantly enhance the data classification and consequently the key recovery results.

6.4 Takeaway Messages

To enhance the DL-SCA, it is highly recommended to add Gaussian noise during the training especially when an over-fitting effect is detected. More generally speaking, adding some regularization layers (*e.g.* dropout, batch-normalization, noise) or applying data augmentation method as highlighted in [7] is of great interest to avoid over-fitting and to guarantee a good training which results in an efficient key recovery phase.

7 Impact of the Leakage Model on the Efficiency of DL-SCA and Comparison with the Stochastic Models

7.1 Context and Experimental Set-up

In SCA context, it is often assumed that a device leaks information following the Hamming weight or the value of the processed data. This assumption is quite realistic and many security analyses in the literature have been conducted

following this model [6, 32]. However, this assumption is not complete in real hardware [26], due to small load imbalances, process variations, routing, *etc.* For instance, authors in [26] have characterized, using a *stochastic approach* (*a.k.a.* Linear Regression Analysis) [14, 37, 38], the leakage of four AES Sbox outputs when implemented in three different devices. The obtained results prove that the leakage is very unbalanced for each Sbox output and hence the Hamming weight and the value leakage models are unsound in practice.

So far, the authors of the published papers related to DL-SCA have only considered these two leakage models [7, 27]. To the best of our knowledge, they never studied the real-world scenario where the leakage model deviates from these idealized models. In this section, our goal is to assess the efficiency of the DL-SCA in such realistic context. To do so, we select four types of leakage model functions[7] denoted (f) that are listed in the following:

- Hamming weight leakage model: $f(x) = \sum([x\&(1 \ll i) > 0 \text{ for i in [1, n]})$,
- value leakage model: $f(x) = x$,
- 1^{st}-order random leakage model: $f(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i \cdot x_i$,
- 2^{nd}-order random leakage model: $f(x) = \alpha_0 + \sum_{i=1}^{n} \alpha_i \cdot x_i + \sum_{\substack{i_1, i_2 = 1 \\ i_1 < i_2}}^{n} \alpha_{i_1, i_2} \cdot x_{i_1} \cdot x_{i_2}$,

where x is an n-bit value, x_i is its i^{th} bit value and the coefficients α_i are random real values uniformly picked from the interval [-1, 1].

Moreover, we target an unprotected and a first-order masked implementation. Our experimental set-up to generate the corresponding simulated traces $(\mathbf{T_i})_{1 \leq i \leq L}$, of $S = 20$ time samples each, is described in the following:

- unprotected implementation:

$$\mathbf{T_i}[s] = \begin{cases} f(Z) + \mathcal{N}(0, \sigma) \text{ if } s = 9, \\ f(R) + \mathcal{N}(0, \sigma) \text{ otherwise,} \end{cases}$$

- masked implementation:

$$\mathbf{T_i}[s] = \begin{cases} f(M) + \mathcal{N}(0, \sigma) & \text{if } s = 9, \\ f(Z \oplus M) + \mathcal{N}(0, \sigma) & \text{if } s = 15, \\ f(R) + \mathcal{N}(0, \sigma) & \text{otherwise,} \end{cases}$$

where $\mathcal{N}(0, \sigma)$ denotes a white Gaussian noise of null mean and standard deviation $\sigma = 0.5$ and R denotes a random integer in $[0, 255]$.

Now, for each considered leakage model function and targeted implementation, we execute the following process:

1. generate $L = 20,000$ traces for the profiling phase, $L = 100$ traces for the validation phase and $L = 1,000$ traces for the attack phase following.

[7] Please note that we use the same leakage models as those studied in [13].

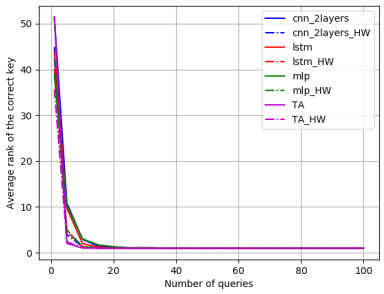
2. Train the different DL models. It is worthy to highlight that for each used DL architecture, we consider two strategies for the data labeling during the training phase. The first one consists in labeling the profiling traces with respect to the value of the sensitive data. The second strategy is to label the traces according to the Hamming weight of the sensitive value. The corresponding DL architectures, called HW-DL architectures in the sequel, are denoted by the suffix "_HW". The goal behind is to check what is the most suitable data labeling strategy to adopt for a given leakage model function.
3. Perform the key recovery and compute the average rank of the correct key. For the sake of comparison, we perform the template attack (for the traces generated according to Hamming weight and value leakage models) and the stochastic attack (for the traces generated according to the first and second-order random leakage model).

The different hyper-parameters of the considered DL architectures are provided in Tab. 2 and Tab. 8 in Appendix A.

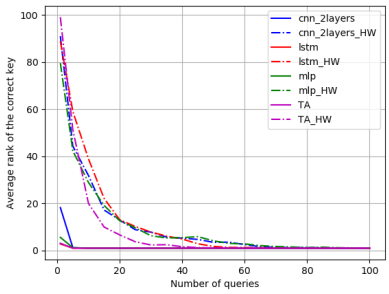
7.2 Experimental Results: Unprotected Implementation

The obtained results for the unprotected implementations are shown in Fig. 6. From this figure, the following observations could be emphasized:

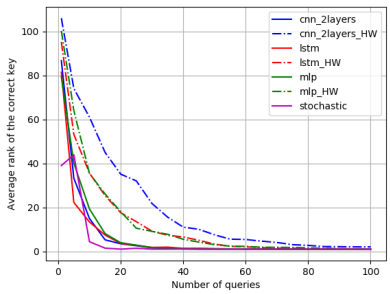
- as expected, when the device leaks following the Hamming weight or the value leakage models, both template attack and DL-SCA achieve good results during the key recovery phase.
- When the leakage model deviates from the idealized models (Hamming weight or value), the DL-SCA still succeed to recover the good value of the key. Interestingly, when the random leakage function is quadratic (*i.e.* 2^{nd}-order), the obtained results for the DL-SCA are much better (in terms of number of traces required to succeed the attack) compared to the scenario where the random leakage function is linear (*i.e.* 1^{st}-order). This could be explained by the fact that the quadratic combinations of the bit-coordinates of the sensitive data are more informative than the linear ones which helps the DL architectures to better discriminate and classify the data during the training. Overall, one can conclude that DL-SCA are not sensitive to a leakage function deviation from the idealized models.
- When the random leakage function is linear, the stochastic attack slightly outperforms the DL-SCA. However, when the random leakage function is quadratic, both attacks are of the same effectiveness.
- Regardless of the leakage model used to generate the traces, labeling the data according to the value of the sensitive variable is more efficient than applying the Hamming weight labeling strategy. Indeed, it is clear from Fig. 6 that when using the HW-DL architectures, the good key value is recovered, but more traces are needed (compared to the value data labeling strategy). Besides, using the Hamming weight reduces the number of classes to predict



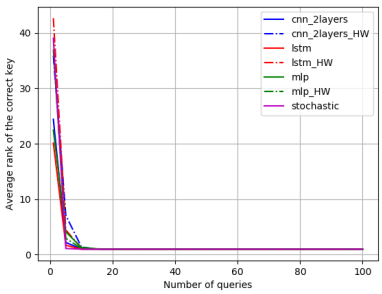
(a) Hamming weight leakage.



(b) Value leakage.



(c) First-order random leakage.



(d) Second-order random leakage.

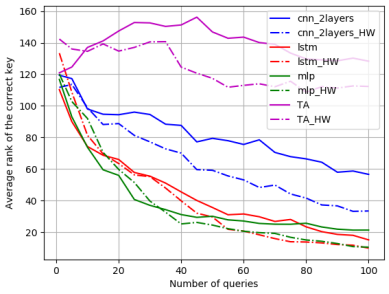
Fig. 6: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when targeting an unprotected implementation.

(from 256 to 9 in our case). As a consequence, the classification of the data is less complex (only a few numbers of epochs are required for the training phase). However, this has an impact on the efficiency of the network to discriminate the good value of the sensitive data during the attack phase. This observation is in-line with the results obtained in [36] where the authors compared these two labeling strategies on the ASCAD database.

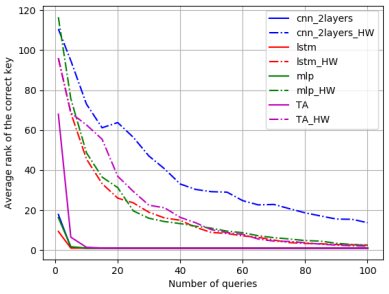
7.3 Experimental Results: First-Order Masked Implementation

The obtained results for the masked implementation are shown in Fig. 7. In the following, we provide a summary of the observations emerged from the analysis of the experiment outcomes:

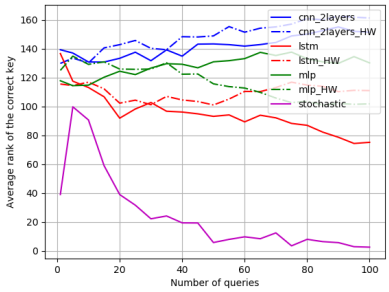
– DL-SCA (and template attack) require more traces to recover the correct value of the key when the device leaks the Hamming weight of the data compared to the case where it leaks the value. This result is expected since the profiling attacks are more discriminating when the number of labels/-classes is greater.



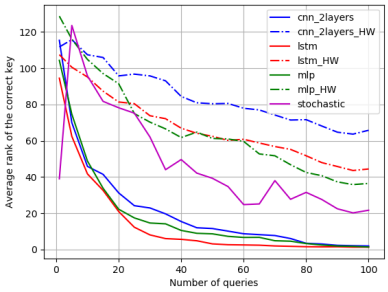
(a) Hamming weight leakage.



(b) Value leakage.



(c) First-order random leakage.



(d) Second-order random leakage.

Fig. 7: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when targeting a masked implementation.

– As already observed for the unprotected case, when the random leakage function is quadratic, the obtained results for the DL-SCA are much better compared to the scenario where the random leakage function is linear.
– Similarly, when the random leakage function is linear, the stochastic attack highly outperforms the DL-SCA. However, when the random leakage function is quadratic, the DL-SCA using the value labeling strategy outperform the stochastic attack and the DL-SCA based on the HW-DL architectures.

7.4 Takeaway Messages

From the previous experiment, one can conclude that the DL-SCA are not very sensitive to a leakage model deviation from the idealized models especially when the real leakage model function contains quadratic combinations of the bit coordinates of the sensitive data. Moreover, labeling the data according to the value of the sensitive variable is the best choice regardless of the leakage model of the targeted device.

8 DL-SCA Versus Shamir's Secret Sharing

8.1 Context and Experimental Set-up

In side-channel context, the most classical countermeasure consists in applying Boolean masking. That is, the sensitive variable Z is split into d shares a_i such that $Z = a_1 \oplus \cdots \oplus a_d$ to ensure a d^{th}-order security. The simplicity of the Boolean masking is an advantage from an implementation complexity point of view but, on the flip side, it helps the attacker: the information on the shared data is relatively easy to rebuild from the observed shares. Starting from this remark, Prouff and Roche in [35] and Goubin and Martinelli in [16] proposed independently to apply Shamir's Secret Sharing (SSS) instead of Boolean masking: the core principle of SSS is to split any sensitive variable Z into $n \geqslant 2d+1$ shares a_i which correspond to the evaluation, in n distinct non-zero public elements, of a random degree-d polynomial with constant term Z [39]. We denote this sharing by (n, d)-SSS in the sequel. The d^{th}-order security property comes as a direct consequence of the so-called *collusion resistance* of Shamir's sharing which essentially ensures that at least $d+1$ evaluations (*a.k.a.* shares a_i) must be involved to recover Z.

In this section, our goal is mainly to evaluate the resistance of SSS against DL-SCA. To perform this security assessment, we consider two experiment scenarios: the first one consists in generating simulated traces while for the second one we use real acquisitions captured on the ChipWhisperer (CW) platform [30]. The details of both experiments and the corresponding results are described in the following sections.

8.2 Simulation Set-up and Results

For our simulation set-up, we consider two implementations protected respectively with a $(3,1)$-SSS and a $(5,2)$-SSS. Both implementations leak the value of the manipulated data. Besides, the corresponding simulated traces $(\mathbf{T_i})_{1 \leq i \leq L}$, of S time samples each, are generated as follow:

$$\mathbf{T_i}[s] = \begin{cases} \text{select a random polymial } P_Z(X) = Z + \sum_{i=1}^{d} \mu_i X^i, & \\ P_Z(a_s) + \mathcal{N}(0,\sigma) & \text{if } s \text{ in } [1,n], \\ R + \mathcal{N}(0,\sigma) & \text{otherwise,} \end{cases}$$

where n is the number of shares (*i.e.* $n = 3$ for $(3,1)$-SSS and $n = 5$ $(5,2)$-SSS), a_s denotes the s^{th} public point, and $P_Z(X)$ is a random polynomial of degree d (with $d = 1$ for $(3,1)$-SSS and $d = 3$ for $(5,2)$-SSS) and whose coefficients μ_d and $\mu_{1 \leq i < d}$ are randomly generated in $[1, 255]$ and $[0, 255]$ respectively. For our simulation, the number of samples per trace is $S = 10$ and the noise standard deviation is $\sigma = 0.5$. For $(3,1)$-SSS, the used public points set is $\{86, 23, 115\}$ while for the $(5,2)$-SSS we select the set $\{86, 23, 115, 107, 189\}$ (denoted set #1 in the sequel). For both targeted implementations, we perform the following:

1. generate $L = 256,000$ traces for the profiling phase, $L = 1,000$ traces for the validation phase and $L = 2,000$ traces for the attack phase.
2. Train 3 different DL models: the CNN_2LAYERS, the MLP and the LSTM whose hyper-parameters are detailed in Tab. 2 in Appendix A.
3. Perform the key recovery and compute the average rank of the correct key.

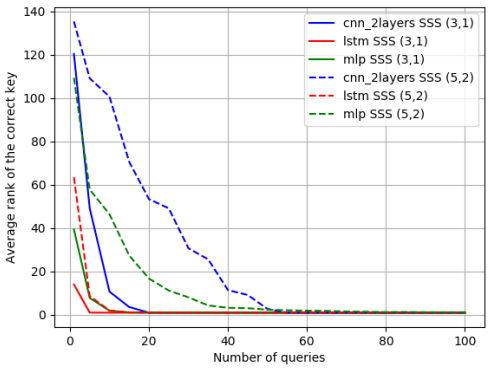


Fig. 8: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when targeting a $(3,1)$-SSS and a $(5,2)$-SSS.

The obtained attack results are depicted in Fig. 8. They demonstrate that DL-SCA are very efficient to break SSS implementation. More interestingly, the LSTM architecture outperforms the CNN and the MLP. This observation highlights, again, that the LSTM is an interesting neural network to consider in a side-channel evaluation especially when the sensitive data leakage is the combination of several shares leaking on different time samples of the traces (as it is the typical case of SSS). Finally, as expected the $(5,2)$-SSS offers a higher resistance against DL-SCA compared to the $(3,1)$-SSS. This is in-line with the rule: the higher the sharing (masking) order, the more secure the implementation is.

8.3 Practical Set-up and Results

To confirm the simulation results, we conduct some practical attacks on the ChipWhisperer platform. Mainly, we implement a $(5,2)$-SSS on an 8-bit AVR microprocessor ATxmega128d3 and we acquire power-consumption traces thanks to the ChipWhisperer-Lite (CW1173) basic board. We collect $256,000$ traces for the profiling phase, $1,000$ traces for the validation phase and $2,000$ traces for the attack phase. Then, we use the same DL architectures selected during our simulation to run the training and the attack phases. Moreover, and for the sake of comparison, we perform a Higher-Order Template Attack (HOTA) following

the procedure described in [10, 25]. The obtained average rank of the correct key for each attack is described in Fig. 9.

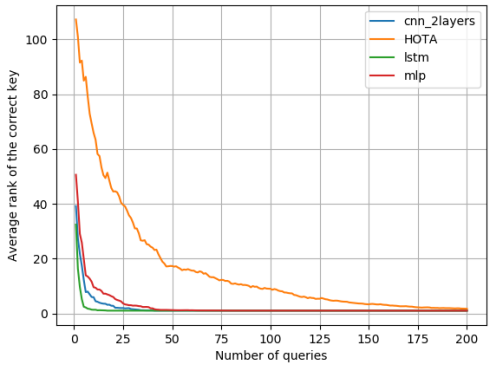


Fig. 9: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when targeting a $(5, 2)$-SSS implemented on the CW board.

As expected, the obtained results for DL-SCA with the real traces are in-line with those obtained with the simulation. Moreover, the DL-SCA outperform the HOTA. This observation is not surprising given that the DL-SCA are outperforming (or at least as efficient as) the first-order template attack when the Boolean masking is involved as protection against SCA.

8.4 Impact of the used Public Points Set on DL-efficiency

In [10], the authors exhibited a very interesting property regarding the effectiveness of the SSS scheme. Indeed, they demonstrated through several simulations and practical experiments (when performing the HOTA) that the choice of the public points in Shamir's secret sharing scheme has an impact on the countermeasure strength.

In this section, our goal is to validate this property, through simulations, when applying DL-SCA (rather than HOTA). To do so, we repeat the experience done for the $(5, 2)$-SSS in Sec. 8.2 when considering different sets of public points which are listed hereafter.

- set $\#2 = \{78, 104, 66, 56, 85\}$ and
- set $\#3 = \{125, 246, 119, 104, 150\}$.

The simulation results are plotted in Fig. 10. The obtained results with DL-TA confirm the property highlighted by the authors in [10] using HOTA. From Fig. 10, it is quite obvious that the choice of the public points in SSS plays a

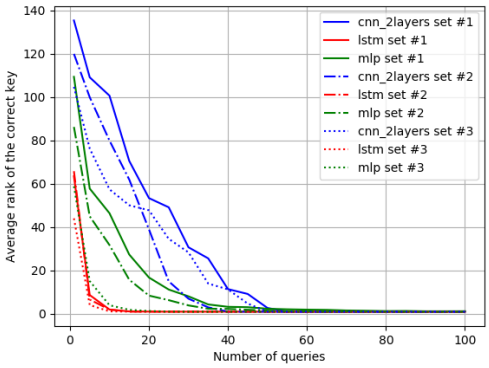


Fig. 10: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when considering three different sets of public points.

role in the security level (and the efficiency) of this countermeasure. It is worth to note also that the efficiency order of the different DL architectures used is invariant with respect to the set of public points used. That is, the LSTM is the best model followed by the MLP and the CNN_2LAYERS.

8.5 Takeaway Messages

The different simulations and practical experiments conducted in this section have demonstrated that the SSS is vulnerable to DL-SCA. From an adversary's perspective, the LSTM architecture in particular (and the time dependency neural networks in general) is very suitable to break an SSS implementation. This is due to its ability to discriminate and to predict data that depend on some shares leaking sequentially over time. Now, from a security developer's perspective, to strengthen the resistance of an SSS implementation, a strategy would consist in (1) choosing carefully the set of public points to use, (2) increasing the degree of the sharing, (3) adding some fake operations (*e.g* using a fake polynomial to evaluate the public points).

9 DL-SCA against Common SCA Countermeasures

9.1 DL-SCA against 1-amongst-N Countermeasure

The 1-amongst-N is commonly used to protect cryptographic implementations against SCA. The idea consists in executing the sensitive operation randomly amongst $(N-1)$ other fake computations. In this section, we assess the resistance of the countermeasure against DL-SCA through simulation and practical experiments with the CW board.

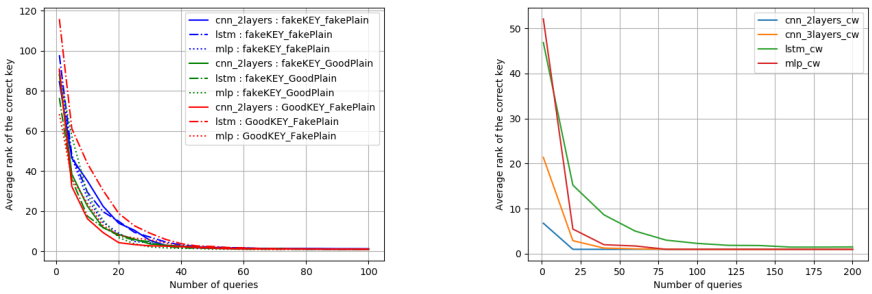
Simulation Set-up and Results To protect the Sbox operation using the 1-amongst-N countermeasure, several options are available. The Sbox input of the fake computations is:

- the XOR result of a fake key and a genuine plaintext,
- the XOR result of the genuine key and a fake plaintext, or
- the XOR result of a fake key and a fake plaintext.

In our simulations, we consider $N = 8$ (*i.e.* one genuine computation and 7 fake computations) and we target these several fashions of implementing the 1-amongst-N countermeasure to pinpoint which one better resists DL-SCA. The simulation protocol used to generate L traces $(\mathbf{T_i})_{1 \leq i \leq L}$, of $S = 8$ time samples each, is described hereafter:

$$\mathbf{T_i}[s] = \begin{cases} \text{select a random integer R in [1, 8]} \\ \text{for s in [1,8]:} \\ \quad Z + \mathcal{N}(0, \sigma) & \text{if } s = R, \\ \quad Z_{fake} + \mathcal{N}(0, \sigma) & \text{otherwise,} \end{cases}$$

where Z_{fake} denotes the fake computation generated using one of the options described above. To perform our attack, we generate $L = 200,000$ traces for the profiling phase, $L = 500$ traces for the validation phase and $L = 1,000$ traces for the attack phase. The used DL architectures are detailed in Tab. 2 in Appendix A. We provide in Fig. 11 (a) the obtained results. From this figure, one can conclude that the 1-amongst-N is vulnerable to DL-SCA. Indeed, few number traces are needed to break this countermeasure. Moreover, our results demonstrate that using fake keys and fake plaintexts for the dummy computations slightly improves the resistance of this countermeasure against DL-SCA.



(a) Simulation. (b) CW board.

Fig. 11: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when considering the 1-amongst-N countermeasure.

Another parameter which plays an import role in improving the security of this countermeasure is definitely the number of computations N. To assess the

impact of this parameter on the resistance of the 1-amongst-N countermeasure against DL-SCA, we repeat our simulation when increasing N (only for the case where fake keys and plaintexts are used to generate the fake computations). The results of this experiment are shown in Fig. 12.

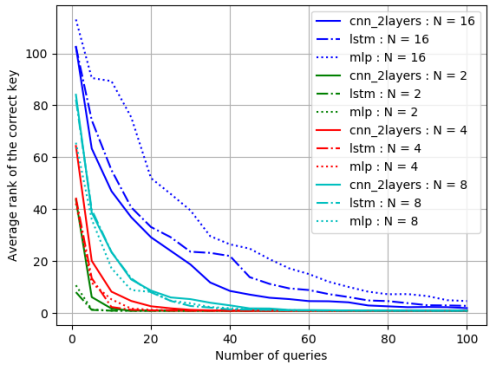


Fig. 12: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when increasing the number of fake computations.

As expected, increasing the number of dummy operations increases the resistance of the 1-amongst-N countermeasure against DL-SCA. Hence, the security designer has to choose the appropriate value of N that guarantees a good security performance trade-off. From an adversary's perspective, CNN seems to be the best architecture to choose when such countermeasure is involved to ensure protection against DL-SCA.

Experimental Results on CW To validate our simulation results in a real-world scenario, we implement the AES AddRoundkey and SubSytes operations of the first round protected with this 1-amongst-N countermeasure when $N = 4$. The source code of this implementation is available on GitHub [3] to ease the reproducibility of our results by the SCA community. Then, we acquire $128,000$ traces for the profiling, $1,000$ traces for the validation and $5,000$ traces for the attack phase. The Python script used for the acquisition is also available on GitHub [3]. The different DL architectures we considered for our evaluation on the CW board are denoted by the suffix "_CW" and are provided in Tab. 9 in Appendix A.

The practical results of our evaluation are plotted in Fig. 11 (b). These results are in-line with those obtained with simulation in the sense that the 1-amongst-N countermeasure is vulnerable to DL-SCA. Indeed, the different DL architectures achieve good training which results in a good key recovery phase (few traces are required to discriminate the good key value).

9.2 DL-SCA against Shuffling Countermeasure

Recently, authors in [44] have demonstrated that the shuffling countermeasure is vulnerable to DL-SCA. More interestingly, the authors have emphasized the interest of using CNN networks, compared to MLP ones, when the temporal position of the leakage varies (from one execution to another) due to the use of the shuffling countermeasure. Our purpose in this section is to assess if the LSTM architecture can outperform the CNN in such context. To do so, we perform some simulations and practical experiments on the CW board.

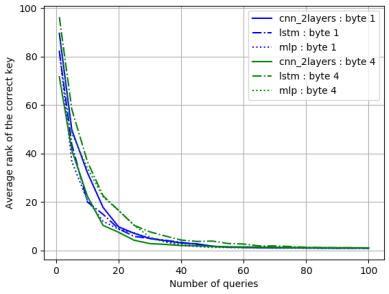
Simulation Set-up and Results In this section, we consider that the sensitive data is a vector of 8 AES Sbox outputs of the first round (denoted $\mathbf{Z}$). The simulation set-up used to generate L traces $(\mathbf{T_i})_{1 \leq i \leq L}$, of $S = 8$ time samples each, is described hereafter:

$$\mathbf{T_i}[s] = \begin{cases} \text{Sh\_t} = sh([1, \ldots, 8]) \\ \text{for s in [1,8]:} \\ \quad \mathbf{Z}[\text{Sh\_t}[s]] + \mathcal{N}(0, \sigma), \end{cases}$$

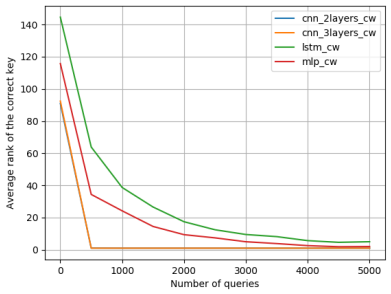
where sh is a permutation function, Sh_t is the shuffled table and $\mathbf{Z}[i]$ is the i^{th} AES Sbox output. By following this set-up, we generate $L = 200,000$ traces for the profiling phase, $L = 500$ traces for the validation phase and $L = 1,000$ traces for the attack phase. The used DL architectures are detailed in Tab. 2 in Appendix A. Moreover, for the attack phase, we target two Sbox outputs (the first and the forth one). The results of our investigations are shown in Fig. 13 (a). They confirm the results obtained in [44] in the way that the DL-SCA (regardless of the used DL architecture) defeat shuffling countermeasure. However, these obtained results do not allow us to conclude regarding the most appropriate DL model to use against this countermeasure. Indeed, the efficiency of the used DL models is quite similar. To address this question, we conduct a practical security evaluation of the shuffling countermeasure when implemented on the CW board.

Experimental Results on CW Regarding our practical setup, we acquire $128,000$ traces for the profiling, $1,000$ traces for the validation and $5,000$ traces for the attack phase. For this experiment, we only target the output of the first Sbox. The different hyper-parameters of the considered DL architectures are provided in Tab. 10 in Appendix A.

From the results plotted in Fig. 13 (b), one can see that it is more interesting to consider CNN networks when shuffling is involved as protection. Indeed, this observation is in-line with the conclusion drawn in [44]. Besides, the LSTM architecture is not performing well compared to the CNN as more traces are needed to recover the good key value. This could be explained by the fact that the LSTM method exploits the information of several time samples to classify the data. In the context of shuffling, the position of the relevant time samples (related to the leakage of the targeted byte) varies from one trace to another. We



(a) Simulation.



(b) CW board.

Fig. 13: Evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when considering the shuffling countermeasure.

believe that this behavior (*i.e.* effect of the shuffling countermeasure) decreases the effectiveness of the LSTM network to classify and discriminate the useful computation. We keep the study on how to improve the efficiency of the LSTM network in this context as future work.

10 DL-SCA on Combined Countermeasures

In a practical Common Criteria security evaluation [1], the assessed cryptographic implementations are often protected with a combination of several side-channel countermeasures. To the best of our knowledge, the only combination of countermeasures evaluated so far with respect to DL-SCA is masking with jitter [7, 36]. In this section, our goal is to evaluate the efficiency of DL-SCA on several combinations of side-channel countermeasures that are listed hereafter:

- shuffling and masking,
- shuffling and 1-amongst-N,
- masking and 1-amongst-N and,
- shuffling, masking and 1-amongst-N.

To do so, we present in the following our security evaluation based on both simulations and practical experiments on the CW board[8].

10.1 Simulation Set-up and Results

For our simulation, we consider that the sensitive data is a vector of 8 AES Sbox outputs of the first round (denoted $\mathbf{Z}$). The set-up to generate the simulated traces for each combination of countermeasures is provided in Tab. 1.

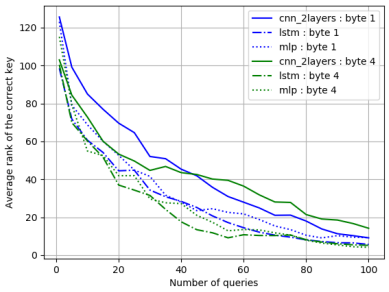
[8] For the 1-amongst-N countermeasure, we choose $N = 8$ and $N = 4$ for the simulations and the experiments on the CW board respectively.

Shuffling and masking	Shuffling and 1-amongst-N
$\mathbf{T_i}[s] = \begin{cases} \text{Sh\_t} = sh([1,\ldots,8]) \\ \text{for s in [1,8]:} \\ \quad \mathbf{M}[s] + \mathcal{N}(0,\sigma) \\ \text{for s in [9,16]:} \\ \quad \mathbf{Z}[\text{Sh\_t}[s]] \oplus \mathbf{M}[\text{Sh\_t}[s]] + \mathcal{N}(0,\sigma) \end{cases}$	$\mathbf{T_i}[s] = \begin{cases} \text{Sh\_t} = sh([1,\ldots,8]) \\ \text{for j in } [1,\ldots,8]: \\ \quad \text{choose R in } [(j-1)\cdot 8+1, j\cdot 8] \\ \quad \text{for s in } [(j-1)\cdot 8+1, j\cdot 8] \\ \quad\quad \text{choose } Z_{fake} \text{ in } [0,255] \\ \quad\quad \text{if s = R :} \\ \quad\quad\quad \mathbf{Z}[\text{Sh\_t}[j]] + \mathcal{N}(0,\sigma) \\ \quad\quad \text{else:} \\ \quad\quad\quad Z_{fake} + \mathcal{N}(0,\sigma) \end{cases}$
Masking and 1-amongst-N	Shuffling, masking and 1-amongst-N
$\mathbf{T_i}[s] = \begin{cases} \text{for j in } [1,\ldots,8]: \\ \quad \text{choose R in } [(j-1)\cdot 8+1, j\cdot 8] \\ \quad \text{for s in } [(j-1)\cdot 8+1, j\cdot 8] \\ \quad\quad \text{choose } Z_{fake} \text{ in } [0,255] \\ \quad\quad \text{if s = R :} \\ \quad\quad\quad \mathbf{Z}[j] + \mathcal{N}(0,\sigma) \\ \quad\quad \text{else:} \\ \quad\quad\quad Z_{fake} + \mathcal{N}(0,\sigma) \end{cases}$	$\mathbf{T_i}[s] = \begin{cases} \text{Sh\_t} = sh([1,\ldots,8]) \\ \text{for s in [1,8]:} \\ \quad \mathbf{M}[s] + \mathcal{N}(0,\sigma) \\ \text{for j in } [1,\ldots,8]: \\ \quad \text{choose R in } [(j-1)\cdot 8+1, j\cdot 8] \\ \quad \text{for s in } [(j\cdot 8+1, (j+1)\cdot 8] \\ \quad\quad \text{choose } Z_{fake} \text{ in } [0,255] \\ \quad\quad \text{if s = R :} \\ \quad\quad\quad \mathbf{Z}[\text{Sh\_t}[j]] \oplus \mathbf{M}[\text{Sh\_t}[j]] + \mathcal{N}(0,\sigma) \\ \quad\quad \text{else:} \\ \quad\quad\quad Z_{fake} + \mathcal{N}(0,\sigma) \end{cases}$

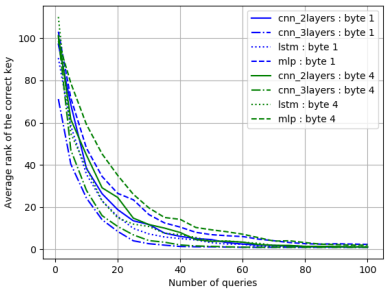
Table 1: Simulation set-up to generate the traces of several combinations of side-channel countermeasures.

For each targeted implementation, we generate $200,000$ traces for the profiling phase, 500 traces for the validation phase and $1,000$ traces for the attack phase. As usual, for our evaluation, we target 3 different DL models whose hyperparameters are listed in Tab. 2 in Appendix A and two key bytes (the first and the forth ones).

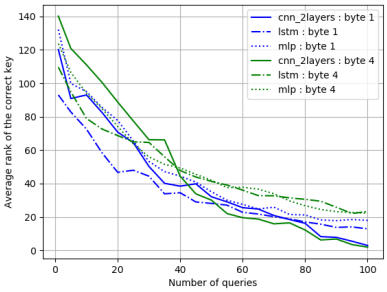
The results of our simulations are provided in Fig. 14. The obtained results demonstrate that the DL-SCA are efficient even when several side-channel countermeasures are combined together to strengthen the resistance of the implementation. Moreover, it is obvious from Fig. 14 that when masking is combined with other countermeasures the corresponding implementation offers a better resistance against DL-SCA.



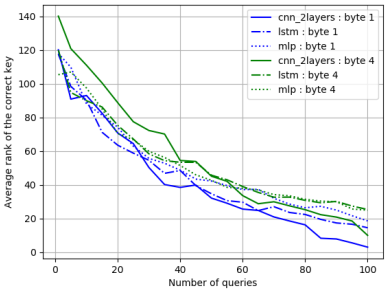
(a) Shuffling and masking.



(b) Shuffling and 1-amongst-N.



(c) Masking and 1-amongst-N.

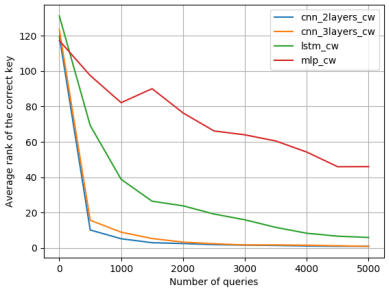


(d) Shuffling, masking and 1-amongst-N.

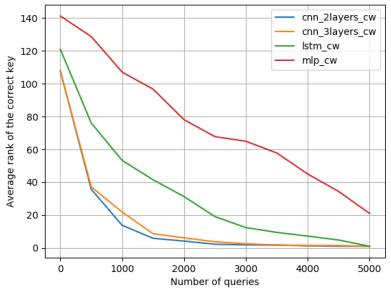
Fig. 14: Simulation results: evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when combining several countermeasures.

10.2 Experimental Results on CW Board

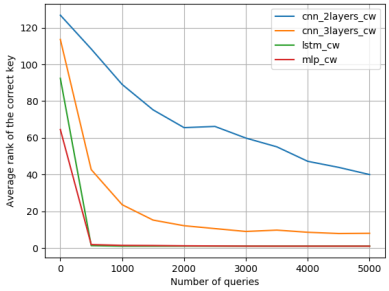
To assess the efficiency of DL-SCA in defeating a combination of countermeasures in a real-world context scenario, we evaluate the implementation of several combinations on the CW board. We kindly recall the reader that these implementations, as well as the Python scripts used for the acquisition, are publicaly available on GitHub [3] to ease the reproducibility of our results. Moreover, the different hyper-parameters of the considered DL architectures are provided in Tab. 11 (for shuffling and masking), Tab. 12 (for shuffling and 1-amongst-N), Tab. 13 (masking and 1-amongst-N) and Tab. 14 (for shuffling, masking and 1-amongst-N) in Appendix A. Regarding the combinations involving masking, we perform a leakage assessment by applying the *t-test* [15] to validate that no first-order leakage of the sensitive data occurs.



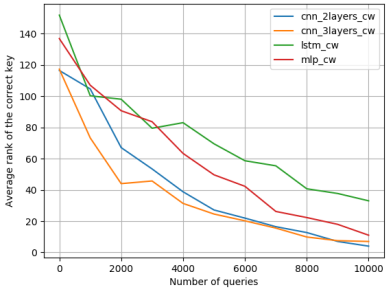
(a) Shuffling and masking.



(b) Shuffling and 1-amongst-N.



(c) Masking and 1-amongst-N.



(d) Shuffling, masking and 1-amongst-N.

Fig. 15: Experimental results: evolution of the correct key rank (y-axis) according to an increasing number of traces (x-axis) when combining several countermeasures.

The outcomes of the experiment are plotted in Fig. 15. From this figure, the following observations could be emphasized:

- the practical results on the CW board confirm the simulation ones. Indeed, the DL-SCA are efficient even when a combination of side-channel countermeasures is involved to ensure protection.
- When shuffling is implemented, the CNN seems to be more efficient than the MLP and LSTM ones. This result is in-line with the one obtained in Sec. 9.2 where only shuffling countermeasure is used to protect the implementation.
- Similarly, when no shuffling is implemented, the LSTM and the MLP are more interesting than the CNN.

10.3 Takeaway Messages

From an adversary's perspective, we conclude that when the leakage is time-invariant, the MLP and LSTM are the best models to consider. In the flip side, when shuffling or jitter-based countermeasures are involved [7, 36], the CNN networks are of great interest. Finally, we emphasize the fact that our goal was not to find the optimal DL architectures when considering these combinations of countermeasures but was more to demonstrate that DL-SCA are efficient in such a context. Moreover, we recall that the reported DL architectures in this work are not a "one-size-fits-all" modeling. In fact, the DL architecture should be defined with respect to the characteristics of the targeted data-set.

From a security designer's perspective, we believe that increasing the number of fake operations (for the scheme that combines the three studied countermeasures) along with inserting some random delays (and/or activating jitter) would strengthen the resistance of a cryptographic implementation with respect to DL-SCA.

11 Conclusion

In this paper, we assessed the efficiency of DL-SCA under different realistic and practical scenarios. First, we studied the impact of the intrinsic characteristics of the manipulated data-set (*i.e.* distance in time samples between the PoI, the dimensionality of the area of interest and the pre-processing of the data) on the effectiveness of DL-SCA to recover the key of a cryptographic implementation. We argue through several simulations and practical experimentation that DL-SCA are sensitive to these characteristics and security analysts have to carefully design their DL architectures to maximize their efficiency. Second, we evaluated the tolerance of DL-SCA with respect to the leakage model function and the addition of artificial noise. Our results have proven that the DL-SCA are not sensitive to a variation of these parameters and more interestingly this variation can result in a more efficient attack. Then, we assessed the SSS countermeasure against DL-SCA. As expected, this side-channel protection is vulnerable to DL-SCA. Finally, we studied the effect of combining several countermeasures to

resist these profiling attack. Our investigations have demonstrated that DL-SCA are still efficient to break them. For all the performed experiments, we provided some recommendations and practical hints to either enhance the efficiency of the DL-SCA (from an adversary's perspective) or to strengthen the resistance of the cryptographic implementations against these attacks (from a security developer's perspective).

As a future work, we plan to assess the DL-SCA efficiency against the targeted combinations of side-channel countermeasures when implemented on modern CPUs.

References

1. Common Criteria Portal. https://www.commoncriteriaportal.org/.
2. Keras Library. https://keras.io/.
3. Source code of the implementations of the countermeasures we assessed on the CW board. https://github.com/DLCW/Coutermeasur_Implementation_on_CW_Board.
4. ANSSI. ASCAD database. https://github.com/ANSSI-FR/ASCAD.
5. Christopher M. Bishop. *Neural Networks for Pattern Recognition*. Oxford University Press, Inc., New York, NY, USA, 1995.
6. Éric Brier, Christophe Clavier, and Francis Olivier. Correlation Power Analysis with a Leakage Model. In *CHES*, volume 3156 of *LNCS*, pages 16–29. Springer, August 11–13 2004. Cambridge, MA, USA.
7. Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529 of *Lecture Notes in Computer Science*, pages 45–68. Springer, 2017.
8. Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Kernel discriminant analysis for information extraction in the presence of masking. In Kerstin Lemke-Rust and Michael Tunstall, editors, *Smart Card Research and Advanced Applications*, pages 1–22, Cham, 2017. Springer International Publishing.
9. Mathieu Carbone, Vincent Conin, Marie-Angela Cornelie, François Dassance, Guillaume Dufresne, Cécile Dumas, Emmanuel Prouff, and Alexandre Venelli. Deep learning to evaluate secure RSA implementations. *IACR Cryptology ePrint Archive*, 2019:54, 2019. To appear in the proceedings of TCHES 2019.
10. Herv Chabanne, Houssem Maghrebi, and Emmanuel Prouff. Linear repairing codes and side-channel attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2018(1):118–141, Feb. 2018.
11. Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2002. San Francisco Bay (Redwood City), USA.
12. Omar Choudary and Markus G. Kuhn. Efficient Template Attacks. Cryptology ePrint Archive, Report 2013/770, 2013. http://eprint.iacr.org/2013/770.
13. Guillaume Dabosville, Julien Doget, and Emmanuel Prouff. A New Second Order Side Channel Attack Based on Linear Regression. *IEEE Trans. Computers*, 2012. (*In press*).

14. Julien Doget, Emmanuel Prouff, Matthieu Rivain, and François-Xavier Standaert. Univariate side channel attacks and leakage modeling. *J. Cryptographic Engineering*, 1(2):123–144, 2011.
15. Gilbert Goodwill, Benjamin Jun, Josh Jaffe, and Pankaj Rohatgi. A testing methodology for side-channel resistance validation, September 2011. NIST Non-Invasive Attack Testing Workshop.
16. Louis Goubin and Ange Martinelli. Protecting AES with Shamir's Secret Sharing Scheme. In Preneel and Takagi [33], pages 79–94.
17. Isabelle Guyon. A scaling law for the validation-set training-set size ratio. In *AT & T Bell Laboratories*, 1997.
18. Michiel Hermans and Benjamin Schrauwen. Training and analysing deep recurrent neural networks. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 190–198. Curran Associates, Inc., 2013.
19. Benjamin Hettwer, Stefan Gehrer, and Tim Gneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. Cryptology ePrint Archive, Report 2019/143, 2019. https://eprint.iacr.org/2019/143.
20. Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.
21. Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.
22. Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise: Unleashing the power of convolutional neural networks for profiled side-channel analysis. Cryptology ePrint Archive, Report 2018/1023, 2018. https://eprint.iacr.org/2018/1023.
23. Yann LeCun and Yoshua Bengio. The handbook of brain theory and neural networks. chapter Convolutional Networks for Images, Speech, and Time Series, pages 255–258. MIT Press, Cambridge, MA, USA, 1998.
24. Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In Stefan Mangard and Axel Y. Poschmann, editors, *Constructive Side-Channel Analysis and Secure Design - 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, volume 9064 of *Lecture Notes in Computer Science*, pages 20–33. Springer, 2015.
25. Victor Lomné, Emmanuel Prouff, Matthieu Rivain, Thomas Roche, and Adrian Thillard. *How to Estimate the Success Rate of Higher-Order Side-ChannelAttacks*, pages 35–54. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014.
26. Victor Lomné, Emmanuel Prouff, and Thomas Roche. Behind the scene of side channel attacks. In Kazue Sako and Palash Sarkar, editors, *Advances in Cryptology - ASIACRYPT 2013 - 19th International Conference on the Theory and Application of Cryptology and Information Security, Bengaluru, India, December 1-5, 2013, Proceedings, Part I*, volume 8269 of *Lecture Notes in Computer Science*, pages 506–525. Springer, 2013.
27. Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, pages 3–26, 2016.

28. Jonathan Masci, Ueli Meier, Dan Cireşan, and Jürgen Schmidhuber. Stacked convolutional auto-encoders for hierarchical feature extraction. In *Proceedings of the 21th International Conference on Artificial Neural Networks - Volume Part I*, ICANN'11, pages 52–59, Berlin, Heidelberg, 2011. Springer-Verlag.
29. Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. *IACR Cryptology ePrint Archive*, 2018:1196, 2018. To appear in the proceedings of COSADE 2019.
30. Colin O'Flynn and Zhizhang (David) Chen. Chipwhisperer: An open-source platform for hardware embedded security research. Cryptology ePrint Archive, Report 2014/204, 2014. http://eprint.iacr.org/2014/204.
31. Keiron O'Shea and Ryan Nash. An introduction to convolutional neural networks. *CoRR*, abs/1511.08458, 2015.
32. Éric Peeters, François-Xavier Standaert, Nicolas Donckers, and Jean-Jacques Quisquater. Improved Higher-Order Side-Channel Attacks with FPGA Experiments. In *CHES*, volume 3659 of *LNCS*, pages 309–323. Springer, 2005.
33. Bart Preneel and Tsuyoshi Takagi, editors. *Cryptographic Hardware and Embedded Systems - CHES 2011 - 13th International Workshop, Nara, Japan, September 28 – October 1, 2011. Proceedings*, volume 6917 of *LNCS*. Springer, 2011.
34. Emmanuel Prouff, Matthieu Rivain, and Régis Bevan. Statistical Analysis of Second Order Differential Power Analysis. *IEEE Trans. Computers*, 58(6):799–811, 2009.
35. Emmanuel Prouff and Thomas Roche. Higher-Order Glitches Free Implementation of the AES Using Secure Multi-party Computation Protocols. In Preneel and Takagi [33], pages 63–78.
36. Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.
37. Werner Schindler. Advanced stochastic methods in side channel analysis on block ciphers in the presence of masking. *Journal of Mathematical Cryptology*, 2(3):291–310, October 2008. ISSN (Online) 1862-2984, ISSN (Print) 1862-2976, DOI: 10.1515/JMC.2008.013.
38. Werner Schindler, Kerstin Lemke, and Christof Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In LNCS, editor, *CHES*, volume 3659 of *LNCS*, pages 30–46. Springer, Sept 2005. Edinburgh, Scotland, UK.
39. Adi Shamir. How to Share a Secret. *Communications of the ACM*, 22(11):612–613, November 1979.
40. François-Xavier Standaert, Benedikt Gierlichs, and Ingrid Verbauwhede. Partition vs. Comparison Side-Channel Distinguishers: An Empirical Evaluation of Statistical Tests for Univariate Side-Channel Attacks against Two Unprotected CMOS Devices. In *ICISC*, volume 5461 of *LNCS*, pages 253–267. Springer, December 3-5 2008. Seoul, Korea.
41. François-Xavier Standaert, Tal Malkin, and Moti Yung. A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In *EUROCRYPT*, volume 5479 of *LNCS*, pages 443–461. Springer, April 26-30 2009. Cologne, Germany.
42. Benjamin Timon. Non-profiled deep learning-based side-channel attacks. *IACR Cryptology ePrint Archive*, 2018:196, 2018. To appear in the proceedings of TCHES 2019.
43. Alice Zheng and Amanda Casari. *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists*. O'Reilly Media, Inc., 1st edition, 2018.

44. Yevhenii ZOTKIN, Francis OLIVIER, and Eric BOURBAO. Deep learning vs template attacks in front of fundamental targets: experimental study. Cryptology ePrint Archive, Report 2018/1213, 2018. https://eprint.iacr.org/2018/1213.

A Hyper-parameters of the used DL architectures

CNN_2LAYERS
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(256, activation="softmax")
CNN_3LAYERS
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 32, padding='same', input_shape=(nb_samples,1), activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 16, padding='same', activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(256, activation="softmax")
MLP
nb_epoch = 100 batch_size_training = 128 Dense(20, activation="relu", input_shape=(nb_samples,)) Dense(50, activation="relu") Dense(256, activation="softmax")
LSTM
nb_epoch = 100 batch_size_training = 128 LSTM(26, input_shape=(nb_samples,1), return_sequences=True) LSTM(26) Dense(256, activation='softmax')

Table 2: Hyper-parameters of the basic DL architectures used in this work.

CNN_3LAYERS
nb_epoch = 50 batch_size_training = 128 Convolution1D(8, 32, padding='same', input_shape=(nb_samples,1), activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 16, padding='same', activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(50, activation="relu") Dense(256, activation="softmax")
MLP
nb_epoch = 50 batch_size_training = 128 Dense(20, activation="relu", input_shape=(nb_samples,)) Dense(50, activation="relu") Dense(256, activation="softmax")
LSTM
nb_epoch = 50 batch_size_training = 128 LSTM(26, input_shape=(nb_samples,1), return_sequences=True) LSTM(26) Dense(256, activation='softmax')

Table 3: Hyper-parameters of the used DL architectures in Sec. 5.

CNN_2LAYERS_BIS
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 32, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(256, activation="softmax")
MLP_BIS
nb_epoch = 100 batch_size_training = 128 Dense(20, activation="relu", input_shape=(nb_samples,)) Dense(50, activation="relu") Dense(256, activation="softmax")
LSTM_BIS
nb_epoch = 100 batch_size_training = 128 LSTM(60, input_shape=(nb_samples,1), return_sequences=True) LSTM(60) Dense(256, activation='softmax')

Table 4: Hyper-parameters of the enhanced DL architectures in Sec. 3 and Sec. 4.

CNN_2LAYERS
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") GaussianNoise(stddev) Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(256, activation="softmax")
CNN_3LAYERS
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 32, padding='same', input_shape=(nb_samples,1), activation="relu") GaussianNoise(stddev) MaxPooling1D(pool_size=3) Convolution1D(8, 16, padding='same', activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(256, activation="softmax")
MLP
nb_epoch = 100 batch_size_training = 128 Dense(20, activation="relu", input_shape=(nb_samples,)) GaussianNoise(stddev) Dense(50, activation="relu") Dense(256, activation="softmax")
LSTM
nb_epoch = 100 batch_size_training = 128 LSTM(26, input_shape=(nb_samples,1), return_sequences=True) GaussianNoise(stddev) LSTM(26) Dense(256, activation='softmax')

Table 5: Hyper-parameters of the used DL architectures in Sec. 6.

CNN_2LAYERS_W_NOISE
nb_epoch = 50 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(20, activation="relu") Dense(256, activation="softmax")
CNN_3LAYERS_W_NOISE
nb_epoch = 50 batch_size_training = 128 Convolution1D(8, 32, padding='same', input_shape=(nb_samples,1), activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 16, padding='same', activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(20, activation="relu") Dense(256, activation="softmax")
MLP_W_NOISE
nb_epoch = 50 batch_size_training = 128 Dense(10, activation="relu", input_shape=(nb_samples,)) Dense(5, activation="relu") Dense(256, activation="softmax")
LSTM_W_NOISE
nb_epoch = 50 batch_size_training = 128 LSTM(12, input_shape=(nb_samples,1), return_sequences=True) LSTM(5) Dense(256, activation='softmax')

Table 6: Hyper-parameters of the DL architectures in Sec. 6 (without adding the artificial noise layer).

CNN_2LAYERS_W_NOISE
nb_epoch = 50 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(20, activation="relu") GaussianNoise(stddev=0.2) Dense(256, activation="softmax")
CNN_3LAYERS_W_NOISE
nb_epoch = 50 batch_size_training = 128 Convolution1D(8, 32, padding='same', input_shape=(nb_samples,1), activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 16, padding='same', activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(20, activation="relu") GaussianNoise(stddev=0.2) Dense(256, activation="softmax")
MLP_W_NOISE
nb_epoch = 50 batch_size_training = 128 Dense(10, activation="relu", input_shape=(nb_samples,)) GaussianNoise(stddev=0.3) Dense(5, activation="relu") Dense(256, activation="softmax")
LSTM_W_NOISE
nb_epoch = 50 batch_size_training = 128 LSTM(12, input_shape=(nb_samples,1), return_sequences=True) GaussianNoise(stddev=0.25) LSTM(5) GaussianNoise(stddev=0.25) Dense(256, activation='softmax')

Table 7: Hyper-parameters of the enhanced DL architectures in Sec. 6 (with adding the artificial noise layer).

CNN_2LAYERS_HW
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(9, activation="softmax")
CNN_3LAYERS_HW
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 32, padding='same', input_shape=(nb_samples,1), activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 16, padding='same', activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(9, activation="softmax")
MLP_HW
nb_epoch = 100 batch_size_training = 128 Dense(20, activation="relu", input_shape=(nb_samples,)) Dense(50, activation="relu") Dense(9, activation="softmax")
LSTM_HW
nb_epoch = 100 batch_size_training = 128 LSTM(26, input_shape=(nb_samples,1), return_sequences=True) LSTM(26) Dense(9, activation='softmax')

Table 8: Hyper-parameters of the used DL architectures in Sec. 7.

CNN_2LAYERS_CW
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(256, activation="softmax")
CNN_3LAYERS_CW
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 4, padding='same', activation="relu") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(256, activation="softmax")
MLP_CW
nb_epoch = 100 batch_size_training = 128 node=nb_samples layer_nb=5 Dense(node, input_dim=nb_samples, activation='relu') **for** i **in range**(layer_nb−2): Dense(node, activation='relu') BatchNormalization() Dense(256, activation='softmax')
LSTM_CW
nb_epoch = 100 batch_size_training = 128 LSTM(nb_samples, input_shape=(nb_samples,1), return_sequences=True) BatchNormalization() LSTM(100) Dense(256, activation='softmax')

Table 9: Hyper-parameters of the used DL architectures in Sec. 9.1.

CNN_2LAYERS_CW
nb_epoch = 200 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(256, activation="softmax")
CNN_3LAYERS_CW
nb_epoch = 200 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 4, padding='same', activation="relu") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(256, activation="softmax")
MLP_CW

nb_epoch = 200
batch_size_training = 128
node=nb_samples
layer_nb=4
Dense(node, input_dim=nb_samples, activation='relu')
for i **in range**(layer_nb−2):
  Dense(node, activation='relu')
  BatchNormalization()
Dense(256, activation='softmax')

LSTM_CW
nb_epoch = 100 batch_size_training = 128 LSTM(nb_samples, input_shape=(nb_samples,1), return_sequences=True) BatchNormalization() LSTM(nb_samples, return_sequences=True) LSTM(50) Dense(256, activation='softmax')

Table 10: Hyper-parameters of the used DL architectures in Sec. 9.2.

CNN_2LAYERS_CW
nb_epoch = 20 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(256, activation="softmax")
CNN_3LAYERS_CW
nb_epoch = 20 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 4, padding='same', activation="relu") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(256, activation="softmax")
MLP_CW

nb_epoch = 20
batch_size_training = 128
node=nb_samples
layer_nb=4
Dense(node, input_dim=nb_samples, activation='relu')
for i **in range**(layer_nb−2):
  Dense(node, activation='relu')
  BatchNormalization()
Dense(256, activation='softmax')

LSTM_CW
nb_epoch = 20 batch_size_training = 128 LSTM(nb_samples, input_shape=(nb_samples,1), return_sequences=True) BatchNormalization() LSTM(nb_samples, return_sequences=True) LSTM(400) Dense(256, activation='softmax')

Table 11: Hyper-parameters of the used DL architectures in Sec. 10 (for evaluating shuffling and masking countermeasures on CW board).

CNN_2LAYERS_CW
nb_epoch = 500 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(256, activation="softmax")
CNN_3LAYERS_CW
nb_epoch = 500 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 4, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(256, activation="softmax")
MLP_CW

nb_epoch = 500
batch_size_training = 128
node=nb_samples
layer_nb=5
Dense(node, input_dim=nb_samples, activation='relu')
for i **in range**(layer_nb−2):
  Dense(node, activation='relu')
  Dropout(0.01)
  BatchNormalization()
Dense(256, activation='softmax')

LSTM_CW
nb_epoch = 500 batch_size_training = 128 LSTM(nb_samples, input_shape=(nb_samples,1), return_sequences=True) BatchNormalization() LSTM(200) Dense(256, activation='softmax')

Table 12: Hyper-parameters of the used DL architectures in Sec. 10 (for evaluating shuffling and 1-amongst-N countermeasures on CW board).

CNN_2LAYERS_CW
nb_epoch = 50 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.4) Dense(256, activation="softmax")
CNN_3LAYERS_CW
nb_epoch = 50 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.2) MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="relu") Dropout(0.2) MaxPooling1D(pool_size=2) Convolution1D(8, 4, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dropout(0.1) Dense(256, activation="softmax")
MLP_CW

nb_epoch = 50
batch_size_training = 128
node=5000
layer_nb=5
Dense(node, input_dim=5000, activation='relu')
for i **in range**(layer_nb−2):
  Dense(node, activation='relu')
  Dropout(0.01)
  BatchNormalization()
Dense(256, activation='softmax')

LSTM_CW
nb_epoch = 100 batch_size_training = 128 LSTM(nb_samples, input_shape=(200,1), return_sequences=True) BatchNormalization() LSTM(200, return_sequences=True) BatchNormalization() LSTM(100) Dense(256, activation='softmax')

Table 13: Hyper-parameters of the used DL architectures in Sec. 10 (for evaluating masking and 1-amongst-N countermeasures on CW board).

CNN_2LAYERS_CW
nb_epoch = 50 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") Dropout(0.002) BatchNormalization() MaxPooling1D(pool_size=2) Convolution1D(8, 8, padding='same', activation="tanh") Flatten() Dropout(0.01) Dense(256, activation="softmax")
CNN_3LAYERS_CW
nb_epoch = 100 batch_size_training = 128 Convolution1D(8, 16, padding='same', input_shape=(nb_samples,1), activation="relu") MaxPooling1D(pool_size=3) Convolution1D(8, 8, padding='same', activation="relu") MaxPooling1D(pool_size=2) Convolution1D(8, 2, padding='same', activation="tanh") MaxPooling1D(pool_size=2) Flatten() Dense(256, activation="softmax")
MLP_CW

nb_epoch = 50
batch_size_training = 128
node=5000
layer_nb=6
Dense(node, input_dim=5000, activation='relu')
for i **in range**(layer_nb−2):
  Dense(node, activation='relu')
  Dropout(0.001)
  BatchNormalization()
Dense(256, activation='softmax')

LSTM_CW
nb_epoch = 50 batch_size_training = 128 LSTM(2000, input_shape=(1,nb_samples), return_sequences=True) BatchNormalization() LSTM(2000, return_sequences=True) BatchNormalization() LSTM(2000) Dense(256, activation='softmax')

Table 14: Hyper-parameters of the used DL architectures in Sec. 10 (for evaluating shuffling, masking and 1-amongst-N countermeasures on CW board).