A Comparison of Deep Learning Approaches for Power-based Side-channel Attacks

Roberto Capoferri$^{1[0009-0005-4535-2186]}$, Alessandro Barenghi$^{1[0000-0003-0840-6358]}$, Luca Breveglieri$^{1[0000-0001-5294-6840]}$, Niccolò Izzo$^{2[0000-0003-4966-1339]}$, and Gerardo Pelosi$^{1[0000-0002-3812-5429]}$

[1] Politecnico di Milano, 20133 Milano, Italy
`name.surname@polimi.it`
[2] Micron Semiconductor Italia Srl, 20871 Vimercate (MB), Italy
`niccoloizzo@micron.com`

Abstract. Side-channel attacks aim to recover cryptographic secrets by exploiting involuntary information channels coming from the computational platform of the targeted cipher implementation, e.g., power consumption or electromagnetic emissions. Among them deep learning based attacks have recently obtained great attention from both industry and academia, due to their greater efficiency and accuracy with respect to other methodologies. We provide a systematic comparison of the effectiveness of deep learning based attacks by considering different data acquisition and training methods. We also tackle the problem of the portability of the derived information leakage model, by analysing multiple instances of the same device, an ARM Cortex-M4 32-bit processor, running a software implementation of the AES-128 cipher. We complement the exploration of the attack space by considering datasets corresponding to cipher executions employing, for each run, either the same fixed secret key or a randomly chosen one. Furthermore, we generalize the set of inputs considered to build the model, by adding also the plaintexts fed to each cipher run. Finally, from the perspective of efficiency, we point out several unexpected and counterintuitive benchmark points.

Keywords: Side-channel Analysis · Deep Learning · Applied Cryptography

1 Introduction

Side-channel Attacks (SCAs) are a very relevant threat to the security of computing devices that offer cryptographic functions. There are various examples of attacks that target real-world devices, such as smartphones [21] or secure elements [36]. These kinds of attack require a physical access to the device, for measuring information leakage, which is a realistic scenario with the diffusion of mobile and IoT devices. Deep learning (DL) based SCAs against cryptographic implementations have become an active subject of research in recent years, resulting in similar if not better performance with respect to other types of profiled attacks, such as the template attacks [22].

Attacks based on the analysis of the profile of both the dynamic power consumption or the electromagnetic radiation observed during the execution of a cryptographic algorithm (either in hardware or in software) define an important category of side-channel attacks. For each input provided to the cryptographic implementation at hand, assumed to be known in any possible detail (except for the value of the secret key), the numerical time series, a.k.a. *trace*, resulting from the measurement of one of the aforementioned parameters is recorded. The *divide-et-impera* observation at the core of the attack assumes that the secret key is composed as a long binary string that is processed, by the underlying computational platform, one (small) fragment at a time. Since the operation computed by employing the key fragment is known, it is easy to derive the switching activity (toggle count) resulting from its computation, by guessing the value of the key fragment and knowing the input fed into the algorithm. Repeating such a guessing for each possible value of the key fragment allows the attacker to derive a "model", which in turn can be correlated with the actual switching activity measured out from the execution of the target cryptographic implementation, for different inputs to the algorithm.

Over the years, numerous techniques have been designed to successfully execute a (secret-)key recovery attack. Many of these techniques, e.g., [13,9,20], make use of statistical tools to infer the value of the secret key, starting from both the model of a proper operation executed by the cryptographic algorithm and a set of traces. Other techniques, e.g., [11], assume the availability of multiple instances of the same computational platforms, therefore many analyses can be performed on just one of them to the end of quickly derive the secret-key employed in a device deployed on field. In recent years, the replacement of the statistical tools applied to execute a power-based SCA with deep-learning techniques has become more and more common, with interesting improvements in terms of effectiveness, robustness and performance [28].

Contributions. In this work, we provide a systematic comparison of the effectiveness of DL SCAs by considering different data acquisition and training methods. In particular, we explore the use of different datasets composed by traces collected from three distinct instances of the device under test, which is based on an ARM Cortex-M4 32-bit processor, running a software implementation of the AES-128 cipher. Traces are also distinguished (especially when training DL models) by whether they were measured from runs of the target cryptographic implementation in which the secret key was kept either fixed, or chosen randomly each time. The use of different devices is important to determine the degree of *portability* of the conclusions that can be drawn about the SCA vulnerability of a particular device instance, to the whole family of devices. In fact, common DL operations such as *tuning* and *training* are performed on device instances other than those that are being attacked. At the best of our knowledge, this is the first assessment of the application of DL SCAs techniques and their portability that addresses a (quite complex) 32-bit microcontroller as target computational platform. From the perspective of efficiency, we also point out several interesting (not expected and not intuitive) benchmark points during the exploration of the attack space generated by DL analyses. Finally, the source code [35] developed to perform our analysis, as well as the employed datasets [33,34], are made available for reproducibility.

Paper Organization. The rest of the paper is organized as follows. In Sect. 2 we present some background notions regarding deep learning techniques and side-channel analysis. In Sect. 3 and its subsections we describe the methodology defined for the exploration of the attack space, i.e., data collection, model selection and training. The results are described in Sect. 4, followed by a discussion of related work in Sect. 5. Finally, Sect. 6 reports our conclusions.

2 Background

In the following, we briefly recall the relevant notions about neural networks (NNs), which underlie every DL technique, and the main concepts regarding power-based SCAs, detailing how NNs are used in profiled side-channel attacks.

Deep Learning. It is a subset of machine learning that uses multi-layered NNs, to simulate a complex decision-making power. While there are several ways of building a neural network, in this work we consider the Multi-Layer Perceptron (MLP) [23,15] for its ability to learn nonlinear relationships among data and well modeled tasks such as classification, regression, and pattern recognition. An MLP is the extension of a single perceptron, i.e., the unit able to linearly combine its inputs and fed an evaluation function, by introducing multiple layers of fully connected neurons (each with a nonlinear kind of activation function [37]), and is organized in a feed-forward manner. As shown in Fig. 1, the structure of an MLP includes:

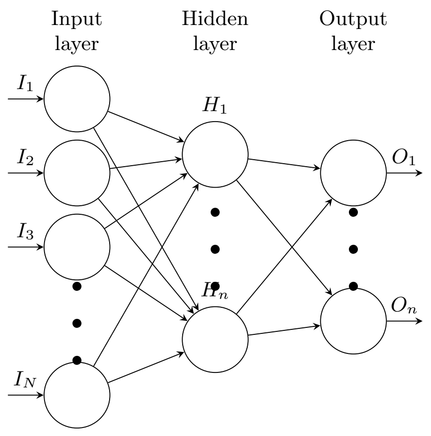


Fig. 1. Structure of an MLP

- An *input layer* $\mathbf{x} = [I_1, I_2, \dots, I_N]$, where each perceptron takes as input an input data feature I_j.
- $\ell \geqslant 1$ *hidden layer*s. In each of these layers, each perceptron out of ℓ_k, $1 < k \leqslant \ell$, takes an input from each perceptron in layer $k-1$ and forwards its output to each perceptron in layer k. Any value moved from the i-th perceptron at layer $k-1$ to the j-th perceptron at layer $k-1$, is multiplied by a weight coefficient $w_{j,i}$, prior to be part of a summation computed by perception k, which is in turn fed into its activation function [37] to provide its output.

$$\mathbf{W}_1 = \begin{pmatrix} w_{1,1} & w_{1,2} & \cdots & w_{1,\ell_1} \\ w_{2,1} & w_{2,2} & \cdots & w_{2,\ell_1} \\ \cdots & \cdots & \cdots & \cdots \\ w_{N,1} & w_{N,2} & \cdots & w_{N,\ell_1} \end{pmatrix}, \quad \mathbf{W}_2, \quad \dots, \quad \mathbf{W}_\ell$$

- An *output layer* $\mathbf{y} = [y_1, y_2, \dots, y_n]$, where each perceptron yields a component of the final output. The number of nodes in the output layer depends on the nature of the task.

The weights in an MLP are updated during the training process, by using an optimization algorithm called *backpropagation algorithm*, often combined with the *Gradient Descent* (GD) procedure [2]: after feeding an input to the network, the output is compared against the desired result $\bar{y_i}$ and an error is computed by means of a *loss function*. For multi-class classification tasks, a typical loss function is the categorical cross-entropy L, which is defined as follows:

$$L = -\sum_{i=1}^{n} (\bar{y_i} - y_i) \cdot \log(y_i)$$

This process is repeated for each piece of data in the dataset used as training set, concluding an *epoch* of the training process. Multiple epochs are performed to minimize the loss function, thus finding the set of weight values that best models (fits) the training data, with the given structure of the MLP.

Side-channel Attacks. They are a class of non-invasive security threats that exploit the observation of time series (a.k.a. *traces*) related to environmental parameters, e.g., execution time, dynamic power consumption [19] and electromagnetic radiation [30] (a.k.a. side-channel information), during the execution of a cryptographic primitive on a target computational platform, to derive the value of the secret parameters employed by the algorithm. The main idea consists of targeting an operation of the executed algorithm that involves the use of a portion of the secret key, guesses the value of such a key portion out of all possible values, and assesses the value of a *leakage function* fed with the said key hypothesis and the other possible inputs to the considered operation. The evaluation of such a leakage function is meant to mimic (or, better, to be directly proportional) to the actual side-channel measured from the executing device. In case of a power-based side-channel, the CMOS technology employed for most electronic devices suggests to use as leakage function related to the dynamic power consumption triggered by the switching activity induced by the data processing of the targeted operation [25], either the Hamming weight of the computed value or the computed value itself. The values of the leakage function for each possible guess of the considered key can be used in statistical and computational tools that are employed to correlate them with the values derived from the actual side-channel measurement taken from the device.

In *Non-profiled Side-channel Attacks*, the attacker targets directly the device she/he wants to break and collects traces from it. Examples of such techniques are Differential Power Analysis (DPA) [20] and Correlation Power Analysis (CPA) [8]. Our interest is towards another class of attacks called *Profiled Side-channel Attacks*, where the attacker is assumed to have access to another instance of the same device. In such a scenario, she/he performs her/his analysis on such an additional instance and computes a leakage model by taking as many measurement as necessary from the additional device (*profiling phase*). In a subsequent step, the attacker measures the side-channel from the attacked device only a few times (ideally, one time) to combine such data with the pre-computed model and so derive the secret key of the targeted device (*attack phase*). This is the base upon which Template Attacks (TA) [11] and Deep Learning SCAs are performed. The main difference between the two approaches is in the construction of the leakage model, which is performed applying a statistical approach for TAs, while a data-driven approach is followed when a DL technique is applied.

Deep learning Side-channel Attacks. These are a particular class of Profiled SCAs that leverage DL to retrieve the secret key, which has been shown to outperform traditional machine learning techniques [22]. Their advantage over other profiled methods is that there is no need to use data preprocessing, although sometimes it can improve performance [26], or to make assumptions about the noise distribution. The ability of NN to perform automatic feature extraction allows the profiling phase, also called training phase (we will use the two terms interchangeably), to be completely automated. In many cases only a few traces from the target device are required during the attack phase to fully retrieve the secret key [28]. While many approaches are possible, the two main types of model used are the Convolutional Neural Network (CNN), which has also shown resilience against temporal misalignment in the trace measurements [10], and the Multilayer Perceptron. In general one cannot *a priori* prefer one type of network architecture over another, as seen in other works [26].

To have a realistic attack scenario, the profiling and attack devices must be different, thus making sure that the resulting model is more general and that it does not overfit on the specific peculiarities of the device. This topic has been explored in detail in [6], concluding that the performance of an attack can be vastly overestimated if only a single device is considered during the training and testing phases. This is due to the *portability problem.* The authors propose the Multiple Device Model (MDM) as a solution to the problem, where the NN is trained using traces from different devices.

3 Methodology

The objective of this work is to evaluate different approaches to the dataset collection and training method in the context of DL-SCA.

For this purpose, we have collected datasets that use a fixed or random key from multiple devices of the same type, as detailed in Sect. 3.1, by covering the first round *SBox* operation of the AES cipher. To select the best model, multiple models are trained and evaluated by using a genetic algorithm to select the best set of hyperparameters from the best performing ones; this procedure is described in Sect. 3.2. After this selection, the best performing model is trained on the whole dataset available. Different options are explored. Details on the training process are in Sect. 3.3:

– **target**: observe the effect of using different target intermediates on the network performance. In particular we use the identity leakage model, by taking directly the output of the *SBox* as label for the network (`SBOX_OUT` scenario) or its Hamming weight (`HW_SO` scenario)
– **multi-device**: validate the work in [6], by checking whether our device is also affected by the portability problem, and see if the performance improves when using traces from multiple devices in the training set and attacking the same or different devices.
– **plaintext**: find out what happens when giving more information to the NN besides the raw trace data. Giving the plaintext used during the encryption as additional input to the NN is possible, since during the profiling phase the attacker has a full control on the device, and also during testing it is necessary to have this information when retrieving the key. This is also easy to implement and does not require additional preprocessing of data. To our knowledge, such an approach has not been tested yet in the literature. We indicate the two scenarios with "ptx" when the plaintext is used, and "no ptx" when it is not used.

All the evaluation work was carried out on a virtual machine running Debian 12, with 20 CPU cores, 128 GB of RAM and two Nvidia A100 40GB GPUs. The computational times reported below refer to this system. The GPUs are used to train two models in parallel to speed up the results.

3.1 Dataset Collection

Fig. 2 shows the Riscure Pinata [32] board employed in our experiments, which is based on an STM32F4 microcontroller with an ARM Cortex-M4, a widely used low-power processor that features a 32-bit architecture and is clocked at 168 MHz. The power consumption is measured with a current probe connected in series with the board, which asserts a trigger signal on one of its GPIO pins at the beginning of the *SBox* computation of the first AES round, and deasserts it when the operation concludes, just before the *MixColumns* operation. The cipher is implemented in software without the use of countermeasures or optimizations,

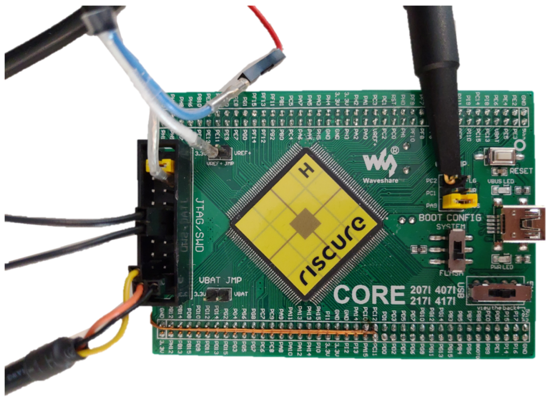


Fig. 2. Capturing setup. Blue, white and grey wires in the upper left are connected to the current probe to measure power consumption. In the bottom left there is the serial connection to the capturing machine. The probe to the right on the PC2 pin is the trigger connected to the scope

and the *SBox* operation specifically is realized as a lookup in a precomputed table stored in memory. This signal is used by an oscilloscope, a Tektronik MSO58, to start the capture of the power trace from the current probe, and ensures that all the traces are temporally well aligned. The sampling rate is set to 625 MHz, with a vertical resolution of 8 bits. A single capture consists of 4, 402 samples. The traces are then resampled at 168 MHz in order to reduce the number of input neurons to the NN, thus obtaining traces with 1, 183 samples.

To be able to test the portability scenario, the traces were captured from three different boards, denoted as D1, D2 and D3. For each board the data collected consists of:

– 200, 000 traces collected with a fixed 128-bit key, `key 0`, see Tab. 1. This is used in the fixed key scenario. Collection time is about 4 hours.
– 200, 000 traces collected using a randomly generated key for each capture, so that each trace uses a different key. This is used in the random key scenario. This is similar to the ASCAD [5] variable key dataset, however in ASCAD one third of the measured traces uses a single fixed key, while in our dataset all traces use random keys. Collection time is about 5 hours, longer than the fixed key scenario because an additional command is needed to set the key at every capture.
– 30, 000 traces collected with a fixed 128-bit key, namely `key 1`, see Tab. 1, different from `key 0`. This is used during the testing phase to evaluate the performance of the network. Collection time is about 40 minutes.

The plaintext is selected randomly at each capture, and is stored in the trace metadata together with the key used and the resulting ciphertext. The software used to perform the capture is `Riscure Inspector` [31], so the traces are saved

Table 1. Fixed keys used

key	**purpose**	**value (bytes)**
`key 0`	training	`98 84 37 ED BA 40 4D DF 83 27 59 57 43 DF D4 FF`
`key 1`	testing	`36 A0 0A BD F4 01 B9 4E 15 78 E1 B7 02 5E 58 F7`

Table 2. Hyperparameter space

parameter	**possible values**
hidden layers	[1, 2, 3, 4, 5]
hidden neurons	[100, 200, 300, 400, 500]
dropout rate	[0.0, 0.1, 0.2, 0.3]
l2	$[0.0, 5 \cdot 10^{-2}, 1 \cdot 10^{-2}, 5 \cdot 10^{-3}, 1 \cdot 10^{-3}, 5 \cdot 10^{-4}, 1 \cdot 10^{-4}]$
optimizer	['adam', 'rmsprop', 'sgd']
learning rate	$[5 \cdot 10^{-3}, 1 \cdot 10^{-3}, 5 \cdot 10^{-4}, 1 \cdot 10^{-4}, 5 \cdot 10^{-5}, 1 \cdot 10^{-5}]$
batch size	[128, 256, 512, 1024]

Table 3. Parameters for the genetic algorithm

nGen	**popSize**	**selPerc**	**scProb**	**mProb**
20	15	30%	20%	20%

in the `.trs` format. The fixed key dataset is available at [33] and the random key one at [34].

3.2 Hyperparameter Tuning

The MLP input layer has as many neurons as the samples in the trace, plus one extra neuron for the models trained with plaintext information, while the output layer can have either 256 neurons in the `SBOX_OUT` scenario or 9 neurons when targeting `HW_SO`. The SOFTMAX activation function gives a probability distribution over all the possible values of the selected target intermediate.

To build the network a hyperparameter tuning is performed, by searching for the optimal parameters in the space defined in Tab. 2. To perform the tuning a genetic algorithm [24] is employed. The specific genetic algorithm chosen here is the one used by Matt Harvey et al. [17]. Settings for the algorithm are reported in Tab. 3. The algorithm starts by generating *popSize* different networks by selecting a random combination of hyperparameters from the *hpSpace*, reported in Tab. 2. Then, each element of the initial population is used to build an MLP, which is successively trained on the training set and evaluated on the validation set. The validation loss of each network is stored and used to compare the performances of all the networks in order to select the top *selProb* best performing ones, and also some bad performing ones with *scProb* probability, to add diversity to the population. The selected combinations are used to generate *offsprings*

Table 4. Selected hyperparameters for all the possible cases

dataset	target	variant	n. devices	layers	neurons	dropout	l2
fixed key	SBOX OUT	no ptx	1	4	200	0.1	$1 \cdot 10^{-2}$
			2	4	300	0.2	$1 \cdot 10^{-2}$
		ptx	1	4	500	0.0	0.0
			2	5	200	0.0	0.0
	HW SO	no ptx	1	4	100	0.1	$1 \cdot 10^{-3}$
			2	5	500	0.3	$5 \cdot 10^{-4}$
		ptx	1	3	200	0.1	0.0
			2	5	300	0.0	$1 \cdot 10^{-4}$
random key	SBOX OUT	no ptx	1	4	200	0.1	$1 \cdot 10^{-4}$
			2	4	300	0.2	0.0
		ptx	1	5	400	0.2	0.0
			2	5	400	0.3	0.0
	HW SO	no ptx	1	5	400	0.2	0.0
			2	5	500	0.3	0.0
		ptx	1	4	400	0.3	$5 \cdot 10^{-3}$
			2	4	100	0.0	$5 \cdot 10^{-3}$

that will compose the new population. Offspring combinations are composed by randomly choosing from the selected parents, but sometimes a random *mutation* can happen to some hyperparameters with probability *mProb*. In this case a random value from Tab. 2 is selected. This process is repeated *nGen* times, and at the end the best performing configuration is selected, according to the validation loss. The results of the selection are reported in Tab. 4. Optimizer, learning rate and batch size are not included due to lack of space. The optimizer selected was always `adam`. The time required to complete the tuning varies considerably, depending on the complexity of the models that are selected during the process, ranging from a few hours to almost a full day per case. Considering all the different cases that have been tested, this is the most time consuming part of the process.

3.3 Training

After choosing the optimal model, a neural network with the selected hyperparameters is built and trained on the dataset. The dataset composed of 200,000 traces (fixed key or random key, depending on the specific model) is loaded and labeled according to the selected target (`SBOX_OUT` or `HW_SO`), then the traces are randomly shuffled. If multiple devices are used in training, the total number of traces is always 200,000, coming in equal amount from every device, and selecting a random subset of all the available traces from each one.

The loaded data is then divided into a training set and a validation set, using a 90/10 training / validation split, and scaled to fit in the $[0-1]$ range as follows.

$$X_{\texttt{scaled}} = \big(X - \mathbf{min}(X)\big) \;/\; \big(\mathbf{max}(X) - \mathbf{min}(X)\big)$$

If the plaintext is used, the corresponding byte is appended at the end of the trace. We tried both scaling it to the same range and not doing that. The networks trained without scaling the plaintext consistently perform slightly better than the ones with the scaled plaintext. Therefore, we avoided scaling the appended byte in all the experiments. At this point the neural network is built according to the selected hyperparameters, which specify the number of layers, the number of neurons and all the other network parameters. All the code related to the NN was implemented by using TENSORFLOW 2.14.0 [1] and KERAS 2.14.0 [12], and it is available at [35]. Some regularization options are also enabled, which help in preventing overfitting and render the network more robust:

- **Dropout:** which percentage of neurons to turn off in a layer during training. This value is tuned during the hyperparameter search.
- **L2 regularization**: limits the value of the weights by adding a penalty to larger values. This value is also tuned.
- **Batch normalization** [18]: normalizes the input to each layer, thus helping in stabilizing the training process.
- **Early stopping** [29]: monitors training and validation accuracy, and stops the training process if overfitting is detected, i.e., when the training accuracy keeps increasing but the validation accuracy starts decreasing.

The resulting network is trained on the training set for 200 epochs or until the early stopping monitor blocks the process, targeting the fifth byte of the AES key. Learning rate scheduling is used to reduce the learning rate when the validation loss becomes stable.

3.4 Evaluation Metric

The Guessing Entropy (GE) is often used during the evaluation of the attack [38]. It is defined as the average rank of the correct key byte when sorting the key predictions by their probability value in ascending order. The probabilities from different predictions are multiplied together [39]. Ideally, the more traces are added, the more the compound probability of the correct key among all predictions should increase, thus leading to a GE of 0 (first place in the ranking is at index 0). The graphs reported in Sect. 4 report the evolution of the correct key position from 1 to 300 traces used in the prediction, thus highlighting when it reaches 0. Since we have $30,000$ test traces, it is possible to divide them in 100 disjoint sets of 300 traces and average the results. The number of traces required for the GE to converge is an indicator of the performance of the network. A better model will need fewer traces to correctly guess the key byte.

4 Performance Results

In this section, the attack performance results are presented, classified by the dataset and training method used. Both attack targets `SBOX_OUT` and `HW_SO` are considered, in each section, highlighting similarities and differences. Finally, the results are summarized in Tab. 5 and Tab. 6. All the graphs are structured in the same way so that they can be compared easily, by reporting the average GE value w.r.t. the number of traces used to perform the attack. All the figures are also available in svg format on the code repository [35], so that one can easily download and inspect them at a higher resolution.

In each graph there are three color-coded lines, which represent three different scenarios, with a slightly different interpretation depending on whether the model was trained with a fixed or random key. In the fixed key case we consider the same scenario as in [6], excluding the "same device and same key" one since it is trivially unrealistic:

– The green line, labeled `same_devs_diff_key`, represents the scenario that does not consider portability, as the targeted device is the same as the profiling one, while the test traces are collected using a different fixed key.
– The blue line, labeled `diff_devs_same_key`, identifies the case in which the device under attack is different from the one used in training, but the attack traces used are collected using the same key. This is only used to compare against the full portability scenario, detailed below.
– The red line, labeled `diff_devs_diff_key`, represents the full portability scenario, where during the attack both the device and the key used are different from the ones in the training set. This is the realistic attack scenario and the one that should be used to evaluate the attack performance.

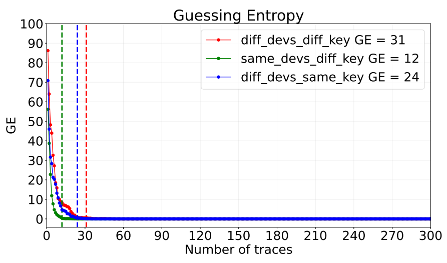
In the random key case we also have the three graphs, but the concept of same/different key no longer applies. Instead, the three scenarios become:

– The green line, labeled `same_devs_key_1`, targets the same device used during training, by means of traces captured with `key 1`.
– The blue line, labeled `diff_devs_key_0`, targets different devices from the ones used during training, by means of traces captured with `key 0`.
– The red line, labeled `diff_devs_key_1`, targets different devices from the ones used during training, by means of traces captured with `key 1`. These last two scenarios are useful to check whether performance changes significantly depending on the specific target key.

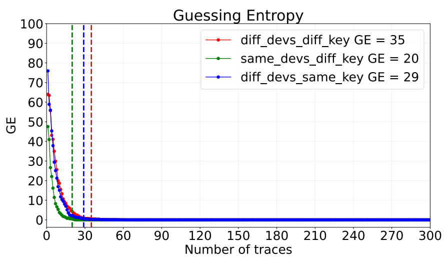
In every case the vertical dashed line represents the number of traces needed to achieve $GE < 0.5$, and that is the value reported in the legend.

4.1 Training with fixed-key dataset

In Fig. 3 we can see the evolution of the GE with an increasing number of traces. This is the most common case that matches the expectations and results in the

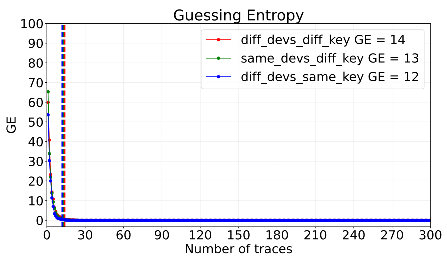


(a) `SBOX_OUT`

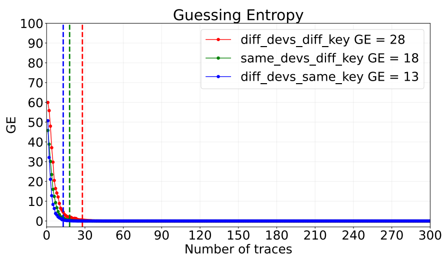


(b) `HW_SO`

Fig. 3. Fixed key, training with 1 device



(a) `SBOX_OUT`



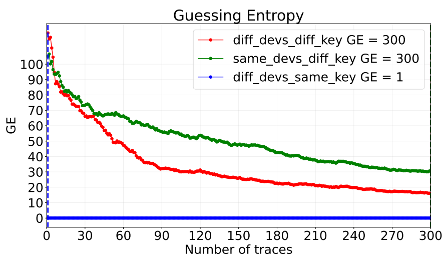
(b) `HW_SO`

Fig. 4. Fixed key, training with 2 devices

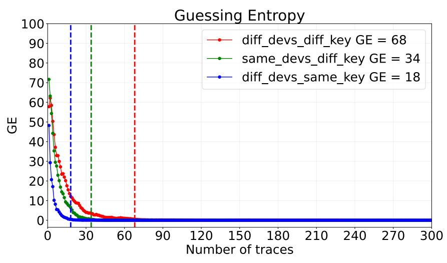
related literature, employing also similar or different computational platforms, e.g., 8-bit microcontrollers. Indeed, we can see that both methods can successfully recover the correct key byte, with the `SBOX_OUT` performing slightly better than the `HW_SO`. The portability concern is also evident here, as the performance decreases when attacking a different device from the one used during the profiling phase.

When using the multi-device model, taking traces from two different devices to train the network, we can see in Fig. 4 that the performance increases, requiring fewer traces to recover the key. This shows that, at least for a fixed training key dataset, using multiple devices for profiling is to be preferred, thus confirming the results found in [6]. Also when considering the `SBOX_OUT`, we can see that the portability becomes much less prominent, with all three graphs becoming almost identical, showing that the obtained model is more general and can be applied to different devices without a significant performance hit.

When adding the plaintext information during the training, the performance of the network gets worse, as reported in Fig. 5. This was not expected, as in principle more information should be beneficial to the network. In particular we see a confirmation of this in two different cases. On one hand, training on the `SBOX_OUT` target appears to be overfitting on the specific key value, since

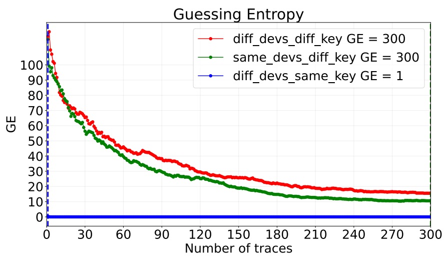


(a) `SBOX_OUT`

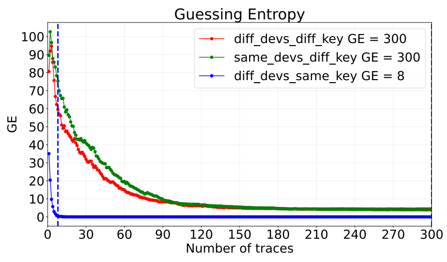


(b) `HW_SO`

Fig. 5. Fixed key, training with 1 device and plaintext



(a) `SBOX_OUT`



(b) `HW_SO`

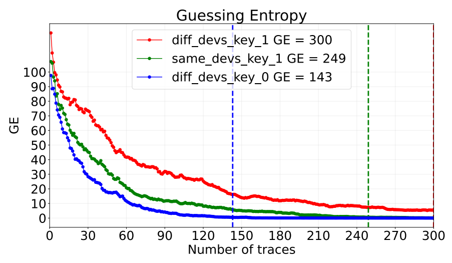
Fig. 6. Fixed key, training with 2 devices and plaintext

when guessing against traces taken from another device but using the same key, the line in blue, the network only needs one trace to correctly recover it. Further investigating the motivation underlying this behaviour, we found that the weights of the network are heavily biased in correspondence of the plaintext information. This may lead to an underestimation of the leakage due to the key. On the other hand, the cases against a different key show a decrease in performance of more than one order of magnitude. For the `HW_SO` we also see a decrease in performance, but not so severe, and the network is still able to learn how to extract the key in all three cases, and the portability scenario in red still proves to be more difficult than the other two.

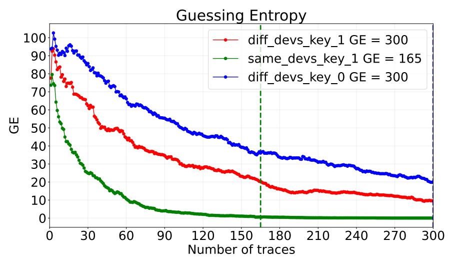
When considering multiple devices, `SBOX_OUT` behaves the same, while `HW_SO` gets worse, as it can be seen in Fig. 6, with two out of three scenarios not converging to $GE = 0$ but flattening out at $GE = 5$. Counterintuitively, we repeatedly confirmed that when using a dataset with a fixed key and providing the plaintext information to the network, this yields to worse performance overall.

4.2 Training with random-key dataset

In the case of random key, it is evident that the performance is worse both for `SBOX_OUT` and `HW_SO`. The trend of the former performing better than the latter

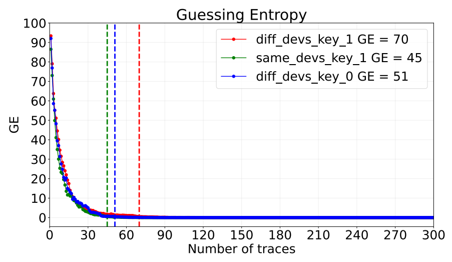


(a) `SBOX_OUT`

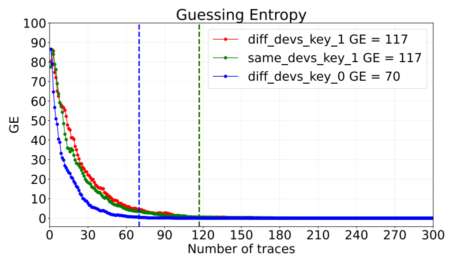


(b) `HW_SO`

Fig. 7. Random key, training with 1 device



(a) `SBOX_OUT`

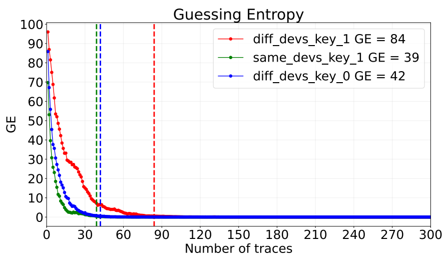


(b) `HW_SO`

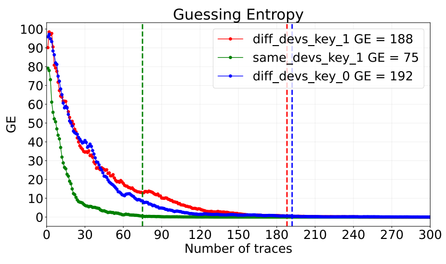
Fig. 8. Random key, training with 2 devices

is still present here. The interesting behaviour that appears from the graphs, is that in this case the performance of the model also depends on the key used during testing. For instance, in Fig. 7 using `key 0` performs better than using `key 1`.

However, we also maintain the problem of the portability when training with a single device. In fact, the GE converges to 0 faster when attacking traces captured with `key 1` on the same device used during training, while it needs more traces when attacking the same `key 1` but with traces from a different device. Similarly to the fixed key scenario, also when using a random key the performance increases when using the Multi-Device Model, but in this case the increase is more significant, although w.r.t. fixed key the overall performance is still lower. We see a difference from the fixed key scenario when considering what happens with the plaintext. When considering only one training device, the performance increases in both `SBOX_OUT` and `HW_SO` w.r.t. not using the plaintext. This contrasts what was seen with the fixed key, where performance became significantly worse. When considering two training devices (see Fig. 10), in the `HW_SO` the network seems to learn with greater difficulty, requiring a great number of traces to lower the guessing entropy. Instead, when considering the `SBOX_OUT` case the performance is better than without plaintext, and approaches

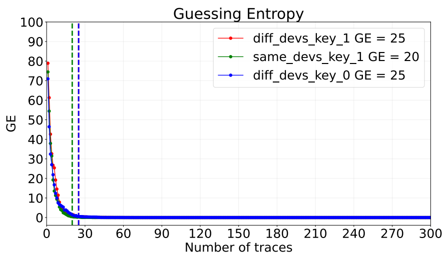


(a) `SBOX_OUT`

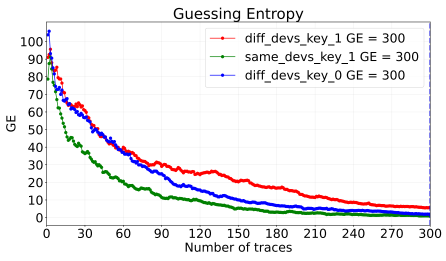


(b) `HW_SO`

Fig. 9. Random key, training with 1 device and plaintext



(a) `SBOX_OUT`



(b) `HW_SO`

Fig. 10. Random key, training with 2 devices and plaintext

the performance of the fixed key case, also closing the gap in the portability scenario, as the network requires just five more traces to correctly guess the key when considering a different target device.

4.3 Comparisons

Here we summarize all the guessing entropy behaviours discussed in Sect. 4 in Tab. 5 and Tab. 6. We note that having a dataset of traces collected using a single fixed key is the best option, both for a single or multiple training devices. Unexpectedly, the use of the plaintext information should be avoided in this case. It can instead be a great advantage when using a dataset collected using random keys, depending on the target leakage function that is selected to train the network and on the number of devices used.

5 Related Work

There is a number of publicly available datasets that target AES-128, both with and without countermeasures in place [4,5,7,14,16], but all of them only capture traces from a single device for both the training and testing phases, making them

Table 5. Results for the 'fixed key' dataset. No. of traces to get $GE < 0.5$

target	variant	n. devices	same_devs_diff_key	diff_devs_same_key	diff_devs_diff_key
SBOX OUT	no ptx	1	12	24	31
		2	13	12	14
	ptx	1	> 300	1	> 300
		2	> 300	1	> 300
HW SO	no ptx	1	20	29	35
		2	18	13	28
	ptx	1	34	18	68
		2	> 300	8	> 300

Table 6. Results for the 'random key' dataset. No. of traces to get $GE < 0.5$

target	variant	n. devices	same_devs_key_1	diff_devs_key_0	diff_devs_key_1
SBOX OUT	no ptx	1	249	143	> 300
		2	45	51	70
	ptx	1	39	42	84
		2	20	25	25
HW SO	no ptx	1	165	> 300	> 300
		2	117	70	117
	ptx	1	75	192	188
		2	> 300	> 300	> 300

unfit for testing the portability scenario. The dataset that we sampled and used in our analyses, which we also made publicly available in [33,34], provides the advantage of including measurements from a set of distinct instances of the same device. This feature makes our dataset more suitable for testing realistic attack scenarios.

Concerning the definition of the leakage function as the Hamming weight of the result of the considered operation, i.e., SBOX output, our conclusions align with those in [27], that is, the performance of DL-based attacks decreases because of the imbalance induced by the set of Hamming weight values, as they allow to gather the dataset samples in nine classes with quite different cardinalities.

Concerning the portability of the DL-SCA vulnerability proved when considering a single device instance or multiple device instances during the training phase, we mostly confirm the results provided in [6], where the attackers considered an AES-128 software implementation for an 8-bit microprocessor clocked at only 16 MHz (in contrast to our 32-bit CPU clocked at 168 MHz). Indeed we show that, when only one device is used for profiling, attacking a device different from the profiled one leads to a decrease in the performance of the model. We also show that using traces from multiple device instances for training, in general, helps the model to better generalize and to become more portable.

In our study, see Fig. 10b, a notable discrepancy is highlighted by the use, during the training phase, of the plaintext values fed to each cipher run and of traces obtained with a random key, when a worsening of the attack performance is observed only in the case of a dataset composed by traces coming from multiple device instances (two).

6 Concluding Remarks

We have provided a systematic comparison of different approaches to train an MLP for power-based SCA against a software implementation of AES. We have tested the effect of changing different parameters, such as the type of traces used, i.e., with fixed or random key, the target leakage function used, and the usage of plaintext information. We have validated results on a 32-bit CPU platform, namely the Riscure Pinata. In addition, we have provided a new study on the use of the plaintext information during the training, thus showing that its influence depends on other factors, like the choice of target or the available data. In the end, using the plaintext provides a benefit when dealing with datasets collected using a random key. We have found that when collecting a dataset, using a single fixed key is the better choice in terms of attack performance, but that it is also possible to get a similar, albeit lower, performance by using random keys and providing plaintext information. We have performed our tests also considering the portability of the model, and we have confirmed that using traces from multiple training devices helps in building a better model. Finally, we provide a new dataset to perform further studies concerning the portability on a complex target, thus filling the gap present in the currently available public datasets.

Future work will consider protected AES implementations, e.g., by masking, and the effect of selecting other points of interest (other than the *SBox* output) from the traces. It would also be interesting to study the behaviour of optimized AES implementations. In all these cases, different intermediate values need to be chosen, thus leading to the use of different and potentially more complex leakage models. Further experimentation is needed to evaluate the attack effort required. This also applies if we consider ciphers other than AES. Since the leakage behaviour is tightly related to the microarchitectural details of the processor [3], our results are representative for targets based around the same processor, i.e., Cortex M4, which is widely diffused. Processors with a significantly different microarchitecture may exhibit other behaviours.

Acknowledgments. This work was supported in part by project SERICS (PE00000014) under the NRRP MUR program funded by the EU – NGEU. We also thank Luca Castellazzi for his initial realization of our NN models.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Aggarwal, C.C.: Neural Networks and Deep Learning - A Textbook. Springer (2018). https://doi.org/10.1007/978-3-319-94463-0, https://doi.org/10.1007/978-3-319-94463-0

3. Barenghi, A., Pelosi, G.: Side-channel security of superscalar cpus: evaluating the impact of micro-architectural features. In: Proceedings of the 55th Annual Design Automation Conference, DAC 2018, San Francisco, CA, USA, June 24-29, 2018. pp. 120:1–120:6. ACM (2018). https://doi.org/10.1145/3195970.3196112, https://doi.org/10.1145/3195970.3196112
4. Bellizia, D., Bronchain, O., Cassiers, G., Momin, C., Standaert, F.X., Udvarhelyi, B.: Spook SCA CTF (2021). https://doi.org/10.14428/DVN/W2SV5G, https://doi.org/10.14428/DVN/W2SV5G
5. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptogr. Eng. **10**(2), 163–188 (2020). https://doi.org/10.1007/S13389-019-00220-8, https://doi.org/10.1007/s13389-019-00220-8
6. Bhasin, S., Chattopadhyay, A., Heuser, A., Jap, D., Picek, S., Shrivastwa, R.R.: Mind the portability: A warriors guide through realistic profiled side-channel analysis. In: 27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020. The Internet Society (2020), https://www.ndss-symposium.org/ndss-paper/mind-the-portability-a-warriors-guide-through-realistic-profiled-side-channel-analysis/
7. Bhasin, S., Jap, D., Picek, S.: Repository for AES_HD. https://github.com/AESHD/AES_HD_Dataset (2018)
8. Brier, E., Clavier, C., Olivier, F.: Optimal Statistical Power Analysis. IACR Cryptol. ePrint Arch. p. 152 (2003), http://eprint.iacr.org/2003/152
9. Brier, E., Clavier, C., Olivier, F.: Correlation Power Analysis with a Leakage Model. In: Joye, M., Quisquater, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004). https://doi.org/10.1007/978-3-540-28632-5_2, https://doi.org/10.1007/978-3-540-28632-5_2
10. Cagli, E., Dumas, C., Prouff, E.: Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Preprocessing. In: Fischer, W., Homma, N. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. Lecture Notes in Computer Science, vol. 10529, pp. 45–68. Springer (2017). https://doi.org/10.1007/978-3-319-66787-4_3, https://doi.org/10.1007/978-3-319-66787-4_3
11. Chari, S., Rao, J.R., Rohatgi, P.: Template Attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3, https://doi.org/10.1007/3-540-36400-5_3
12. Chollet, F., et al.: Keras. https://keras.io (2015)
13. Clavier, C., Feix, B., Gagnerot, G., Roussellet, M., Verneuil, V.: Horizontal Correlation Analysis on Exponentiation. In: Soriano, M., Qing, S., López, J. (eds.) Information and Communications Security - 12th International Conference, ICICS 2010, Barcelona, Spain, December 15-17, 2010. Proceedings. Lecture Notes in Computer Science, vol. 6476, pp. 46–61. Springer (2010). https://doi.org/10.1007/978-3-642-17650-0_5, https://doi.org/10.1007/978-3-642-17650-0_5
14. Coron, J.S., Kizhvatov, I.: Trace Sets with Random Delays – AES_RD. https://github.com/ikizhvatov/randomdelays-traces (2009)

15. Cybenko, G.: Approximation by superpositions of a sigmoidal function. Math. Control. Signals Syst. **2**(4), 303–314 (1989). https://doi.org/10.1007/BF02551274, https://doi.org/10.1007/BF02551274
16. Fei, Y.: Northeastern University TeSCASE Dataset – AES_HD_MM. https://chest.coe.neu.edu/?current_page=POWER_TRACE_LINK&software=ptmasked (2014)
17. Harvey, M., Heckscher, N.: Evolve a neural network with a genetic algorithm (2017), https://github.com/harvitronix/neural-network-genetic-algorithm
18. Ioffe, S., Szegedy, C.: Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In: Bach, F.R., Blei, D.M. (eds.) Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015. JMLR Workshop and Conference Proceedings, vol. 37, pp. 448–456. JMLR.org (2015), http://proceedings.mlr.press/v37/ioffe15.html
19. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25, https://doi.org/10.1007/3-540-48405-1_25
20. Kocher, P.C., Jaffe, J., Jun, B., Rohatgi, P.: Introduction to differential power analysis. J. Cryptogr. Eng. **1**(1), 5–27 (2011). https://doi.org/10.1007/S13389-011-0006-Y, https://doi.org/10.1007/s13389-011-0006-y
21. Lisovets, O., Knichel, D., Moos, T., Moradi, A.: Let's take it offline: Boosting brute-force attacks on iphone's user authentication through SCA. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2021**(3), 496–519 (2021). https://doi.org/10.46586/TCHES.V2021.I3.496-519, https://doi.org/10.46586/tches.v2021.i3.496-519
22. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking Cryptographic Implementations Using Deep Learning Techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings. Lecture Notes in Computer Science, vol. 10076, pp. 3–26. Springer (2016). https://doi.org/10.1007/978-3-319-49445-6_1, https://doi.org/10.1007/978-3-319-49445-6_1
23. McCulloch, W.S., Pitts, W.H.: A Logical Calculus of the Ideas Immanent in Nervous Activity. In: Boden, M.A. (ed.) The Philosophy of Artificial Intelligence, pp. 22–39. Oxford readings in philosophy, Oxford University Press (1990)
24. Mitchell, M.: An introduction to genetic algorithms. MIT Press (1998)
25. Ng, L.L., Yeap, K.H., Goh, M.W.C., Dakulagi, V.: Power consumption in cmos circuits. In: Song, H.Z., Yeap, K.H., Goh, M.W.C. (eds.) Electromagnetic Field in Advancing Science and Technology, chap. 5. IntechOpen, Rijeka (2022). https://doi.org/10.5772/intechopen.105717, https://doi.org/10.5772/intechopen.105717
26. Perin, G., Wu, L., Picek, S.: Exploring Feature Selection Scenarios for Deep Learning-based Side-channel Analysis. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2022**(4), 828–861 (2022). https://doi.org/10.46586/TCHES.V2022.I4.828-861, https://doi.org/10.46586/tches.v2022.i4.828-861
27. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2019**(1), 209–237 (2019). https://doi.org/10.13154/TCHES.V2019.I1.209-237, https://doi.org/10.13154/tches.v2019.i1.209-237
28. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: SoK: Deep Learning-based Physical Side-channel Analysis. ACM Comput. Surv. **55**(11), 227:1–227:35 (2023). https://doi.org/10.1145/3569577, https://doi.org/10.1145/3569577

29. Prechelt, L.: Early stopping - but when? In: Montavon, G., Orr, G.B., Müller, K. (eds.) Neural Networks: Tricks of the Trade - Second Edition, Lecture Notes in Computer Science, vol. 7700, pp. 53–67. Springer (2012). https://doi.org/10.1007/978-3-642-35289-8_5, https://doi.org/10.1007/978-3-642-35289-8_5
30. Quisquater, J., Samyde, D.: ElectroMagnetic Analysis (EMA): Measures and Counter-Measures for Smart Cards. In: Attali, I., Jensen, T.P. (eds.) Smart Card Programming and Security, International Conference on Research in Smart Cards, E-smart 2001, Cannes, France, September 19-21, 2001, Proceedings. Lecture Notes in Computer Science, vol. 2140, pp. 200–210. Springer (2001). https://doi.org/10.1007/3-540-45418-7_17, https://doi.org/10.1007/3-540-45418-7_17
31. Riscure: Inspector (2024), https://www.riscure.com/security-tools/inspector-sca/
32. Riscure: Pinata (2024), https://www.riscure.com/products/pinata-training-target/
33. Roberto Capoferri: Fixed-key dataset for three riscure Pinata devices. https://zenodo.org/records/11443025 (2024)
34. Roberto Capoferri: Random-key dataset for three riscure Pinata devices. https://zenodo.org/records/11199202 (2024)
35. Roberto Capoferri: Source code and models. https://github.com/RobertoCapoferri/DLSCA-article (2024)
36. Roche, T.: EUCLEAK. Cryptology ePrint Archive, Paper 2024/1380 (2024), https://eprint.iacr.org/2024/1380
37. Silva, T.C., Zhao, L.: Machine Learning in Complex Networks. Springer (2016). https://doi.org/10.1007/978-3-319-17290-3, https://doi.org/10.1007/978-3-319-17290-3
38. Standaert, F., Malkin, T., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings. Lecture Notes in Computer Science, vol. 5479, pp. 443–461. Springer (2009). https://doi.org/10.1007/978-3-642-01001-9_26, https://doi.org/10.1007/978-3-642-01001-9_26
39. Wu, L., Weissbart, L., Krček, M., Li, H., Perin, G., Batina, L., Picek, S.: On the attack evaluation and the generalization ability in profiling side-channel analysis. Cryptology ePrint Archive, Paper 2020/899 (2020), https://eprint.iacr.org/2020/899