Toward Practical Autoencoder-based Side-Channel Analysis Evaluations

Servio Paguada[1,2][0000−0003−4665−7457], Lejla Batina[1][0000−0003−0727−3573], and Igor Armendariz[2][0000−0002−5055−455X]

[1] Digital Security Group, Radboud University, Nijmegen, The Netherlands
servio.paguadaisaula@ru.nl, lejla@cs.ru.nl
[2] Ikerlan Technology Research Centre, Arrasate-Mondragón, Gipuzkoa, Spain
{slpaguada,iarmendariz}@ikerlan.es

Abstract. This paper introduces a practical evaluation procedure based on autoencoders for profiled side-channel analysis evaluations. An autoencoder is a learning model able to pre-process leakage traces improving in this way the guessing entropy. Nevertheless, this learning model's design should aim to code the leakage distribution to avoid relevant information being removed. For this reason, we propose an autoencoder built upon dilated convolutions. When using these learning models, the evaluation produces new assets, e.g., new versions of the dataset and new models based on learning algorithms. Our procedure comprises meaningful metrics and visualization techniques, namely signal-to-noise ratio and weight visualization, to evaluate those assets' effectiveness. After applying our procedure and our new autoencoder architecture to the ASCAD random key database, our results outperform state-of-the-art.

Keywords: profiled attacks · side-channel analysis · dilated convolutions · autoencoders · convolutional neural network

1 Introduction

Side-Channel Analysis (SCA) comprises a set of techniques to exploit information buried in signals (named leakage traces) that becomes available from the devices in the form of sound, heat, electromagnetic radiation, or power consumption. An adversary is someone that uses this information to steal devices' secrets, such as cryptographic keys.

In a booming market of the Internet of Things, device manufacturers invest huge resources to prevent this kind of attacks. Scientists are, at the same time, working on new ways for protection of embedded devices. These latter initiatives start with a detailed analysis of the attacks' effectiveness on countermeasures (against SCA).

These efforts have led to discovering new SCA types of attacks; among them, profiled attacks introduced new ways to conduct SCA. The procedure of evaluating this attack's effectiveness implies identifying a target device from which an adversary wants to steal secrets. He/she gets an access to a hardware clone of

the target device and then uses it to collect side-channel information. By using this information, the adversary builds a model aimed to attack the target device.

Profiled attacks were first introduced in the form of Template Attacks [1]. A decade or so later, a new wave of profiled attacks based on machine and deep learning arose [2–4].

The whole procedure to conduct profiled SCA changed ever since, advertising that just a single learning model is enough to conduct the procedure, for example, using Multi-Layer Perceptron (MLP) [4] and, more recently, Convolutional neural network (CNN) [5, 6]. Moreover, it was argued that pre-processing steps such as noise reduction and feature selection, i.e., locating points of interest (PoIs), are no longer required.

Nevertheless, relying on one single model does not reduce the uncertainty of an SCA evaluation [7]. Also, pre-processing steps are still conducted to reduce a learning model's complexity as well as improving its performance [8].

So, instead of taking a single learning model as a standalone solution that does not use any pre-processing for SCA evaluation, we take another approach. We combine those two in a procedure based on autoencoders. Some of these learning model's applications could not discard but rather substitute pre-processing steps for SCA evaluation, not only for noise reduction but also for locating PoIs.

More generally, this type of neural network (NN) could derive other types of attack based on semi-supervised and unsupervised NN models. To our best knowledge, they have not received enough attention yet.

Researchers have used autoencoders to conduct attacks and filter out countermeasures [9, 10]. Contrary to those works, we aim to develop a more general procedure to improve SCA evaluation with alternative deep learning models. Building the foundation for autoencoders will be useful for other researchers that might look into those.

Part of our motivation for building a procedure is the lack of one that uses autoencoder in it. Current profiled SCA procedures and methodologies are not focused on using pre-processing steps since they rely on a single NN model to do the SCA evaluation. Nevertheless, the suggestions on designing a single model for SCA [11–13] are applicable for autoencoders.

In this sense, we increase the potential of a profiled SCA evaluation by distributing the analysis of its assets, namely the dataset, the profiling model (regarding its feature selection performance), into separate tasks. In this way, the procedure systematically generates information early for the evaluator, helping him/her to reduce the uncertainty in this already challenging area that SCA is.

In addition to the new procedure, we also proposed an autoencoder architecture that uses dilated convolutions. First used in [13] the authors show how this type of convolutions could boost the effectiveness of a CNN-based SCA evaluation because of its feature selection performance. To apply dilated convolutions, we extend the criteria addressed in [13] to be used with autoencoder's architecture.

Our experiments demonstrate the proposed procedure with two types of convolutional autoencoders (CAEs); one is our dilated CAE (D-CAE), and the sec-

ond one is a normal CAE (N-CAE). We qualify the assets generated by each autoencoder through the procedure. Finally, we observe how the side-channel attack's effectiveness differs according to those assets. We have tested our result in commonly used datasets for benchmarking, namely, ASCAD [11] (fixed and random key).

Contribution

This work has several contributions as follows:

1. We have derived a novel SCA evaluation procedure based on autoencoders. The procedure comprises of three main tasks, and through them, the evaluator monitors the assets of the evaluation. This ensures that each task's assets are evaluated with well-known metrics.
2. To extend the autoencoder's comprehension in the field, we explain how visualization techniques can be used (with autoencoders) to depict automated feature selection.
3. We introduce a dilated convolution-based autoencoder to use in profiling SCA. To do so, we suggest some criteria concerning autoencoders architecture's design that keeps a significant amount of side-channel information as possible in the latent space.
4. We have tested the procedure and the D-CAE architecture in ASCAD dataset (random key), outperforming previous state-of-the-art results.

Paper organization

Paper is organized as follows. Sect. 2 details theoretical aspects of topics used for this work. Sect. 3 discusses related works. Sect. 4 gives information about datasets used for our experiments. Sect. 5 describes our proposed evaluation procedure. In Sect. 6 we present the extension of the criteria about dilated convolution to be used with autoencoders. Sect. 7 shows results of our experiments and Sect. 8 concludes the paper.

2 Background

Through this section, we contextualize the paper by explaining profiled attacks and CNN architectures. We introduce some theoretical aspects of normal and dilated convolutions, we explain autoencoders, and finally we mention metrics and visualization techniques that we use in the remainder of this work.

2.1 Profiled attacks

Profiled attacks belong to a special type of a side-channel attack where an adversary has under control a device, which is a clone hardware of the actual device he/she wants to attack, which allows us to execute a customized firmware. The

device owned by the adversary is called a profiling device, and he/she uses it to implement a crypto algorithm, draw leakage traces, and train a model (*profiling phase*). The second phase (called *attack phase*) uses that model to attack the actual device.

After its introduction as Template Attacks [1], profiled attacks have received a lot of attention from the research community. Early improvements involve procedures increasing its portability through devices and algorithms about choosing points of interest (PoIs) [14,15] to achieve better SCA results.

When profiled attacks got took on the machine learning twist, a wave of new evaluations came into place [16,17]. It has had such an impact, so that certifying entities demand today that a device under test (DUT) must be evaluated using machine learning-based SCA before being certified and launched into the market.

Machine learning-based profiled attacks' effectiveness highly depend on the feature selection process [8]. When the model is not able to learn from the most representative features, its capability decreases considerably.

Researchers have studied the impact of preprocessing leakage traces to remove as many irrelevant features as possible and improve model effectiveness. Methods like PCA and LDA [18] aim to compress the input signal, keeping the most amount of relevant information as possible in the compressed version. They usually fail in figuring out the function that correctly estimates the distribution of data, which has negative repercussions in the compressed space later on.

There exist implementations of cryptographic algorithms with countermeasures, to protect against profiled attack (or any other side-channel attack). They can be grouped into two categories; masking and hiding countermeasures [19,20]. In this work we are considering masking implementations.

The masking's fundamental concept relies upon minimizing the dependency of the leakage and the sensitive values (secret dependent). Usually, these sensitive values are outputs of some non-linear function in the cryptographic algorithm (see Eq. (9)) of interest, e.g. the S-box computation

If the side-channel attack is effective enough, it can minimize the countermeasure's effect after several algorithms' executions. So, the sensitive values become accessible for the classifier. In the SCA context, the numbers of executions maps to the number of leakage traces.

Usually, masking implementations are also combined with desynchronization, which is achieved by implementing random delays into the DUT software code. With this combination, the machine learning model used to attack them becomes more complex. This fact motivates researchers to look into new methods that allow evaluators to conduct more effective analysis [12, 13, 21, 22].

2.2 Convolutional neural network

CNN is a deep learning model's architecture type; it comprises a feature learning part and a classification part (see Fig. 1). The former aims to create abstract representations called feature maps of the input signal using the most relevant feature.

Feature maps pass through a pooling operation ending up with a pooling feature map. A pooling operation has similar behavior to convolution operations, and it also uses kernel and a value to control its displacement (i.e., the stride hyperparameter). It is responsible for granting the CNN model the capability of dealing with spatial transformations that the input signal could have e.g., desynchronization produced by random delays. Still, this spatial transformations increase two possible factors in evaluating the cryptographic algorithm implementation's resilience; (1) the CNN model's complexity and (2) the number of traces required.

The convolution operation is conducted with sets of kernels. Typically, kernel's elements (a.k.a weights) are initialized with random values, or by using a particular weight initialization function e.g. *Glorot uniform* or *He uniform*. A loss function trains the weights; usually, this loss function is chosen according to the task that the network has to fulfill (e.g., classification or regression).

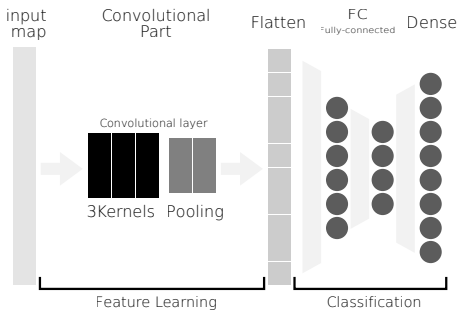


Fig. 1. Convolutional neural network illustration

Through learning iterations, the loss is back-propagated updating kernels' elements [23]. When the feature learning part of a CNN model reaches a satisfactory performance, it means that the kernel's elements are well trained. In other words, the neural network has learned from the input signal's features.

From the SCA perspective, the signal's features should optimally represent the leakage of sensitive values. So, it is crucial that during the training, the model automatically chooses those features (the most relevant ones). The SCA evaluation effectiveness highly depends on that.

Similarly, as happened with SCA based on machine learning, it has been discovered that for profiled attacks based on deep learning, there are also ways to guide the model in choosing the crucial features. In recent works e.g., [12,13,21] the issue on improving this feature selection automatic process to increase CNN classifier's effectiveness in SCA has been addressed. Those recent arguments explain what is in the following summarized:

1. A single convolutional operation has to combine the leak of sensitive values, namely, the mask and the masked sensitive values [21].

2. The same convolution should not overuse those sensitive value, neither irrelevant points [12] and
3. Since fulfilling the previous two statements is tricky for normal convolutions, dilated convolution [13] could be used to achieve them.

From this list, one can assume that in order to design a deep learning model for SCA; an evaluator should know where sensitive value leaks. In white-box scenarios, this is possible because he/she has more details about the DUT (e.g., plaintext, ciphertext, and mask values). A challenging case is a black-box scenario.

Those two scenarios brought up the need to have criteria for designing effective deep learning architectures for SCA. Moreover, new criteria also open new research lines in evaluation procedures designs to allow an evaluator to compare different approaches that ensure an effective SCA evaluation (with fewer false-positives).

We have tested our evaluation procedure in white-box scenarios. Nevertheless, it could be also used in black box scenarios.

Our proposed D-CAE brings new criteria as well as constraints about autoencoder architectures; we discuss those in Sect. 6 on more details. For now, we further discuss normal convolution and dilated convolutions to show how dilated convolution might increase autoencoder's effectiveness in SCA evaluations.

Normal convolutions We refer to normal convolutions as those that keep the operation unaltered. Eq. (1) shows a discrete version of the convolution operation, the same is used for normal CNN models. For explanation purposes, this version suggests that the whole signal f moves from left to right and passing by kernel k.

Here, $f[\cdot]$, and $k[\cdot]$ represent a signal's element and a kernel's element respectively; the amount of elements a kernel has, defines its length l_k (see Eq. (3)). To explain why this type of convolutions have a pitfall related to the feature selection process, let us take a signal time sample $f[i]$ and suppose it goes into the convolution's kernel.

That sample is used until the kernel's end; precisely, it is present in each computation of the kernel with the signal (i.e., l_k times). That is when we say the convolution overuses that sample. Moreover, if the evaluator increases kernel's length pretending to fulfill (1) from the previous list; it also increases the times the same sample is used, which has repercussion in the CNN's performance [12, 13].

$$\begin{aligned} f[x] \circledast k[x] &= \textstyle\sum_{n=-\infty}^{\infty} f[x-n] \cdot k[x] \\ &= \cdots + \\ &\quad (f[x-n_i] \cdot k[n_i]) + \\ &\quad (f[x-n_{i+1}] \cdot k[n_{i+1}]) \\ &\quad + \cdots \\ \text{where:} \quad & x, n \in \mathbb{Z} \end{aligned} \tag{1}$$

Dilated Convolutions A dilated convolution is an alteration that consists of inserting zeros between kernel's elements. Eq. (2) shows this variant from previous definition of discrete convolutions in Eq. (1). The hyperparameter *dilatation rate* (dr) controls the amount of zeros inserted. Eq. (3) shows the arithmetic relationship between original kernel length l_k and dilatation rate dr.

$$\begin{aligned} f[x] \circledast k_d[x] &= \textstyle\sum_{n=-\infty}^{\infty} f[x-n] \cdot k_d[x] \\ &= \cdots + \\ &\quad (f[x-n_i] \cdot 0) + \\ &\quad (f[x-n_{i+1}] \cdot k_d[n_{i+1}]) + \\ &\quad (f[x-n_{i+2}] \cdot 0) + \\ &\quad (f[x-n_{i+3}] \cdot k_d[n_{i+3}]) \\ &\quad + \cdots \\ \text{where:} \quad & x, n \in \mathbb{Z} \\ & k_d \text{ is a dilated kernel} \end{aligned} \tag{2}$$

Contrary to normal convolutions, when the evaluator increases the kernel's length with the same aim as said before, he can do it by increasing the number of zeros in between, i.e., by tuning the dr instead of the l_k hyperparameter.

This action has the same effect of covering a wider section of the signal, but sample points alternate between being or not nullified because of those zeros.

While in one convolution computation, a sample point is multiplied by a non-zero kernel's element, in the following one, it will be. With this alteration, a CNN model fulfills the aspects of the previous list [13].

$$\hat{l_k} = l_k + (l_k - 1)(dr - 1) \tag{3}$$

2.3 Autoencoders

An autoencoder has two parts, an encoder and a decoder [23]. The encoder ends in a so-called latent space (see Fig. 2). This latter corresponds to a compressed version of the input signal. Formally said, latent space is an abstract representation of the input signal; it is the space in which the signal is reduced by taking its most remarkable features. The encoder does this by figuring out a function that represents the data's distribution.

Another way to refer to this process is that the encoder downsamples the original signal by taking it to a new space with fewer dimensions (dimensionality reduction). Encoders can also upsample the signals, which is the exact opposite effect. However, we are not using an autoencoder for that purpose in this paper.

Normally, latent space is taken as part of the encoder, putting the latter as the important part of an autoencoder. Eq. (4) formulates the latent space s, as a function of the encoder e whose parameter is the input signal f.

$$s = e(f) \tag{4}$$

We want the encoder to take the most representative feature to build the latent space, which contains useful information needed by further models, like

CNN's-based models or any other machine learning-based model. The encoder job allows those models to make accurate predictions (or classifications) since several redundant features that contain little or none information were discarded.

There are other implicit benefits in performing dimensionality reduction, namely, model interpretability, minimization of model's overfitting, and reducing the training dataset's size, which implies reducing training time.

Similarly, the decoder takes its input from the latent space to rebuild the original input signal; Eq. (5) shows this relation. A loss function $\mathcal{L}$ at the decoder's end computes the difference between the original input signal and decoder input signal's version (see Eq. (6)), the output of the loss function is then back-propagated to tune the network's weights.

$$d(s) = \tilde{f} \tag{5}$$

Using an encoder also has pitfalls. If the encoder is not well trained or cannot figure out the data distribution, it will also remove relevant features, affecting further model's performance. This paper aims to elaborate this statement in further sections (see Sect. 5 and Sect. 6).

The encoder part is the focus of this contribution. We are inspecting the effects provoked in an autoencoder when combining dilated convolution. An encoder is built by using even multi-layer perceptrons [23] or CNNs.

Our proposed autoencoder uses CNNs. By doing this, the encoder inherits CNN's capacity for dealing with displacement or any other spatial transformation presented in the input signal. Furthermore, when applied dilated CNNs, the requirements in the previous list are also fulfilled.

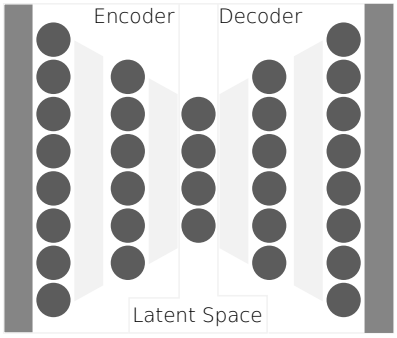


Fig. 2. A typical autoencoder illustration. *Latent space* is where the input signal is taken by the encoder's learning process using its most remarkable features. Decoder's task consists of re-constructing, as much as possible, the original input signal.

Mean square error (MSE) is a commonly used loss function for training autoencoders. It takes two vectors as arguments; namely f, and $\hat{f}$, they correspond

to original leakage trace and predicted leakage trace, respectively (see Eq. (6)). At the end of each learning iteration (a.k.a epoch), the loss function output is then back-propagated through the model to tune the autoencoder's weights.

$$\begin{aligned} MSE = \mathcal{L}(f, \tilde{f}) &= \tfrac{1}{m} \textstyle\sum_{i=1}^{m} (f[i] - \tilde{f}[i])^2 \\ \text{where:} \quad m &= \dim(f) \end{aligned} \tag{6}$$

MSE could also be represented as a function of the output data's distribution $p(f|s)$. as Eq. (7) expresses. From that point of view, the job is to recover the input distribution given a latent space s. Be aware that we are assuming a Gaussian distribution here. A suitable encoder is capable of building a latent space most representative of the data distribution.

$$\begin{aligned} \mathcal{L} &= log(f, s) \\ &= -log \textstyle\prod_{i=1}^{m} \mathcal{N}(f[i], \tilde{f}[i], \sigma^2) \\ &= -\textstyle\sum_{i=1}^{m} log\, \mathcal{N}(f[i], \tilde{f}[i], \sigma^2) \\ &\propto \textstyle\sum_{i=1}^{m} (f[i] - \tilde{f}[i])^2 = \text{MSE} \\ &\text{where:} \quad m = \dim(f) \end{aligned} \tag{7}$$

To better understand how well the learning process went, one common practice is to derive a *validation dataset* from the training dataset. Setting this up, at the end of each learning iteration, samples from the validation dataset are taken to perform predictions. It does not affect the back-propagation in anyhow; it is only a collection of early predictions over data that the model has never seen.

Nevertheless, it helps to visualize the deep learning model's generalization capability, more specifically, to visualize whether it is suffering from the so-called overfitting (or underfitting). In general terms, it brings ideas about which actions may take place to prune the autoencoder's architecture in further trials of this task.

2.4 Metrics in SCA

Guessing entropy (GE) It is one the most well-known metrics in SCA; it has been studied in [24, 25]. GE defines the average rank of key-value $k[\cdot]$ over all the key hypotheses k. A GE vector $\vec{g}$ is built from consecutive trials of attacks with a subset of traces from the attack set.

We evaluate the attack effectiveness by taking the average position of $k[\cdot]$ in $\vec{g}$ given a number of traces. In that sense, a GE permanently equal to position one is considered a successful attack; nevertheless, any attack showing a downtrend is a potential threat.

Signal-to-Noise Ratio (SNR) It defined in [26] and it quantifies the amount of exploitable information; the higher the SNR is, the easier it is to detect the exploitable information [19]. Moreover, it helps to emphasize where, in the leakage traces, are those points that contain relevant information.

Along with SNR, Correlation Power Analysis (CPA) [27] can also be used to point out exploitable information; its magnitude's value is more significant than SNR, being convenient in situations where SNR is not than discriminative emphasizing the PoIs. Works such as [28] uses these two metrics to contrast them with automatic feature selection of a deep learning model.

In Eq. (8), f_t corresponds to a noisy observation of the leakage signal f at time sample t. $Z^{'}$ is the value to classify f, namely, group the leakage traces according to a value or a function, e.g. $Z^{'}=\mathcal{Z}[i]$ (from Eq. (9)).

$$\begin{aligned} &\text{SNR} = \frac{\mathbf{Var}[\mathbf{E}[f_t|Z^{'}]]}{\mathbf{E}[\mathbf{Var}[f_t|Z^{'}]]} \\ &\text{where:} \\ &\mathbf{Var} \quad \text{is the variance, and} \\ &\mathbf{E} \quad \text{is the expected value} \end{aligned} \tag{8}$$

A drawback of using SNR to detect PoIs becomes noticeable with the presence of countermeasures. One of those that disturb the most SNR's result is desynchronization; it randomly scatters $Z^{'}$ over time sample t. In such a case, a previous step like re-synchronizing the traces could reduce this pitfall's effect.

2.5 Visualization techniques

Besides metrics, there exist also feature selection visualization techniques to help in deep learning model interpretability. Beyond that goal, those techniques are also used to pin down the time points where the leakage is available in the algorithm's execution, which is necessary for a leakage assessment task [29–31]. This means, if the model is considering those, actual leakage points, it is learning the relevant information.

When a deep learning model is trained, it evaluates what neurons influence the most in order to generate an efficient classification. Those kernels get significant values for their weights. The technique tracks down those neurons, identifying which trace's sample points the network is taking as the features to learn from.

In simple terms, visualization techniques plot the most relevant feature from the input signal used to update kernels' elements. So, these techniques' typical application suggests to compare with metrics plots and visualize how similar they are. In Sect. 7, we show results when using these techniques and contrasting their plots against metrics plots to look for similarities.

In [28,30,32,33] the authors have discussed visualization techniques to use in SCA evaluations. Those techniques are mostly based on input activation gradient to characterize the automated feature selection, we named them here gradient visualization techniques.

There are two drawbacks of using those visualization techniques; (1) they aim to evaluate how the whole network performs in the feature selection. In our case, it would be the encoder and the decoder, and we are just interested in the former. (2) some of those techniques require a model's loss function as a starting

point; they evaluate the leakage traces' prediction using that loss function. An encoder does not have a loss function, so we cannot apply such a visualization technique.

From all the available techniques, weigh visualization and heatmap suit the task of visualizing autoencoders' automated feature selection performance. Weight visualization aims to visualize the convolutional part of a learning model [34], and we use convolutions in our autoencoder architecture. At the same time, the heatmap visualizes each convolutional layer of the convolutional part [12]. We use these two to contrast them with SNR to look for similarities.

2.6 Evaluation phases of profiling attacks

The procedure we propose in this paper aims to evaluate profiling attacks. As mentioned in Sect. 2, an adversary (or evaluator) identifies a target device whose firmware contains a secret aimed to be exploited. To train a *learning model*[3], the adversary gets an access to a clone from the target device.

While the clone device executes the target algorithm, the adversary draws a set of traces $\mathcal{P}$ from the device using equipment to measure its power consumption. Using those traces, he/she trains the learning model (profiling phase).

To attack the device, the adversary composes a set of traces $\mathcal{A}$ from the target device (normally $|\mathcal{A}| << |\mathcal{P}|$), the profiling learning model uses $\mathcal{A}$ to output the guessing entropy. The Fig. 3(a) shows the mentioned phases of the profiled SCA process.

In this evaluation process, the only feedback the evaluator has is the guessing entropy at the end of the process. He/she could do another iteration to look for an improvement in the result. Nonetheless, since it is a single learning model-based evaluation, it ends up being a process of modifying the learning model, expecting an improvement.

Overall, what we propose is a pre-processing phase based on autoencoders (see Fig. 3(b)). In this phase, we also train a learning model, which in our case is an autoencoder based on dilated convolutions. However, this learning model is trained using unsupervised learning, meaning we do not need to have information about the sensitive value aimed to attack.

This learning model aims to remove the information not related to sensitive values. Besides, it also removes any other aspect that could affect the metric's result at the end of the process (e.g., desynchronization).

In this sense, the evaluator has more than one feedback. He/she can evaluate the traces after being pre-processed for the learning model (we call it assets evaluations) before proceeding to the profiling phase.

The metrics used in those evaluations are, in a sense, up to the evaluator. He/she has to decide which metrics best fit the purpose. In this paper, one

[3] In general terms, there are different learning models regarding the attack technique used. For example, template attack, machine learning attack, deep learning-based attacks

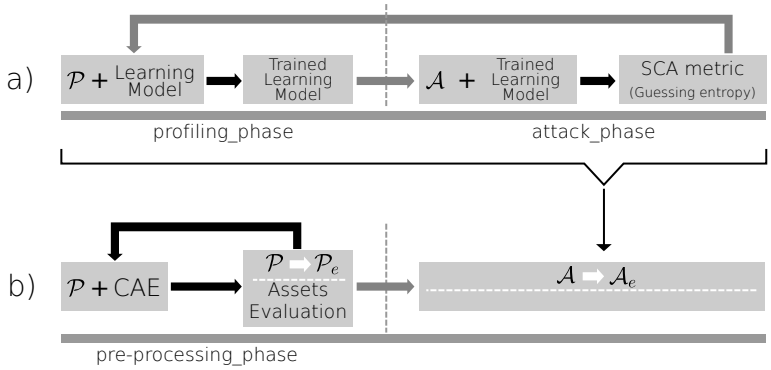


Fig. 3. Profiling SCA evaluation processes, a) is a usual profiled SCA process, b) is the profiled SCA process derived from the procedure proposed in this work. $\mathcal{P}$ is the traces dataset to perform the profiling phase, $\mathcal{A}$ is the traces dataset to perform the attack phase. From our proposal, two datasets derived from the previous become available; $\mathcal{P}_e$ is encoder version of $\mathcal{P}$, while $\mathcal{A}_e$ is the encoder version of $\mathcal{A}$; both after having been processed by the encoder.

of those metrics we suggest to use is SNR because it gives us the amount of exploitable information gained (or lost).

The evaluator uses $\mathcal{P}$ to train the autoencoder; after, he/she uses the autoencoder to build a compressed version $\mathcal{P}_e$ (one of the assets aimed for evaluation) of $\mathcal{P}$.

Once the pre-processing phase is done, the evaluator conducts the two following phases using $\mathcal{P}_e$ for the profiling phase and $\mathcal{A}_e$ for the attack phase (which is a compressed version of $\mathcal{A}$).

When the evaluator observes the metric's results, he can decide to continue or perform another iteration (see Sect. 5 for more details). This information helps him/her to reduce the uncertainly in the final guessing entropy result.

3 Related works

The closest work that we can compare with is [10], where authors present a denoising strategy as their main contribution. Still, it has been specifically designed to fulfill their goals. We have defined our procedure for more general autoencoder SCA evaluation, and we argue that it could be used on top of the denoising strategy from [10].

An autoencoder-based SCA evaluation has more assets than commonly used profiling SCA evaluations. We take this as motivation to define a procedure that uses this type of unsupervised learning models. About this latter claim, we have also noticed that unsupervised learning models in SCA have not been studied extensively; still, there are other recent works on this such as [9, 35].

Since this is the introduction of our procedure, we have not included some new trends about using transverse methodologies such as six-sigma and design of experiments [36,37], like some other works ([7,11,38]), where authors presented how transverse methodologies could reduce the uncertainty in SCA. Nevertheless, we concur that those methodologies could improve this first version of our procedure, and we will consider them for future works.

To our best knowledge, there are no previous works discussing criteria for SCA evaluation using autoencoder's architectures. Nevertheless, since this type of deep learning architecture shares several aspects with CNNs; the works explaining criteria to build an effective CNN architecture for SCA are useful for CAE as well [11,16].

Recent papers explain how the convolutions of a CNN model should take sensitive values. Some others introduced new techniques to build a convolutional block that ensures using features that represent sensitive values in its operation [12,13,21].

Our autoencoder proposal extends the criteria from our previous work [13] where we introduce the usage of dilated convolution as a way to improve the performance of the deep learning models based on convolutions for side-channel attacks.

4 ASCAD dataset

We chose this dataset since it contains the traces of implementations protected by a masking countermeasure, which also feature desynchronization.

ASCAD dataset was introduced in [11]. Leakage signals were collected from an Atmega8515 8-bit microcontroller. The algorithm is AES-128 [39] software implementation and the masking [20] countermeasure was used to protect it against first-order leakage[4].

During the acquisition process, an oscilloscope with the EM sensor sampled the signal at 2 GS/s. Conveniently, the storage points are a trim of the leakage signals, including only the relevant section of the cryptographic algorithm's execution, specifically the third masked Sbox in the first round.

Since this dataset was conceived to conduct SCA using neural networks, the file's structure allocates traces into two groups; *profiling_traces* contains traces to perform the profiling phase and *attack_traces*, which contains traces to perform the attack phase.

The dataset has two versions, collected traces with fixed key encryption and collected traces with random key encryption; due to these key characteristics, ASCAD random version represents a challenging way to conduct a profiled attack. Table 1 and Table 2 contain a summary of main characteristics from these two version.

Traces can be desynchronized by applying a threshold (N) that moves traces around the x-axis. The typical values to perform the benchmarking are N=0,

[4] This dataset is publicly available at https://github.com/ANSSI-FR/ASCAD

Table 1. Values of the training stage for synchronized traces

ASCAD Fixed Key version	
Profiling traces	50 000
Attack traces	10 000
Trace length	700
Key characteristic	Fixed

N=50, and N=100. For the first experiment, we have used ASCAD fixed key N=0; further experiments include the analysis over ASCAD Random key with N=0 and N=100; we concurred on using N=100 evidence enough the feasibility as well for N=50.

Table 2. Values of the training stage for synchronized traces

ASCAD Random Key version	
Profiling traces	200 000
Attack traces	100 000
Trace length	1 400
Key characteristic	Random

These datasets have no first-order leakage unless the masks leak. Sensitive value has the model represented by the Eq. (9); leakage traces are labeled using $\mathcal{Z}$ value. The byte that is intended to exploit is the third value ($i = 3$). The p represents the plaintext, and k is the possible key hypothesis, while $p[\cdot]$ and $k[\cdot]$ represent one single byte of the plaintext and key, respectively.

$$\mathcal{Z}[i] = \text{Sbox}[p[i] \oplus k[i]] \tag{9}$$

For deep learning training purposes, all the samples were standardized and normalized between 0 and 1 to accelerate the learning process [23]. We have used Keras [40] as a backend of Tensorflow [41] to build and train all the deep learning models. Details of the architecture used in each experiment are included in Sect. 7.

5 Autoencoder-based SCA procedure

This section explains our proposed procedure that allows us to evaluate our autoencoder architecture's effectiveness. We framed it with three tasks, namely (1) autoencoder training, (2) profiling dataset compression, (3) key classification. Those tasks would be commonly done by an evaluator working with an autoencoder for SCA (of course, it is skipping the inclusion of some other crucial tasks, e.g., acquisition setup).

The key classification task is here named as such considering it as an SCA specific issue; see Fig. 4 where this task is depicted as a usual profiled attack. Nevertheless, other scenarios where the goal is not key recovery but some other classification problems, or even using another supervised learning like classical template attacks, could also be considered.

The aim of dividing the procedure into three tasks is generic. We identified all the assets and grouped them into tasks that the evaluator does sequentially. We evaluate each task's asset using already defined metrics, quantifying their reliability before that asset goes for being used in further tasks. In this sense, we can compare them between different SCA evaluation trials, making early hypotheses about what could be changed to outperform the previous one.

By structuring a procedure as we did in this paper, we prepare the road for further steps on this perspective, like combining the tasks with crosswise methodologies similar to the one used in [7, 38]. Since we consider this as a possible improvement, we set this up for future works.

The experimental result section (see Sect. 7) is an example of our procedure; we used it to compare the effectiveness of two autoencoders' architectures (N-CAE and our proposal D-CAE).

Fig. 4 illustrates the overall picture of the procedure. As preparation, we set up the procedure's inputs, which are the original traces dataset and the deep learning architectures.

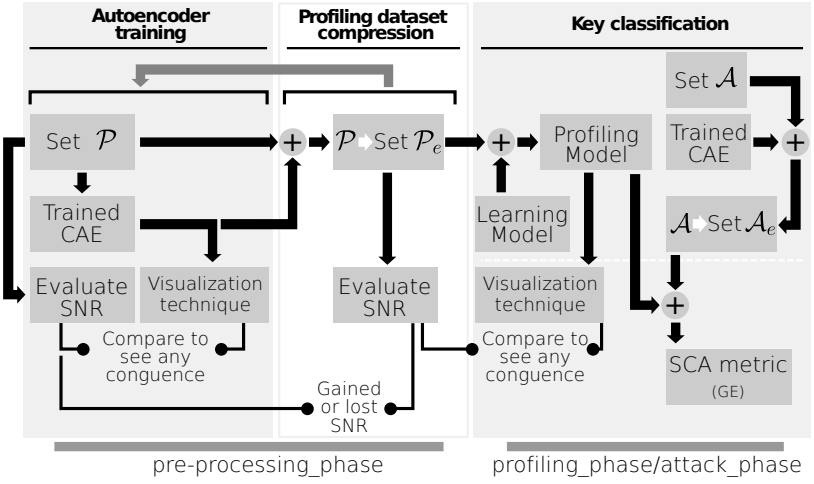


Fig. 4. Evaluation procedure illustration, each task has at least one metric to evaluate its output. Feature selection visualization techniques are used to contrast metrics' results graphically.

Table 3. Input and output assets of Autoencoder training task

Input/output assets	
Inputs	**Outputs**
1 Profiling traces dataset	Trained autoencoder
2 Untrained autoencoder	

5.1 Autoencoder training task

We seek a fine-tuned autoencoder model, so we should find ways to evaluate its effectiveness before the model goes to further tasks. The matter here is to evaluate the autoencoder as a whole and the autoencoder's minimal parts (e.g., the encoder).

MSE:

The autoencoder training task's primary concern is the latent space; by now, we only have information from the autoencoder's training process. As we said, an encoder should figure out the function that correctly estimates the data's distribution (leakage traces). How representative is that estimation reflects in the quality of the latent space. The decoder (the second part of the autoencoder) would do the opposite job in rebuilding what the encoder condensed.

Notice, our interest is to measure the differences between signals at the autoencoder's input with those at the autoencoder's output. It corresponds to an evaluation of the two functions (i.e., encoder and decoder), which gives us an overall idea of how representative the latent space is.

From that point of view, we use the output of the MSE to observe how the autoencoder manages to fit the leakage traces. If there is overfitting (or underfitting), it represents that the autoencoder was unable to find a function that correctly estimates the data.

For explanatory purposes, let us assume a hypothetical scenario where an autoencoder struggles to fit the data correctly, meaning it has problems in figuring out the data distribution. In straightforward terms, this ends up with a latent space that poorly represents that data, so that compressed traces datasets will be arguable useless. The fact of having a well-tuned model reduces uncertainty in the whole process.

Nevertheless, we do not forget the autoencoder is aimed for SCA evaluation in further tasks. Let us go back to the learning process; if the training and validation collide, that does not necessarily mean we have a useful model for conducting the SCA evaluation.

The autoencoders are filters, meaning that we can also damage the leakage in the process because the model just took the wrong points to build the latent space. That is why evaluate aspects such as; type of convolution used, feature selection performance, and learning process have crucial importance.

Remark 1: MSE should be considered just for determining whether the autoencoder is doing well in fitting the data. On the other hand, weight visualization might evidence whether the model is feasible for SCA.

SNR:

SNR is meaningful for the autoencoder training task since it contrasts the amount of noise we removed from the original dataset. In other words, we can depict a before and after of the information related to side-channel.

According to this first task's purpose, we have two concerns; (1) where in x-axis the leakage points arise aimed to find similarities with the visualization technique used in this first task. It will give us information about the leakage trace's features taken as relevant for the classification. (2) SNR's magnitude will be required for the second task of the procedure; we use it to contrast the amount of relevant information we gained or lost.

The SNR's application is up to the evaluator. For example, if the evaluator wants to contrast the mask leakage's exploitable information, he/she defines an SNR over this sensitive value. Similarly, for any sensitive value function of his/her interest. Furthermore, the chosen SNR criterion applies to the rest of the assets evaluated using this metric.

Remark 2: SNR in this task fulfill two purposes; emphasize PoIs to contrast with feature selection visualization technique. Then, in the profiling dataset compression task, it contrasts with the SNR from the compressed dataset to observe the amount of exploitable information remaining there.

Weight visualization:

We have observed that most techniques rely on the deep learning model's predictions; they require a loss function usually placed at the end of the model. That would not be a problem if we are pretending to evaluate the whole autoencoder using the visualization technique; however, we are not.

The visualization technique should only evaluate the encoder part because it is the one that has a direct connection with the latent space. The encoder does not have a loss function at its end (but rather the decoder).

As we previously mentioned, weight visualization is suitable because it does not require computing the prediction at the end of the model, which implies having a loss function.

Moreover, weight visualization was designed to visualize the convolutional part, being meaningful for us because our convolution-based model

Nevertheless, this technique gives plots that do not match the actual length of the leakage traces. It is because the x-axis of a weight visualization corresponds to the output of a convolutional block. It is arguable one of the less intuitive feature selection visualization because of. The evaluator has to look for similarities when comparing with plots produced by the mentioned metrics.

Here, we consider visualizing features taken by the encoder, from the first fully-connected layer after the flatten vector to the encoder's input. In this sense, we visualize the encoder as a convolutional neural network, ending up with plots like in Fig. 6. From the perspective of our contribution, we are evaluating how beneficial the chosen convolution is to build the autoencoder's architecture.

Table 4. Input and output assets of Profiling dataset compression task

Input/output assets		
	Inputs	**Outputs**
1	Trained autoencoder	Compressed profiling traces dataset

5.2 Profiling dataset compression task

In this task, the encoder is used to build compressed leakage traces from the original profiling dataset. The SNR's magnitude from this new dataset gives evidence about exploitable information remaining in the latent space.

We have to point out that an additional purpose for SNR in this second task is to visualize where PoIs fall in the x-axis of the compressed traces. This information will be used in the key classification task.

SNR is feasible in proving how well the encoder did in estimating the distribution. We measure that by looking at the amount of exploitable information in the compressed profiling dataset. Moreover, since we have a new x-axis arrangement, the SNR's limitation (when desynchronization is present) becomes a problem with less concern. The ability of the CNN models to deal with spatial transformation in the signal helped to this fact.

From this point, an evaluator might decide to conduct another iteration of the autoencoder training task if the SNR's magnitude improves doing it. An example of this can be drawn from Sect. 7, if the reader takes the two SNR plots from D-CAE and N-CAE architectures as two iterations of the first and second tasks.

Bear in mind that the SNR magnitude from latent space could be less than the original profiling traces. The result has a strong relationship with the amount of noise in the original traces and the encoder's performance; recall that this latter could have filtered out not only noise but relevant information.

Although we do not consider it here, there is a potential asset; we named it *clean* profiling traces dataset, not generated from the encoder but rather from the whole autoencoder model. This dataset could represent another source of experimentation so that we reserve that idea for further contributions.

5.3 Key classification task

This task is explicitly related to side-channel attacks. The strategies to conduct this task vary according to the approaches that an evaluator decides to apply, for example, template attack or machine/deep learning-based SCA. Moreover, he can also consider denoising strategies like the one defined in [10]. Notice, the task is strongly related to the classification experiment's purpose that the evaluator wants to fulfill with the side-channel information.

In any case, by having conducted the autoencoder SCA process with this evaluation procedure, we now have more information about the aspects that

Table 5. Input and output assets of key classification task

Input/output assets		
	Inputs	**Outputs**
1	Compressed profiling traces dataset	Key classification report

lead to a result in SCA obtained in this task. The global objective concerning SCA key recovery is to maximize the certainty about the secure implementation of a crypto algorithm, minimizing false positives.

In this opportunity, we decide to apply deep learning-based SCA, and we appeal to the approaches available now from [13], and [12] to define the deep learning architectures for the training and attack phases. The metric used to measure the effectiveness of the attack is the guessing entropy.

In this task, the selection of visualization techniques is flexible. We said that it is up to the evaluator to choose the strategy to conduct the side-channel attack, so it is the visualization technique that best suit that strategy. We chose gradient visualization because of our.

This technique gives information about what time samples influence the most in the classification; when those time samples are compared with those in the SNR plots, one can evaluate how well the neural network extracts the essential features.

Since the whole model has a loss function at its end, we can proceed with confidence; we used the SNR plots from profiling dataset compression task to compare with gradient visualization and see how each plot matches.

6 Dilated CAE criteria

This section discusses the criteria and constraints that we follow to build our autoencoder architecture proposal. Those criteria are intrinsically related to the model architecture's hyperparameter values; nevertheless, since there are many of them, we mention the most related to our proposal. Recall that our model is built upon dilated convolution criterion [13], so we are adding up criteria to use them in autoencoder-based SCA evaluation.

6.1 Convolutional block criterion

To an autoencoder based on convolutions applies all criteria related to CNN-based SCA. Usually, a convolutional block (primary parts of CNN) comprises convolution operation and pooling operation; both use kernels to conduct their operation, as well as stride and kernel length hyperparameters.

A criterion to fulfill is that each convolutional block implemented using pooling operation should be consistent on not overuse features from the convolution operation's feature map.

Meet this requirement implies to control the value of the stride hyperparameter. This value determines how many signal sample points a kernel moves between two successive convolutions. If such a value is small enough, the same feature could be used several times, nullifying how the convolution operation did the feature selection process. Usually, a pooling stride value (s_p) equal to a pooling kernel (l_{pk}) length is a possible solution ($s_p = l_{pk}$).

6.2 Encoder downsampling criterion

Even if a CAE's architecture is similar to a CNN, the learning model's purpose changes. The encoder aims to build a latent space, which means that it has to downsample the original traces to a less dimension space. It is called the downsampling process.

The downsampling process is achieved even by modifying convolution strides values or applying pooling operations, and since we are modifying the CAE's autoencoder using dilated convolutions, we have to know what the implications are.

When combining with dilated convolutions, the encoder can only use pooling operations, so we cannot modify stride value to change how many sample points the convolutions move.

This fact makes sense; stride hyperparameter could nullify the effect of dilatation rate hyperparameter. Take the case when a stride hyperparameter has a value big enough to move the same amount of zeros that the dilatation rate inserts into the kernel.

In any case, suffer from this constraint could depend on the software framework used to build the model[5]. We recommend checking the framework's documentation before choosing approaches (e.g., normal or dilated convolutions). Moreover, some frameworks allow the evaluator to customize the neural network layers, suggesting a possibility to overpass the constraint.

Regarding the upsampling process in which the decoder is in charge, it is done by transpose convolutions blocks [23]. In simple words, transpose convolutions are the inverse operation of convolutions. The common practice is to configure its hyperparameters with the same values as their counterparts. In these blocks, the evaluator can use stride value to perform the actual upsampling.

A final transpose convolutional block is used to map (one to one) the autoencoder's output with the original signal shape. So, its hyperparameters are set to 1.

6.3 Latent space criterion

The latent space dimension of an autoencoder is a crucial hyperparameter. This value implies that the encoder's job finding the data distribution function would be more challenging if the dimension is significantly small compared with the

[5] Keras-tensorflow, Caffe, Pytorch, and others

original input trace. In other words, the compression level becomes more demanding with fewer values of length than others with a dimension close to the original.

Theoretically, when the level of compression is demanding, the encoder tries hard to find the best feature so that the latent space would be best. To the evaluator, this would imply to redefine the encoder's architecture to achieve the desired level of compression. The encoder will most likely need to have more convolutional blocks (as well as more dense layers).

As the reader notices, this is another argument for comparing dilated convolution and normal convolution capability. If a convolutional block did well enough in the feature selection process, it implies that the whole autoencoder would need a fewer number of those blocks. It also implies that the model reduces its complexity and so the learning time.

Bear in mind, pushing a large trace into a small latent space could end with an encoder extremely overfitted, meaning not being able to generalize. Conduct a tradeoff could be a possible solution; for example, choose proportions of the latent space based on the signal original length, like some implementation of PCA or LDA. In [10] authors defined a mathematical model using the desirable output length.

7 Experimental result and discussions

In this section, we use our proposed evaluation procedure to conduct a profiling SCA evaluation using autoencoders. We have organized each experiment in sections named by the dataset used. Also, we recall that the evaluation procedure needs as primary inputs, the dataset, and the architecture used. So, we included in each section the respective architecture.

7.1 ASCAD fixed key $N=0$:

One could conclude that by applying our procedure with this dataset without noise, no desynchronization in traces, and with a fixed key for profiling and attacking, we overkilled the evaluation. However, this experiment should be taken as an example of two autoencoders used to perform a profiling SCA evaluation; it represents two iterations of our proposed procedure.

In the second experiment, we deal with a more challenging dataset (ASCAD random key $N=100$) and show how our procedure outperforms current results.

Autoencoder's architecture

This dataset has traces without desynchronization, which implies that SNR metrics would be feasible in showing PoIs in the first task. The criterion for SNR is to evaluate its magnitude related to the mask used for the masking countermeasure[6]. Moreover, since the key is constant in both groups of traces (i.e., profiling

[6] this is possible because, in this evaluation scenario, we have access to those values

and attack groups), the deep learning model used in key classification task will only deal with random values of the mask.

The autoencoder's architecture built upon dilated convolutions (D-CAE) is in Table 6. In the same Table 6 it is the normal convolution (N-CAE) with an equivalent architecture; they only differ in the value of the dilatation rate.

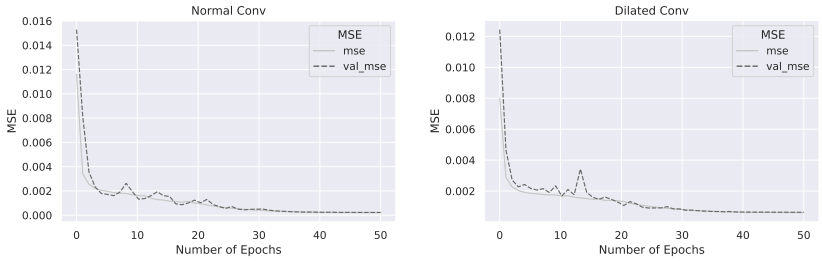
Table 6. D-CAE architecture for experiments using ASCAD fixed key. N-CAE has the same values with a dilatation rate of 1. AF stands for activation function, KI kernel initializer, BN Batch normalization, OCP is One cycle policy to tune the learning rate value through learning process and loss. Lenght of 560 represents 80% of input shape. Pooling type: average

Hyperparameter set	**Value**
Input shape	(700, 1)
Conv block 1	
Nº of kernels	4
Kernel length (l_k)	16
Dilation rate (dr)	6 (1 for N-CAE)
AF, KI	SeLU, He uniform
Regulatization	BN
Pooling	(l_{pk}=s_p=2)
Flatten	
Latent space dimension	560
Transpose conv block 1	
Nº of kernels	4
Kernel length (l_k)	16
AF, KI	SeLU, He uniform
Stride value	2
Transpose conv block 2 (mapping)	
Nº of kernels	1
Kernel length (l_k)	1
AF	Sigmoid
Model's output setup	
Optimizer	RMSprop
Loss	MSE
Learning rate	0.005 (and OCP)
Epochs	50
Batch size	256

Performing autoencoder training task:

After training N-CAE and D-CAE, we have the result of MSE in Fig. 5a and Fig. 5b. Both MSE from training and validation dataset through the epochs. According to validation MSE, the model presented a struggle in generalizing its

learning from 5^{th} to 30^{th} iterations. However, it shows the same tendency as training MSE until reaching the end of the learning process.



(a) (b)

Fig. 5. (a) Training and validation MSE of N-CAE : (b) Training and validation MSE of D-CAE

Notice, both autoencoders do well for the MSE metric. From there, we can ensure that both autoencoders figured out a data's distribution function. However, it does not necessarily mean that the coding made by those two encoders is related to the leakage of exploitable information.

The visualization techniques show in Fig. 6 how N-CAE did worst than D-CAE in the feature selection job. It is expecting that there will be some differences in information in the latent space.

To explain how Fig. 6 is interpreted, recall that we have the SNR metric, and for the autoencoder training task, we are interested in where the leakage points arise. The third and fourth plot in Fig. 6 corresponds to SNR of the unmasked sensitive values and the mask value, respectively (since we are in control of the input data that the DUT encrypts.)

Observe that the resemblance between D-CAE weight visualization and the third SNR plot is higher than N-CAE weight visualization. Beyond their x-axis do not match, some trending patterns can be depicted. Observe at the end of both plots; there are higher picks than in the rest of them. Then at their beginning, there are smaller picks but big enough to locate the interval. Contrary to the N-CAE weight visualization, its more significant picks are in the middle.

Performing profiling dataset compression task:

This task uses the encoder from the first task to build a compressed version of the original dataset's profiling group. We applied the SNR metrics to observe the amount of information that both autoencoders left over the latent space. As we previously said, in this task, we are interested in the SNR magnitude rather than the location of the picks.

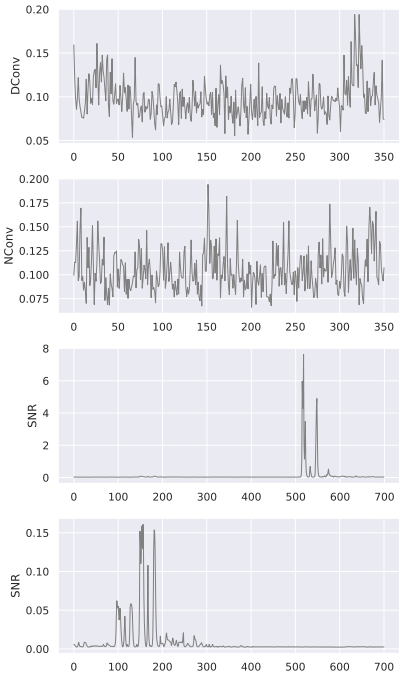


Fig. 6. Separated weigh visualization of N-CAE and D-CAE. The resemblance between CPA and D-CAE is higher than N-CAE

Fig. 7 shows the magnitude level of the mask value. The D-CAE SNR magnitude is higher than N-CAE, which reflects the performance of each autoencoder chosen in the first task. Nevertheless, the information from N-CAE represents useful information for the key classification task.

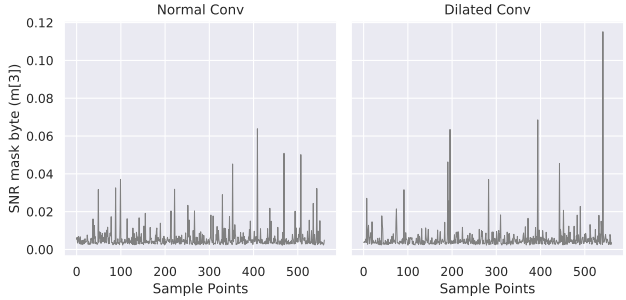


Fig. 7. SNR of mask byte. D-CAE SNR magnitude is higher than N-CAE

Performing key classification:

As we said, the evaluator decides the techniques to use for attacking the compressed traces. Here we apply dilated CNN-based SCA [13]. We based our choice by looking at the SNR plots in the second task, the mask leaks in different sample points, suggesting that we have to deal with irrelevant points in between.

As we explain in previous sections, dilated convolution is a good option in these scenarios. Moreover, another option is the approach described in [12]; the author uses a convolutional block with kernel length of 1, which also avoids overusing relevant and irrelevant points.

Do not discard the possibility of using a Multi-Layer Perceptron neural network; it is a worthy option because there is no desynchronization in the latent space; moreover, a convolutional NN could overkill the solution in some scenarios.

We first tried using the model's architecture suggested in [13], but as we expected, the model is too complex for these compressed traces. By applying an autoencoder against this dataset, we filtered out several features from the original signal, which implies to derive a new architecture, and most likely a less complex one.

The new model's architecture that we used is built upon the proposed architecture from [13], and it is shown in Table 7.

Obtaining less complex learning models for a compressed dataset is usual when using autoencoders. Notice we have also reduced the number of iterations of the training process (which also reduces the training time). Table 8 shows the differences in number of iteration and complexity[7] between our model and the model from [13].

Fig. 8 depicts the attack's guessing entropy. This result confirmes our assumptions about overkilling the evaluation; both autoencoders minimized the leakage magnitude. Moreover, since the dataset is very clean, useful features were filtered out. For this use case, N-CAE did worst in keeping the exploitable information than D-CAE; both guessing entropy curves reflect that.

By visualizing the gradient in Fig. 9a and Fig. 9b we ensure that this result relies mostly in the autoencoders we used and their differences in performance. By comparing gradient visualization and the SNR from the second task, we match the relevant points that the model in Table 7 is using as the most relevant ones.

D-CAE highlights points 100, 200, 300, 450 and 560 similar points are depicted in SNR plot. Similarly, the most relevant points of N-CAE are between 300 and 500; both gradient and SNR are similar over those points. Additionally, D-CAE gradient visualization suggests that the model struggles less with compressed traces than N-CAE, which is also a sign of using dilated convolutions.

As the reader might notice, our guessing entropy results require more traces than the work [13], such a result is expected. We have preprocessed the traces using an autoencoder, and since the dataset we used is one with a very few amount of noise, not only irrelevant features were filtered out.

[7] In number of parameters

Table 7. Dilated CNN to attack compresses traces of ASCAD fixed key. *OCP* was applied, pooling type: average

Hyperparameter set	**Value**
Input shape	(560, 1)
Convolutional block 1	
Nº of kernels	4
Kernel length (l_k)	10
Dilation rate (dr)	3
AF, KI	SeLU, He uniform
Regulatization	BN
Pooling	(l_{pk}=s_p=2)
Flatten	
Fully-Connected 1 2	
Nº of neurons	10
AF, KI	SeLU, He uniform
Fully-Connected 3	
Nº of neurons	256
AF	Softmax
Model's output setup	
Optimizer	Adam
Loss	Cross Entropy
Learning rate	0.005
Epochs	12
Batch size	100

Table 8. Number of training iteration and NN complexity of the model from [13] and our model.

	Model from [13]	New model
Epochs	50	12
Complexity	17 020	14 196

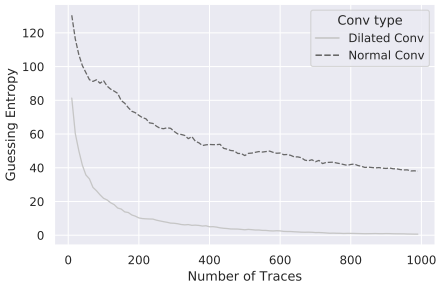
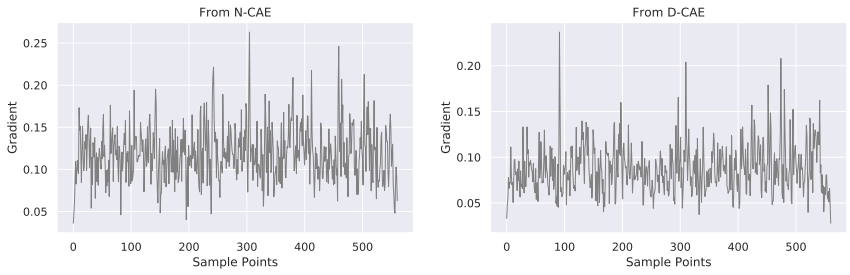


Fig. 8. Guessing entropy after attacking compressed traces from D-CAE and N-CAE



(a) (b)

Fig. 9. Gradient visualization, from the model in Table 7 using compressed traces from: (a) N-CAE and (b) D-CAE

The SNR from the first and second tasks emphasize the differences in magnitude about the exploitable information we filter away. Nevertheless, we have demonstrated that the procedure gives information about what we should expect in the process and that our autoencoder architecture proposal leads to better results. Additionally, We have reduced the training time, and the complexity in the model aimed to attack.

7.2 ASCAD random key $N=100$:

After addressing the autoencoder's feasibility with dilated convolutions, we now apply our procedure where the traces have more noise to deal with. ASCAD random key $N=100$ dataset represents a more challenging scenario; its traces were collected when the microcontroller was processing both random key and random mask values. Moreover, its traces are desynchronized, which also qualify as noise.

In advance, we would like to describe the goal we achieved in this use case. We designed the autoencoder to have a latent space with a similar dimension as the previous experiment. Because we looked for using a CNN with similar complexity as the one used in key classification task of the previous experiment, showing the feasibility of reusing attack architectures.

We ended up with a CNN architecture like Table 7 with slight changes in its first convolutional block and with an additional fully-connected layer. With these two changes, we achieved a better GE result for this dataset than [13].

Additionally, to enforce the evidence of reproducibility and repeatability, we performed our procedure six times; each time, we composed a subset of randomly chosen traces from the dataset; about 40 000 traces where approx. 37 000 are for training the D-CAE and approx. 3 000 for validating it.

Autoencoder's architecture

Table 9 shows the autoencoder's architectures. Due to the noise in this dataset, the D-CAE architecture has an additional convolutional block.

Table 9. D-CAE architecture for the experiments using ASCAD random key $N=100$

Hyperparameter set	**Value**
Input shape	(1400, 1)
Conv block 1, 2	
Nº of kernels	(32, 64)
Kernel length (l_k)	(64, 25)
Dilation rate (dr)	(3, 1)
AF, KI	SeLU, He uniform
Regulatization	BN
Pooling	(l_{pk}=s_p=2,=25)
Flatten	
Latent space dimension	560
Transpose conv block 1, 2	
Nº of kernels	(64, 32)
Kernel length (l_k)	(25, 64)
AF, KI	SeLU, He uniform
Stride value	(25, 2)
Transpose conv block 3 (mapping)	
Nº of kernels	1
Kernel length (l_k)	1
AF	Sigmoid
Model's output setup	
Optimizer	RMSprop
Loss	MSE
Learning rate	0.005 (OCP)
Epochs	100
Batch size	256

Performing autoencoder training task:

Fig. 10 shows the average of MSE after six trials. This result tells us that any random desynchronization in the leakage traces does not overfit (or underfit) the network. Nevertheless, it does not evidence that all the latent spaces from those six autoencoders have properly coded the exploitable information. We should wait until we have more evidence from other assets' evaluation.

Due to the heavy desynchronization in this dataset, the SNR magnitude does not depict large picks indicating PoIs. Nevertheless, it is still worthy of

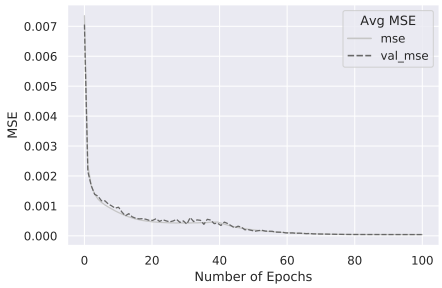


Fig. 10. Average of training and validation MSE from 6 experiments

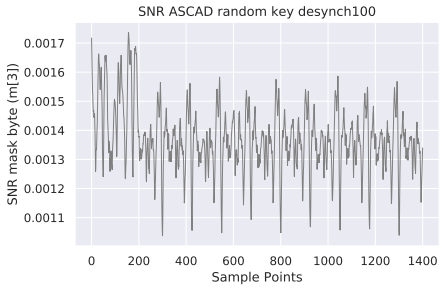


Fig. 11. SNR of ASCAD random key $N = 100$.

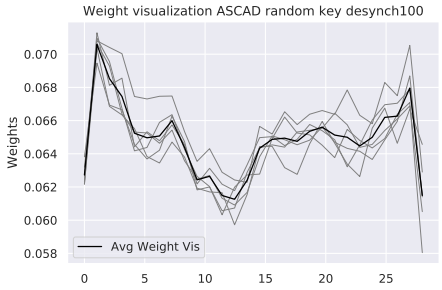


Fig. 12. Weight visualization from six trials, 2^{nd} Convolutional block output

performing it. It will be useful in the second task for comparison. Fig. 11 and Fig. 12 shows the SNR magnitude of the mask value and weight visualization to contrast (see A for additional visualization plots using heatmaps).

Performing profiling dataset compression task:

Fig 13 shows SNR magnitude from compressed dataset. We observed that the SNR magnitude from compressed datasets is bigger than SNR from the original

dataset. This result is because the encoder has already mitigated the effect of the desynchronization. Nevertheless, the evaluator might perform another iteration to look for changes in the SNR magnitude value if required.

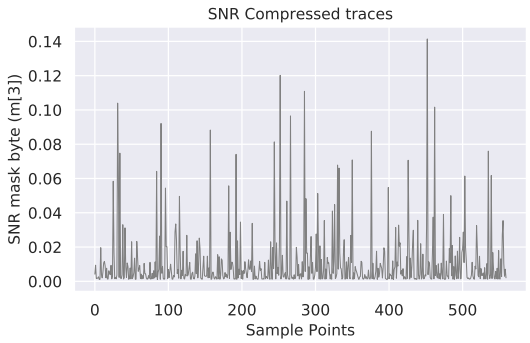


Fig. 13. SNR compressed traces from a random chosen autoencoders between all trials.

Performing key classification:

Table 10 summarizes the CNN's architecture to attacks the compressed traces. As we said, we modify the CNN architecture from 7 to came across a model with similar complexity. We applied the criterion of using kernel length of 1 because the separation between leaks in the compressed traces is overly small (comparing against the first experiment). Also, the leakage of masked sensitive values might share sample points with the mask used for those. Finally, the compressed traces are synchronized, meaning that the leakage position is constant, so that we do not need a long kernel receptive field.

Table 11 compares the complexity of the model used to attack ASCAD random key $N=100$ after applying our procedure using autoencoder and the result from [13].

Finally, Fig. 14a shows the GE result from the third task over 6 experiments. Our procedure outperforms the baseline GE from [13] attacking ASCAD random key $N=100$ (see B for plots of gradient visualization). Fig. 14b shows the confident interval of 99% from the experiment; less than 3000 traces are required to get a guessing entropy convergence of 0 with 99% confidence.

Notice, we have outliers in the GE trials. One explanation of this is that we have used the same CNN for the six compressed datasets. It is likely that a straightforward modification in the CNN architecture to adapt it to that specific compressed dataset would fix that outlier.

Table 10. CNN to attack compressed traces of ASCAD random key. *OCP* was applied, pooling type: average

Hyperparameter set	**Value**
Input shape	(560, 1)
Convolutional block 1	
Nº of kernels	4
Kernel length (l_k)	1
AF, KI	SeLU, He uniform
Regulatization	BN
Pooling	(l_{pk}=s_p=2)
Flatten	
Fully-Connected 1 2 3	
Nº of neurons	10
AF, KI	SeLU, He uniform
Fully-Connected 3	
Nº of neurons	256
AF	Softmax
Model's output setup	
Optimizer	Adam
Loss	Cross Entropy
Learning rate	0.005
Epochs	100
Batch size	100

Table 11. Differences in number of training iteration and in complexity of the model from [13] and our model. The attack target is ASCAD random key N=100

	Model from [13]	New model
Epochs	100	100
Complexity	96 975	14 270

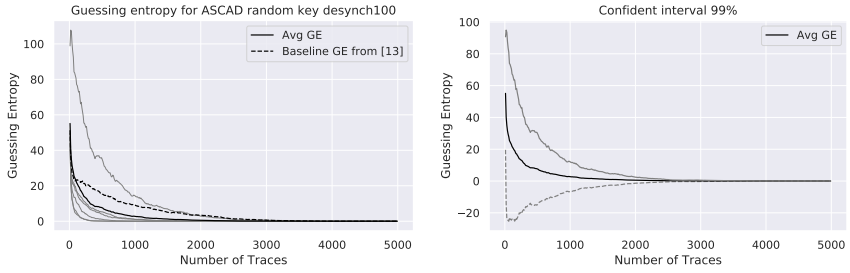
8 Conclusion and perspectives

In this paper, we have proposed an evaluation procedure based on autoencoders. To achieve proper effectiveness by using these unsupervised learning models, we have defined the criteria for designing an autoencoder based on dilated convolutions.

We have observed good performance in a dataset with heavy desynchronization. Nevertheless, we aim to use our procedure with more challenging datasets with traces of other microcontroller architectures.

Our procedure could be used for evaluating profiling SCA, mostly when the noise represents a significant obstacle for the evaluation. Moreover, it can be applied to compare new approaches in autoencoders architecture.

The procedure involves metrics and visualization techniques, helping to reduce the uncertainty of the side-channel evaluations. When the evaluation im-



(a) (b)

Fig. 14. (a) Guessing entropy from 6 experiments using our proposal procedure : (b) Confident interval of 99% from the six experiments.

plies fulfilling specific goals, some metrics could be replaced with others to evaluate the task's assets according to those goals.

For further publications, we will consider other metrics that might fit into the evaluation procedure like the one published in [42]. Moreover, we concur that our procedure could be improved by combining it with traverse methodologies like [7, 38].

References

1. S. Chari, J. R. Rao, P. Rohatgi, Template Attacks, in: B. S. Kaliski, ç. K. Koç, C. Paar (Eds.), Cryptographic Hardware and Embedded Systems - CHES 2002, Springer Berlin Heidelberg, Berlin, Heidelberg, 2003, pp. 13–28.
2. G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, J. Vandewalle, Machine Learning in Side-Channel Analysis: A First Study, Journal of Cryptographic Engineering 1 (4) (2011) 293.
3. T. Bartkewitz, K. Lemke-Rust, Efficient Template Attacks Based On Probabilistic Multi-class Support Vector Machines, in: International Conference on Smart Card Research and Advanced Applications, Springer, 2012, pp. 263–276.
4. Z. Martinasek, J. Hajny, L. Malina, Optimization Of Power Analysis Using Neural Network, in: International Conference on Smart Card Research and Advanced Applications, Springer, 2013, pp. 94–107.
5. H. Maghrebi, T. Portigliatti, E. Prouff, Breaking Cryptographic Implementations Using Deep Learning Techniques, 2016, pp. 3–26. doi:10.1007/978-3-319-49445-6_1.
6. E. Cagli, C. Dumas, E. Prouff, Convolutional Neural Networks with Data Augmentation Against Jitter-Based Countermeasures - Profiling Attacks Without Preprocessing, in: Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings, 2017, pp. 45–68.
7. U. Rioja, S. Paguada, L. Batina, I. Armendariz, The uncertainty of Side-Channel Analysis: A Way to Leverage from Heuristics, arXiv preprint arXiv:2006.12810 (2020).

8. S. Picek, A. Heuser, A. Jovic, L. Batina, A. Legay, The secrets of profiling for side-channel analysis: feature selection matters, IACR Cryptol. ePrint Arch. 2017 (2017) 1110.
9. K. Ramezanpour, P. Ampadu, W. Diehl, SCAUL: Power Side-Channel Analysis With Unsupervised Learning, IEEE Transactions on Computers 69 (11) (2020) 1626–1638. doi:10.1109/TC.2020.3013196.
10. L. Wu, S. Picek, Remove some noise: On Pre-processing of Side-channel Measurements with Autoencoders, IACR Transactions on Cryptographic Hardware and Embedded Systems (2020) 389–415.
11. E. Prouff, R. Strullu, R. Benadjila, E. Cagli, C. Canovas, Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database, IACR Cryptology ePrint Archive 2018 (2018) 53.
12. G. Zaid, L. Bossuet, A. Habrard, A. Venelli, Methodology for Efficient CNN Architectures in Profiling Attacks, IACR Transactions on Cryptographic Hardware and Embedded Systems Volume 2020 (2019) Issue 1–.
13. S. Paguada, I. Armendariz, The Forgotten Hyperparameter: Introducing Dilated Convolution for Boosting CNN-Based Side-Channel Attacks, in: International Conference on Applied Cryptography and Network Security, Springer, 2020, pp. 217–236.
14. C. Rechberger, E. Oswald, Practical Template Attacks, in: Information Security Applications, Springer, 2005.
15. M. O. Choudary, M. G. Kuhn, Efficient, Portable Template Attacks, IEEE Transactions on Information Forensics and Security 13 (2) (2018) 490–501.
16. J. Kim, S. Picek, A. Heuser, S. Bhasin, A. Hanjalic, Make Some Noise: Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis, IACR Transactions on Cryptographic Hardware and Embedded Systems (2019) 148–179.
17. L. Lerman, R. Poussier, O. Markowitch, F.-X. Standaert, Template attacks versus machine learning revisited and the curse of dimensionality in side-channel analysis: extended version, Journal of Cryptographic Engineering 8 (4) (2017) 301–313.
18. I. T. Jolliffe, J. Cadima, Principal Component Analysis: A Review and Recent Developments, Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences 374 (2065) (2016) 20150202.
19. S. Mangard, E. Oswald, T. Popp, Power analysis attacks: Revealing the secrets of smart cards, Vol. 31, Springer Science & Business Media, 2008.
20. J. Blömer, J. Guajardo, V. Krummel, Provably Secure Masking of AES, in: Selected Areas in Cryptography, Springer Berlin Heidelberg, 2004, pp. 69–83.
21. H. Maghrebi, Deep Learning based Side Channel Attacks in Practice, IACR Cryptology ePrint Archive 2019 (2019) 578.
22. B. Hettwer, S. Gehrer, T. Güneysu, Profiled Power Analysis Attacks Using Convolutional Neural Networks with Domain Knowledge, in: Selected Areas in Cryptography - SAC 2018 - 25th International Conference, Calgary, AB, Canada, August 15-17, 2018, Revised Selected Papers, 2018, pp. 479–498.
23. I. Goodfellow, Y. Bengio, A. Courville, Deep learning, MIT press, 2016.
24. F.-X. Standaert, T. G. Malkin, M. Yung, A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks, in: A. Joux (Ed.), Advances in Cryptology - EUROCRYPT 2009, Springer Berlin Heidelberg, Berlin, Heidelberg, 2009, pp. 443–461.
25. J. Massey, Guessing and Entropy, in: Proceedings of 1994 IEEE International Symposium on Information Theory, IEEE, 1994.

26. R. A. Fisher, On the mathematical foundations of theoretical statistics, Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character 222 (594-604) (1922) 309–368.
27. E. Brier, C. Clavier, F. Olivier, Correlation Power Analysis With a Leakage Model, in: International workshop on cryptographic hardware and embedded systems, Springer, 2004, pp. 16–29.
28. B. Timon, Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis, IACR Trans. Cryptogr. Hardw. Embed. Syst. 2019 (2) (2019) 107–131.
29. T. Schneider, A. Moradi, Leakage Assessment Methodology, in: T. Güneysu, H. Handschuh (Eds.), Cryptographic Hardware and Embedded Systems – CHES 2015, Springer Berlin Heidelberg, Berlin, Heidelberg, 2015, pp. 495–513.
30. F. Wegener, T. Moos, A. Moradi, DL-LA: Deep Learning Leakage Assessment: A modern roadmap for SCA evaluations, IACR Cryptol. ePrint Arch. 2019 (2019) 505.
31. O. Reparaz, B. Gierlichs, I. Verbauwhede, Fast Leakage Assessment, in: CHES, 2017.
32. L. Masure, C. Dumas, E. Prouff, Gradient visualization for general characterization in profiling attacks, in: International Workshop on Constructive Side-Channel Analysis and Secure Design, Springer, 2019, pp. 145–167.
33. G. Perin, B. Ege, L. Chmielewski, Neural Network Model Assessment for Side-Channel Analysis, IACR Cryptol. ePrint Arch. 2019 (2019) 722.
34. H. Bischof, A. Pinz, W. G. Kropatsch, Visualization Methods for Neural Networks, in: 11th IAPR International Conference on Pattern Recognition. Vol. II. Conference B: Pattern Recognition Methodology and Systems, Vol. 1, IEEE Computer Society, 1992, pp. 581–582.
35. G. Perin, L. Chmielewski, L. Batina, S. Picek, Keep it Unsupervised: Horizontal Attacks Meet Deep Learning, Cryptology ePrint Archive, Report 2020/891, https://eprint.iacr.org/2020/891 (2020).
36. C.-S. Cheng, S.-C. Lee, P.-W. Chen, K.-K. Huang, The Application of Design for Six Sigma on High Level Smart Phone Development, Journal of Quality 19 (2012) 117–136. doi:10.6220/joq.2012.19(2).02.
37. D. C. Montgomery, Design & Analysis of Experiments, John Wiley & Sons, Inc., USA, 2019.
38. S. Paguada, U. Rioja, I. Armendariz, Controlling the Deep Learning-Based Side-Channel Analysis: A Way to Leverage from Heuristics, in: International Conference on Applied Cryptography and Network Security, Springer, 2020, pp. 106–125.
39. J. Daemen, V. Rijmen, The Design of Rijndael, Springer-Verlag, Berlin, Heidelberg, 2002.
40. F. Chollet, et al., Keras, https://keras.io (2015).
41. M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems, software available from tensorflow.org (2015).
URL https://www.tensorflow.org/
42. J. Zhang, M. Zheng, J. Nan, H. Hu, N. Yu, A Novel Evaluation Metric for Deep Learning-based Side Channel Analysis and Its Extended Application to Imbalanced

Data, IACR Transactions on Cryptographic Hardware and Embedded Systems (2020) 73–96.

A Other visualization techniques

A.1 Heatmaps

Heatmap visualizes the kernel's mean of a convolutional layer (not to be confused with convolutional block). It aims to evaluate the kernels through their weights. Kernels' elements get activated in the convolutional operation when the network is confident with the chosen features that correspond with them.

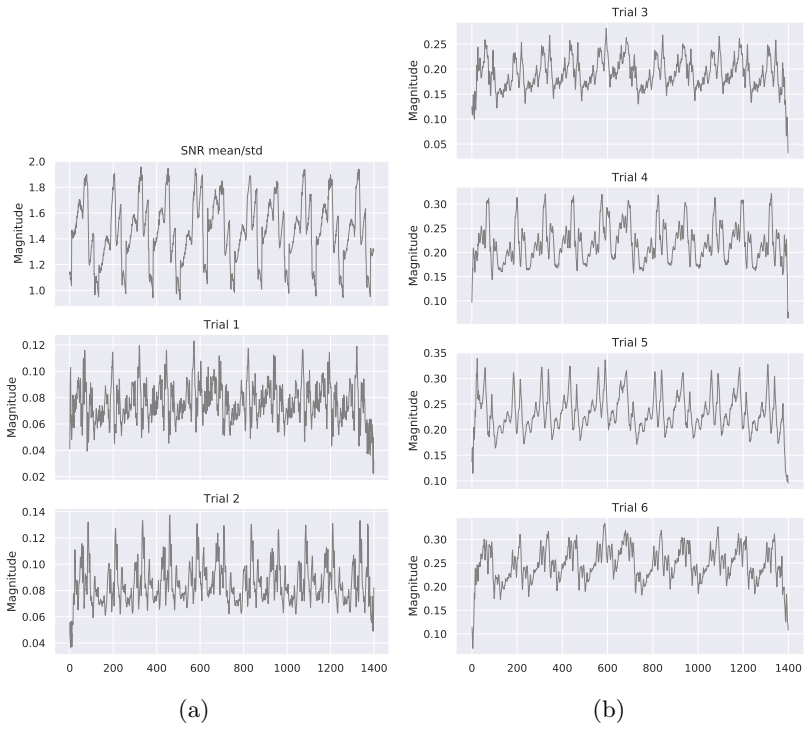


Fig. 15. Comparison between SNR: $\frac{\text{mean(profiling\_traces)}}{\text{std(profiling\_traces)}}$ and Heatmap from 1^{st} convolutional layer of all six experiments

These heatmap plots are contrasted with a more typical SNR application, meaning we apply it to measure both the noise level and the information not related to the side-channel (see Eq. (10)).

$$\text{SNR} = \frac{\text{mean}(\text{profiling\_traces})}{\text{std}(\text{profiling\_traces})} \tag{10}$$

Fig. 15a and Fig. 15b show heatmaps visualization of the trials in the second use case. Observe how the peaks in the SNR plot correspond with those in the first convolutional layer heatmap.

B Additional plots of the ASCAD $N=100$ experiment

Fig 16 shows the gradient visualization of the CNNs used to perform each one of the trials in the second experiment.

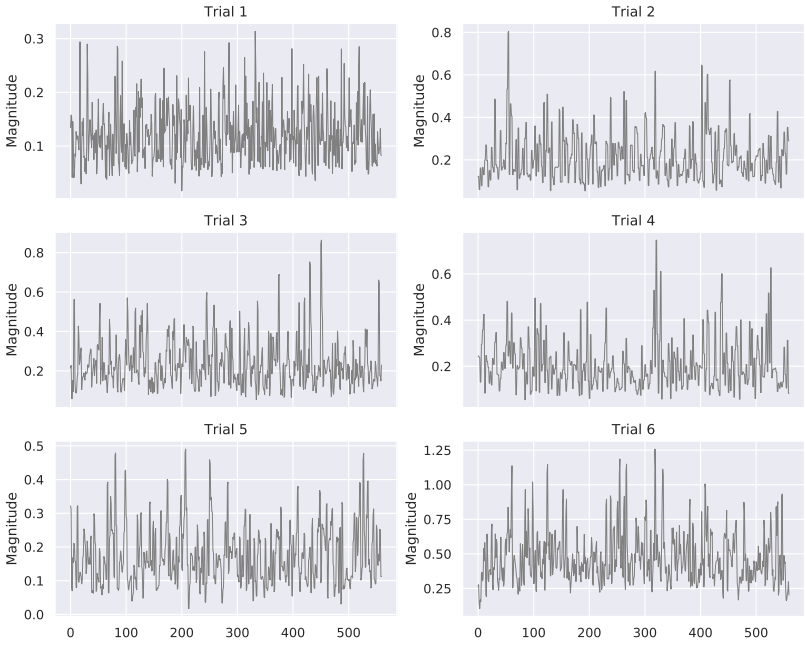


Fig. 16. CNN's gradient visualization of the six experiments