Remove Some Noise: On Pre-processing of Side-channel Measurements with Autoencoders

Lichao Wu and Stjepan Picek

Delft University of Technology, The Netherlands
l.wu-4@tudelft.nl, s.picek@tudelft.nl

Abstract. In the profiled side-channel analysis, deep learning-based techniques proved to be very successful even when attacking targets protected with countermeasures. Still, there is no guarantee that deep learning attacks will always succeed. Various countermeasures make attacks significantly more complicated, and those countermeasures can be further combined to make the attacks even more challenging. An intuitive solution to improve the performance of attacks would be to reduce the effect of countermeasures.
In this paper, we investigate whether we can consider certain types of hiding countermeasures as noise and then use a deep learning technique called the denoising autoencoder to remove that noise. We conduct a detailed analysis of five different types of noise and countermeasures either separately or combined and show that in all scenarios, denoising autoencoder improves the attack performance significantly.

Keywords: Side-channel analysis, Deep learning, Noise, Countermeasures, Denoising autoencoder

1 Introduction

Side-channel analysis (SCA) is a threat exploiting weaknesses in physical implementations of cryptographic algorithms rather than the algorithms themselves [MOP06]. During the execution of an algorithm, leakages like electromagnetic (EM) radiation [QS01] or power dissipation [KJJ99] can happen. Side-channel analysis can be divided into 1) direct attacks like single power analysis (SPA) and differential power analysis (DPA) [KJJ99], and 2) profiled attacks like template attack (TA) [CRR02] and supervised machine learning-based attacks [MPP16, PSK+18, CDP17, KPH+19]. In recent years, machine learning-based approaches and especially deep learning-based approaches proved to be a powerful option when conducting profiled SCA. While such attack methods actively threaten the security of cryptographic devices, there are still severe limitations. More precisely, attack methods commonly rely on the correlation characteristics of the signal, i.e., signal patterns that are related to the data being processed. Once the correlation degrades, attacks become less effective or even useless [BCO04].

In some cases, the low signal-to-noise ratio of the leakage increases the difficulty of identifying these patterns. Additionally, there are various countermeasures in both hardware and software that make the attacks more difficult. The countermeasures can be divided into two categories: masking and hiding. Masking splits the sensitive intermediate values into different shares to decrease the key dependency [CJRR99, BDF+17]. Hiding, on the other hand, aims at reducing the side-channel information by adding randomness to the leakage signals or by making it constant. There are several approaches to hiding. For example, the direct addition of noise [CCD00] or the design of dual-rail logic styles [TV03] are frequently considered options. Exploiting time-randomization is another alternative, e.g., by using Random Delay Interrupts (RDIs) [CK09] implemented in software and clock jitters in hardware. Still, the countermeasures (especially the hiding ones) are not without weaknesses. Regardless of what hiding approaches are used, we can treat their effects as noise due to their randomness. In other words, the ground truth of the traces always exists. If we can find a way to remove the noise (denoise) from the traces and recover the ground truth of the leakage, then the reconstructed traces could become more vulnerable to SCA.

While considering the countermeasures as noise and then removing that noise sounds like an intuitive approach, this is not an easy problem. The noise (both from the environment and countermeasures) is a part of a signal, and those two components cannot entirely be separated if we do not know their characterizations. Additionally, in realistic settings, we must consider the portability and the differences among various devices [BCH+19]. The combination of all these factors makes this problem very complicated, and to the best of our knowledge, there are no universal approaches on how to remove the effects of environmental noise and countermeasures.

Common approaches to remove/reduce noise are to use low-pass filters [WLL+18], conduct trace alignments [TGWC18], and various feature engineering methods [ZZY+15, PHJB19]. More recently, the SCA community started using deep learning techniques that make implicit feature selection and fight countermeasures [CDP17, KPH+19, ZBHV19]. While such techniques are useful, they are usually aimed against a single source of the noise. In cases when they can handle more sources of noise, they lack the interpretability of results. More precisely, in such cases, it is not clear at what point noise removal stops and attack starts (or even if there is such a distinction). We emphasize that being able to reduce the noise comprehensively could bring several advantages, like 1) understanding the attack techniques better, 2) understanding the noise better, and consequently, (hopefully) being able to design stronger countermeasures, and 3) ability to mount stronger/simpler attacks as there is no noise to consider.

In this paper, we propose a new approach to remove several common hiding countermeasures with a denoising autoencoder. Although the denoising autoencoder proved to be successful in removing the noise from an image [Gon16], as far as we are aware, this technique has not been applied to the side-channel domain to reduce the noise/countermeasures effect. We demonstrate the effectiveness of a convolutional denoising autoencoder in dealing with different types of noise and countermeasures separately, i.e., white noise, desynchronization, RDIs, clock jitters, and shuffling. Then, we make the problem more realistic by combining various types of noise and countermeasures with the traces and trying to denoise it with the same machine learning models. The results show that the denoising autoencoder is surprisingly efficient in removing the noise and countermeasures in all investigated situations. We emphasize that denoising autoencoder is not a technique to conduct the profiled attack, but to pre-process the measurements so that any attack strategy can be applied. Our approach is particularly powerful when considering the white-box scenarios, but we also discuss how one can use denoising autoencoders in black-box settings.

1.1 Related Work

The analysis of the leakage traces in the profiled SCA scenario can be seen as a classification problem where the goal of an attacker is to classify those traces based on the related data (i.e., the encryption key). The most powerful attack from the information-theoretic point of view is the template attack (TA) [CRR02]. Still, this attack can reach its full potential only if the attacker has an unbounded number of traces, and the noise follows the Gaussian distribution [LPB+15]. More recently, various machine learning techniques emerged as preferred options for cases where 1) the number of traces is either limited or very high, 2) the number of features is very high, 3) countermeasures are implemented, and 4) we cannot make assumptions about data distribution. First, the side-channel community showed the most interest in techniques like random forest [LMBM13, HPGM16] and support vector machines [HZ12, PHJ+17]. More recently, multilayer perceptron (MLP) [GHO15, PHJ+19] and convolutional neural networks (CNN) [MPP16, CDP17, KPH+19] emerged as the most potent approaches. Convolutional neural networks were demonstrated to be capable of coping with the random delay countermeasure due to their spatial invariance property [CDP17, KPH+19]. At the same time, the fully-connected layers in multilayer perceptron and convolutional neural networks are effective against masking countermeasures as they produce the effect of a higher-order attack (combining features) [BPS+18, KPH+19]. As far as we are aware, the only application of autoencoders for profiled SCA is made by Maghrebi et al., but there, the authors use it for classification and report a poor performance when compared to CNNs [MPP16].

1.2 Our Contributions

We consider how to denoise the side-channel traces with convolutional autoencoder (CAE), which to the best of our knowledge, has not been explored before in the side-channel domain. More precisely, we introduce a novel approach to remove the effect of countermeasures, and we propose:

1. A convolutional autoencoder architecture, which requires a limited number of traces to train and can denoise/remove the effect of various hiding countermeasures.
2. A methodology to recover the ground truth of the traces.
3. A technique to denoise measurements in black-box settings, where we use traces processed with other denoising techniques as a reference "clean" measurements to be used with denoising autoencoder.
4. A benchmark of the attack performance for popular denoising/signal processing techniques used in the side-channel domain.

To conduct experimental analysis, we consider five separate sources of noise or their combination. We investigate the performance of template attack, multilayer perceptron, and convolutional neural networks for the classification task (SCA attack) before and after the noise removal. Finally, we experiment with different datasets, having either fixed or random keys, to show the universality of our approach.

2 Background

Let k^* denote the fixed secret cryptographic key (byte), k any possible key hypothesis, and p plaintext. To guess the secret key, the attacker first needs to choose a leakage model $Y(p, k)$ (or Y when there is no ambiguity) depending on the key guess k and some known text p, which relates to the deterministic part of the leakage. The size of the keyspace equals $|K|$. For profiled attacks, the number of acquired traces in the profiling phase equals N, while the number of traces in the testing phase equals T. For the autoencoder, we denote its input as $\mathcal{X}$. The encoder part of an autoencoder is denoted as ϕ and the decoder part as ψ. Its latent space is denoted as $\mathcal{F}$. As for the training data, we refer to protected traces (with noise and countermeasures) as noisy traces while the unprotected traces are denoted as clean traces.

2.1 Profiled Side-channel Analysis

The profiled side-channel attacks represent the most powerful category of SCAs as we assume an attacker with access to an open (keys can be chosen/or are known by the attacker) clone device. Then, the attacker can use that clone device to obtain N measurements from it and construct a characterization model of the device's behavior. When launching an attack, the attacker then collects a few (T) traces from the attack device where the secret key is not known. By comparing the attack traces with the characterized model, the secret key can be revealed. Ideally, the secret key can be obtained with a single trace from the attack device ($T = 1$). Single trace attack is difficult in practice due to the effect of the noise, countermeasures, and a finite number of traces in the profiling phase (while we assume the attacker is not bounded in his power and he can collect any number of traces, that number represent a small fraction of all possible measurements).

Template Attack Template attack uses Bayes theorem to obtain predictions, dealing with multivariate probability distributions as the leakage over consecutive time samples is not independent [CRR02]. In the state-of-the-art, template attack relies mostly on a normal distribution. It consists of two phases: the offline phase during which the templates are built, and the online phase where the matching between the templates and unseen power leakage happens.

2.2 Neural Networks

A neural network is an interconnected assembly of simple processing elements, units or nodes, whose functionality is based on the biological process occurring in the brain [Gur14]. In general, a neural network consists of three blocks: an input layer, one or more hidden layers, and an output layer, whose processing ability is represented by the strength (weight) of the inter-unit connections, learning from a set of training patterns from the input layer. In the supervised machine learning paradigm, neural networks work in two phases: training and testing. In the training phrases, the goal is to learn a function f, s.t. $f : \mathcal{X} \rightarrow \mathcal{Y}$, given a training set of N pairs (x_i, y_i). Here, for each example (trace) x, there is a corresponding label y, where $y \in \mathcal{Y}$. Once the function f is obtained, the testing phase starts with the goal to predict the labels for new, previously unseen examples.

Multilayer Perceptron The multilayer perceptron (MLP) [GD98] is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP consists of multiple layers of nodes in a directed graph, where each layer is fully connected to the next one, and training of the network is done with the backpropagation algorithm.

Convolutional Neural Networks CNN commonly consists of three types of layers: convolutional layers, pooling layers, and fully-connected layers. Each layer of a network transforms one volume of activation functions to another through a differentiable function. Convolution layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. Pooling decrease the number of extracted features by performing a down-sampling operation along the spatial dimensions. The fully-connected layer computes either the hidden activations or the class scores. To avoid the overfitting, batch normalization layer, which normalizes the input layer by adjusting and scaling the activations is commonly added to the network.

2.3 Guessing Entropy

To assess the performance of the attacker, one uses a metric denoting the number of measurements required to obtain the secret key. A typical example of such a metric is guessing entropy (GE) [SMY09]. GE represents the average number of key candidates an adversary needs to test to reveal the secret key after conducting a side-channel analysis. In particular, given T amount of samples in the attacking phase, an attack outputs a key guessing vector $g = [g1, g2, ..., g_{|K|}]$ in decreasing order of probability. Then, guessing entropy is the average position of k^* in g over several experiments.

2.4 The ASCAD Dataset

The ASCAD dataset was introduced by Prouff et al. to provide a benchmark to evaluate machine learning techniques in the context of side-channel attacks [BPS+18]. An AT-Mega8515 device was used to record the emitted EM radiation during the execution of a software AES implementation protected by known masks. All traces were captured with a sensor attached to an oscilloscope sampling at 2 GS/s.

There are two datasets recorded in different conditions: fixed key encryption and random key encryption. For the data with fixed key encryption, the dataset provided separate HDF5 files with different synchronization level: the traces in the ASCAD.h5 file are time-aligned in a prepossessing step, whereas the traces in ASCAD_desync50.h5 and ASCAD_desync100.h5 have been shifted with a maximum jitters window of respectively 50 and 100 samples [BPS+18]. Each file contains 60 000 EM traces (50 000 training / cross-validation traces and 10 000 test traces). Each trace consists of 700 points of interest (POI). For the data with random key encryption, there are 200 000 traces in the profiling dataset that is provided to train the (deep) neural network models. A 100 000 traces attack dataset is used to check the performance of the trained models after the profiling phase. A window of 1 400 points of interest is extracted around the leaking spot. Throughout the paper, we use the raw traces and the pre-selected window of relevant samples per trace corresponding to masked S-box for $i = 3$. As a leakage model, we use the unprotected S-box output, i.e.:

$$Y(i) = \texttt{S-box}[(p[i] \oplus k[i])]. \tag{1}$$

3 Denoising with Convolutional Autoencoder

3.1 Autoencoders

Autoencoders were first introduced in the 1980s by Hinton and the PDP group [RHW85] to address the problem of "backpropagation without a teacher". Unlike other neural network architectures that map the relationship between the inputs and the labels, an autoencoder transforms inputs into outputs with the least possible amount of distortion [Bal12]. Benefits from its unsupervised learning characteristic, an autoencoder is applicable in settings like data compression [TSCH17], anomaly detection [AC15], and image recovery [Gon16].

An autoencoder consists of two parts: encoder (ϕ) and decoder (ψ). The goal of the encoder is to transfer the input to its latent space $\mathcal{F}$, i.e., $\phi : \mathcal{X} \to \mathcal{F}$. The decoder, on the other hand, reconstructs the input from the latent space, which is equivalent to $\psi : \mathcal{F} \to \mathcal{X}$. When training an autoencoder, the goal is to minimize the distortion when transferring the input to the output (Eq. (2)), i.e., the most representative input features are forced to be kept in the smallest layer in the network:

$$\phi, \psi = \underset{\phi,\psi}{\arg\min} \, \|X - (\psi \circ \phi)X\|^2. \tag{2}$$

When applying the autoencoder for the denoising purpose, the input and output are not identical but represented by noisy-clean data pairs. A similar idea can also be applied to remove the countermeasures from the leakage traces. A well-trained denoising autoencoder can keep the most representative information (i.e., leakage trace value) in its bottleneck while neglecting the less important features such as random noise. Since the original trace (without noise) can be recovered by feeding noisy traces to the input of the autoencoder, one can expect that the attack efficiency will be dramatically improved with the recovered traces.

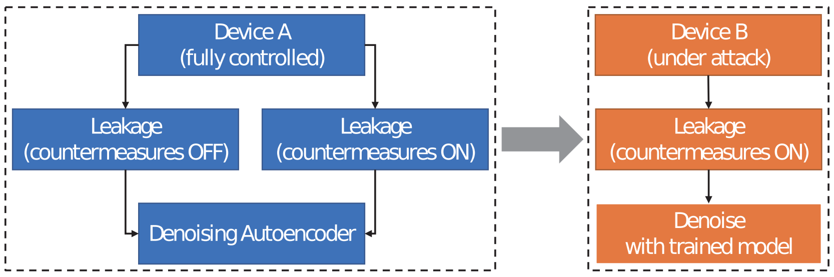


Figure 1: Denoising strategy.

3.2 Denoising Strategy

As discussed in Section 3.1, noisy-clean trace pairs are required to train a denoising autoencoder. Inspired by the profiled side-channel analysis method, we devise an ideal denoising strategy shown in Figure 1.

We assume an attacker with full control of a device (Device A). Specifically, he can enable/disable the implemented countermeasures. To attack the real devices with countermeasures enabled (Device B), he first acquires traces with and without countermeasures from Device A to build the training sets. Then the attacker uses these traces to train the denoising autoencoder. Once the training process is finished, the trained model can be used to pre-process the leakage traces obtained from Device B. Finally, with "clean" (or, at least, cleaner) traces reconstructed by the denoising autoencoder, an attacker can eventually retrieve the secret information with less effort. Still, for practical attack scenarios, the biggest challenge for this strategy is *how* to obtain clean traces:

1. **White-box setting**. When considering software implementations, an attacker might have a device on which he can modify the code, and then he can turn off the countermeasures. For hardware implementations, the scenario is more complicated. Let us consider a cryptographic core on a SoC: an attacker might be able to turn off countermeasures by setting cryptographic core control registers, if he has run-time control on the main processor of the SoC. For signature schemes, sometimes the verification procedure on the public key does not include countermeasures while the signature generation does. This means that verification can be used for learning. Finally, during EMVCo and Common Criteria evaluations, it is common to be able to turn off some (or all) countermeasures.
2. **Black-box setting**. Here, the attacker is not able to obtain clean measurements, but he can apply other denoising techniques like averaging or spectral analysis to reduce the influence of noise or countermeasures. Then, he can use noisy/less noisy pairs to train a denoising autoencoder. While this approach is not realistic for all countermeasures, we show it works for several of them. We see that even if we train autoencoder for different types of noise at the same time, it is successful when applied to settings that do not use all the noise types.

The application of denoising autoencoders is intuitive if we consider the white-box setting. Still, we also see its potential in the black-box setting. Let us consider the case of Gaussian noise. There, with traditional techniques like averaging, we would reduce the initial pool of traces to obtain averaged measurements. Depending on the number of traces in the averaging process, the final set of traces may not be enough for a successful attack. Now, let us consider denoising autoencoders. Our experiments show it works well even when trained with a small number of traces, e.g., 10 000 traces. This means that we can use different techniques to clean the traces and then, use such obtained measurements for the denoising autoencoder. Later, we can use the autoencoder with the initial pool of traces, first to denoise them, and then to attack. As such, the denoised traces processed by the denoising autoencoder turn an impossible attack (from the perspective of guessing entropy with a limited number of traces) into a reality. From the attacker perspective, he can invest more effort to acquire limited numbers of clean traces and train a denoising autoencoder. Once the model is trained, the attacker can launch an attack with (in theory) an unlimited number of denoised traces obtained from denoising autoencoder. What is more, densoising autoencoder serves well as a generic denoiser technique.

3.3 Convolutional Autoencoder Architecture

An autoencoder can be implemented with different neural network architectures. The most common examples are the MLP-based autoencoder and convolutional autoencoder (CAE). We tested different MLP and CNN architectures and then selected the model that has the best performance in denoising all types of noise and countermeasures. As a result, we use the convolution layer as the basic element for denoising. To maximize the denoising ability of the proposed architecture, we tune the hyperparameters by evaluating the CAE performance toward different types of noise, and we select the one that has the best performance on average for all noise types[1]. The tuning range and selected hyperparameters are shown in Table 1. Specifically, the *SeLU* activation function is used to avoid vanishing and exploding gradient problems [KUMH17]. $He\ Uniform$, on the other hand, can improve the initialization of the weight [HZRS15].

Table 1: CAE hyperparameter tuning.

Hyperparameter	**Range**	**Selected**
Optimizer	Adam, RMSProb, SGD	Adam
Weight initialization	Uniform distribution, He uniform	He uniform
Activation function	Tanh, ReLU, SeLU	SeLU
Learning Rate	1e-5, 5e-5, 1e-4, 5e-4	1e-4
Batch size	32, 64, 128, 256	128
Epochs	30, 50, 70, 100, 200	100
Training sets	1 000, 5 000, 10 000, 20 000	10 000
Validation sets	2 000, 5 000	5 000

In terms of the autoencoder architecture, we observed that when dealing with trace desynchronization, an autoencoder with a shallow architecture can denoise the traces successfully. Still, when introducing other types of noise into the traces while keeping the same hyperparameters, such autoencoders cannot recover the ground truth of the traces. Consequently, we decided to increase the depth of the autoencoder to ensure it will be suitable for different types of noise.

The size of the latent representation in the middle of the autoencoder is a critical parameter that should be fine-tuned. One should be aware that although the autoencoder can reconstruct the input, some information from the input is lost. For the denoising purpose, we aim at maximizing the removal of noise while minimizing the loss of useful information. By choosing a smaller size of the bottleneck, the signal quality will be degraded. In contrast, a larger size of bottleneck may introduce less critical features to the output. To better control the size of the latent space, we flatten the output of the convolutional blocks and introduce a fully-connected layer as the middle layer in our proposed architecture.

The details on the CAE architecture used in this paper are given in Table 2. The convolution block (denoted *Convblock*) usually consists of three parts: convolution layer,

[1]We consider all sources of noise or countermeasures equally important and thus, we do not give preference toward any. In case that one aims to explore the behavior of a denoising autoencoder against only one type of noise, more tuning is possible, which will result in better performance when denoising that type of noise.

activation layer (function), and max pooling layer. As we noticed that an autoencoder implemented in this manner suffers from overfitting and poor performance in denoising the validation traces, we add the batch normalization layer to each convolution block.

Note that the size of the latent space is controlled by the number of neurons in the fully-connected layer. To ensure the CAE output has the same shape with the training sets, we develop the following equation to calculate the needed size of the fully-connected layer S_{latent}:

$$S_{latent} = \frac{S_{clean}}{\prod_{i=1}^{n} S_{pool,i}} * N_{filter0}. \tag{3}$$

S_{clean} is the size of the target clean traces, $S_{pool,i}$ represents the ith non-zero pooling stride of the decoder, and $N_{filter0}$ represents the number of the filters of the first Deconvolution block. Note, one can vary the size of the latent space for different cases by changing the size of the pooling layer as well as the number of filters.

Table 2: CAE architecture.

Block/Layer	**Filter size**	**Filter number**	**Pooling stride**
Conv block * 5	2	256	0
Conv block	2	256	5
Conv block * 2	2	128	0
Conv block	2	128	2
Conv block * 2	2	64	0
Conv block	2	64	2
Flatten	-	-	-
Fully-connected	-	-	-
Reshape	-	-	-
Deconv block	2	64	2
Deconv block * 2	2	64	0
Deconv block	2	128	2
Deconv block * 2	2	128	0
Deconv block	2	256	5
Deconv block * 5	2	256	0
Deconv block	2	1	0

We emphasize that from the functionality perspective, a CAE can be easily trained by noisy (protected)–clean (unprotected) traces pairs. Once the training finishes, the autoencoder can be used to denoise the leakages from real-world devices.

4 Experimental Results

Different types of noise must be investigated to evaluate the performance of denoising CAE. At the same time, there are no publicly available datasets concentrating on differences between noise types. To investigate the precise influence of different sources of noise in a fair way, we simulated five types of noise/countermeasures that commonly exist in the devices: Gaussian noise, desynchronization (misalignment), random delay interrupts (RDI), clock jitters, and shuffling. The simulation approaches are based on previous researches as well as the observation or implementation of the real devices. In our experiments, we show the results where the denoising architecture can reduce the GE to 0 (or close value) within 10 000 attack traces. Note that even higher noise levels are affected by CAE, but we require more measurements to reach GE close to 0. Additionally, we do not provide results for smaller levels of noise (i.e., smaller countermeasure effect), as our experiments consistently show those cases to be easier to attack. To compare the denoising performance of CAE with the existing techniques commonly used by attackers, we select and benchmark well-known denoising techniques for each type of noise. More precisely, we use static alignment [MOP06] for the treatment of misaligned traces [BHvW12]; frequency analysis (FA) [Tiu05], and, more specifically, the power spectrum density, to reduce the effect of RDIs and clock jitters [PHF08, ZDZC09]. For shuffling, additional traces are used during the profiling phase [VCMKS12]. Finally, as there is, to the best of our knowledge, no optimal method in denoising the combined noise, FA is selected for attacking the traces with the combination of the noise and countermeasures. Some additional results are given in Appendix A.

Throughout the experiments, we use the ASCAD dataset (with fixed key and random key). Note that the ASCAD dataset is masked, and there is no first-order leakage. As such, we consider the masked S-box output:

$$Y(i) = \texttt{S-box}[(p[i] \oplus k[i]) \oplus r_{out}]. \tag{4}$$

In this paper, besides the template attack, we use two machine learning models, CNN_{best} and MLP_{best} introduced in the ASCAD paper [BPS+18]. The CNN architecture is listed in Table 3, while for MLP, we use six fully-connected layers, each with 200 neurons. We use an NVIDIA GTX 1080 Ti graphics processing unit (GPU) with 11 Gigabytes of GPU memory and 3 584 GPU cores. All of the experiments are implemented with the TensorFlow [AAB+15] computing framework and Keras deep learning framework [C+15]. The time consumption to train a CAE highly depend on the length of the traces, but for the experiments performed in this paper, a CAE can be trained within one hour, on average. We note that there is no conceptual limitation on the trace length for CAE, the only limit comes from the fact that longer traces need more time.

Table 3: CNN architecture used for attacking.

Layer	**Filter size**	**# of filters**	**Pooling stride**	**# of neurons**
Conv block	11	64	2	-
Conv block	11	128	2	-
Conv block	11	256	2	-
Conv block	11	512	2	-
Flatten	-	-	-	-
Fully-connected * 2	-	-	-	4 096

Finally, for the template attack, we selected 20 POIs from the traces according to the trace variation of the Hamming weight of the intermediate data (S-box output). For each POI, the minimum distance is set to 5 to avoid selecting continuous points from the traces. For the selected hyperparameters for MLP and CNN classifiers, we used $Uniform\ distribution$ and *ReLU* activation function. The learning rate is 1e-5, while the batch size is set to 200. We use Adam optimizer for both models. During the training phase, we trained the MLP and CNN for 100 and 1 000 epochs, respectively. Finally, 35 000 traces were used for training, 5 000 for validation, and 10 000 for the attack.

We emphasize that we do not aim to find the best attack models but to show how denoising autoencoders can help improve the performance of various attacks. The quality of the recovered traces is evaluated by guessing entropy (GE). For a good estimation of GE, the attack traces are randomly shuffled, and 100 GEs are computed to obtain the average value.

4.1 Denoising the "Clean" Traces

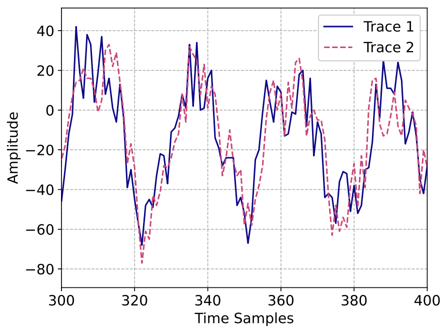
One should notice that the traces regenerated by CAE have information loss because of the bottleneck in the middle of the architecture. An ideal CAE can locate as well as precisely describe the leakage (variation) of the dataset. To evaluate the reconstruction capability, we first use CAE to denoise the "clean" traces. Here, by "clean", we consider the original traces as in the ACAD dataset with no added noise or countermeasures. Still, the traces are not clean as they have noise. In this case, the input and output of the CAE are the same measurements, while the goal of training the model is to learn how to represent the output with fewer features than at the input. As such, CAE is also an interesting strategy for feature selection, as only selecting the most important features will lead to improved recovery when compared to the original traces. We consider this scenario to 1) show that CAE removes mostly noise (features that do not contribute to the useful information), and 2) validate that if the evaluator applies CAE by mistake, the performance of the attack will not be reduced.

Here, we use the CNN_{best} model [BPS+18] for the attack. Interestingly, the SNR [Fis22] value for the traces reconstructed by CAE slightly increases by 0.05, which confirms that CAE can neglect random features (such as noise) and focus on the distinguishing ones and eventually makes the reconstructed trace more "clean". Furthermore, considering the variation for each cluster (divided by the Hamming weight or intermediate data), CAE acts as a regulator to minimize the in-cluster variance, eventually leading to a better SNR. As expected, the improvement of SNR directly leads to better performance in terms of GE: for instance, for CNN, we require 831 traces for the correct key if we use the original traces, while this value decreases to 751 after the traces are reconstructed with CAE.

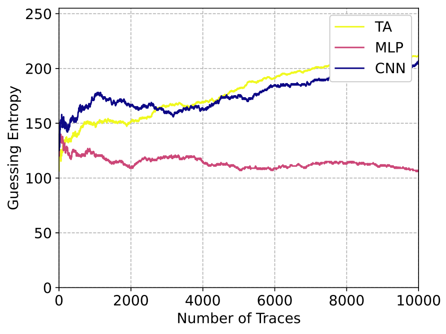
4.2 Gaussian Noise

The Gaussian noise is the most common type of noise existing in side-channel traces. The transistor, data buses, the transmission line to the record devices such as oscilloscopes, or even the work environment can be the source of Gaussian noise (inherent measurement noise). The noise can also be intentionally introduced by dummy operation or dedicated noise engine. In terms of trace leakage, the increment of the noise level hides the correlated patterns and reduces the signal-to-noise (SNR) ratio. Consequently, the noise influences the effectiveness of an attack, i.e., more traces are needed to obtain the attacked intermediate data.

To demonstrate the influence of the Gaussian noise, we add a uniformly distributed random value between -20 to 20 to each point of the trace. The pseudocode for constructing traces with Gaussian noise is shown in Algorithm 1. An example of the zoom-in view of two manipulated traces is shown in Figure 2a. Compared with the baseline traces, the Gaussian noise significantly distorted the shape of the original traces in the amplitude domain, eventually increasing the difficulties in obtaining the correct key (Figure 2).



(a) Gaussian noise: zoom-in view.



(b) Gaussian noise: guessing entropy.

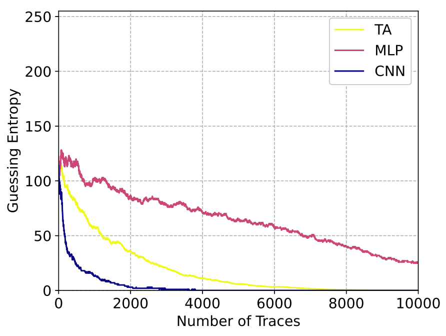
Figure 2: Gaussian noise: demonstration and its influence on guessing entropy.

Next, we denoise the Gaussian noise with trace averaging as well as CAE proposed in this paper. The GE of denoised traces with 10-trace averaging and CAE are shown in Figures 3a and 3b, respectively. From the attack perspective, GE converges in both

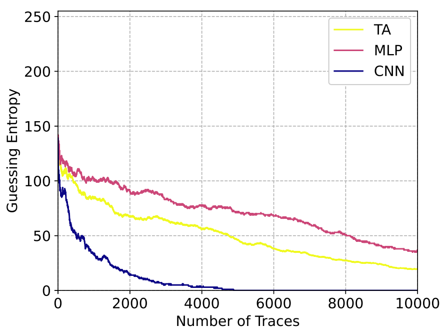
Algorithm 1 Add Gaussian Noise.

```
1: function ADD_GAUSSIAN_NOISE($trace, noise\_level$)
2:     $new\_trace \leftarrow []$                                   ▷ container for new trace
3:     $i \leftarrow 0$
4:     while $i < len(trace)$ do
5:         $level \leftarrow$randomNumber($-noise\_level, noise\_level$)
6:         $new\_trace[i] \leftarrow traces[i] + level$              ▷ add noise to the trace
7:         $i \leftarrow i + 1$
8:     return $new\_trace$
```

denoising cases when the number of trace increases. CNN attack performance after denoising with either averaging or denoising autoencoder is significantly improved over the noisy version: 3 854 averaged traces or 4 880 denoised traces are sufficient to reach GE of 0. It is worth noting that GE for averaged traces is somewhat lower than GE for CAE, confirming that trace averaging is a successful method in removing the Gaussian noise. Still, we can conclude that CAE can also remove the Gaussian noise and consequently improve the attacking efficiency. Interestingly, we see that TA works better with averaging as pre-processing, while MLP performs similarly regardless of the pre-processing method.



(a) GE: denoised with averaging.



(b) GE: denoised with CAE.

Figure 3: Guessing entropy: averaging vs CAE.

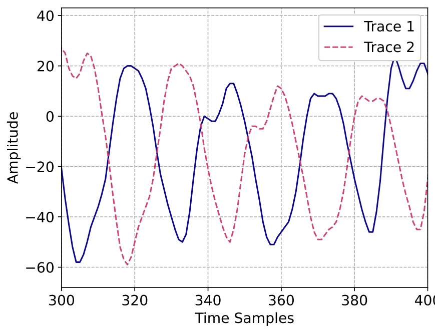
4.3 Desynchronization

Well-synchronized traces can significantly improve the correlation of the intermediate data. The alignment of the traces is, therefore, an essential step for the side-channel attack. To align the traces, usually, an attacker should select a distinguishable trigger/pattern from the traces, so that the following part can be aligned using the selected part as a reference. There are two limitations to this approach. First, the selected trigger/pattern should be distinctive, so that it will not be obfuscated with other patterns and lead to misalignment. Second, due to the existence of the signal jitters and other countermeasures, the selected trigger should be sufficiently close to the points of interest, thus minimizing the noise effect. From a practical point of view, a good reference that meets both limitations is not always easy to find. Even with an unprotected device, sometimes the traces synchronization can be a challenging task.

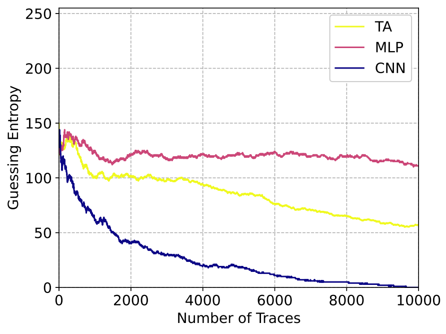
Different from the Gaussian noise, the desynchronization of the traces adds randomness to the time domain. To show the effect of the traces desynchronization, we use traces with a maximum of 50 points of desynchronization. The pseudocode for constructing traces with desynchronization is shown in Algorithm 2. An example of two zoom-in viewed traces with different desynchronization levels is given in Figure 4a, while attack results are shown in Figure 4b. From the attack results, CNN proves its ability to fight against the desynchronization effect, as 9 627 traces are sufficient for the correct key when attacking the noisy traces. Still, considering that the original "clean" traces only needed 831 traces on average to retrieve the key, the desynchronization degraded the performance of the attack. Additionally, one can expect that performance to become even worse with an increased desynchronization level.

Algorithm 2 Add Desynchronization.

```
1: function ADD_GAUSSIAN_NOISE($trace, desync\_level$)
2:     $new\_trace \leftarrow []$                          ▷ container for new trace
3:     $level \leftarrow$ randomNumber($0, desync\_level$)
4:     $i \leftarrow 0$
5:     while $i + level < len(trace)$ do
6:         $new\_trace[i] \leftarrow traces[i + level]$    ▷ add desynchronization to the trace
7:         $i \leftarrow i + 1$
8:     return $new\_trace$
```



(a) Desynchronization: zoom-in view.



(b) Desynchronization: guessing entropy.

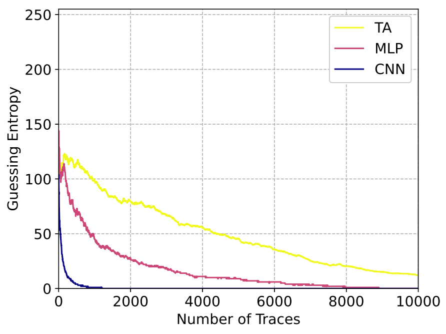
Figure 4: Desynchronization: demonstration and its influence on guessing entropy.

Next, we attack the denoised traces processed by static alignment and CAE. The GE are shown in Figures 5a and 5b. GE of the traces denoised by CAE converges faster than for the static-aligned traces. CAE provides a generic approach to synchronizing the traces, as by training a CAE with desynchronized-synchronized traces pairs, the model can align the traces automatically. As a result, compared with static alignment, the number of required traces to retrieve the key reduces from 1 180 to 822 with CNN (comparable to the attack result with the original traces), from 8 905 to 7 168 with MLP, and from more than 10 000 to 6 398 with TA, on average. Note that if attacking traces with desynchronization, we are successful with CNN only, and we require around 10 000 to reach GE of 0.

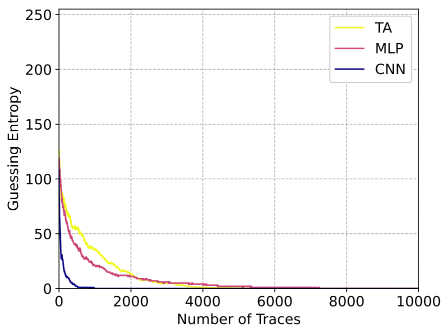
4.4 Random Delay Interrupts (RDIs)

Desynchronization introduces the global time-randomness to the entire trace. RDIs, on the other hand, lead to the time-randomness locally. As a type of countermeasure normally implemented in the software, the existence of RDIs breaks the traces into fragments, thus significantly increasing the randomness of traces in the time domain and reducing the correlation of the attacked intermediate data.

We simulate RDIs based on the Floating Mean method (with parameters a=5 and b=3) introduced in [CK09]. The RDIs implemented in such a way can provide more variance to the traces when compared with the uniform RDI distribution. To further increase the randomness of the injected RDIs, a random number (uniformly distributed between 0 and



(a) GE: denoised with static alignment.



(b) GE: denoised with CAE.

Figure 5: Guessing entropy: static alignment vs CAE.

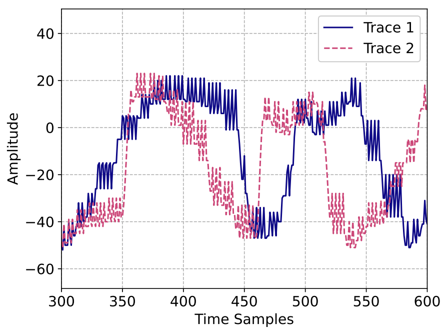
1) is first generated when scanning each point of a trace, then compared with a threshold value. We set the threshold value to 0.5, so the probability of the occurrence of the RDIs in each feature equals 50%. Moreover, in real implementations, instructions, such as *nop*, are used to generate the random delay according to the real implementations. In terms of power profile, whenever a random delay occurs, instead of flatting the power consumption, specific patterns, such as peaks, are shown in the power trace. We consider this effect by generating a small peak by adding a specific value (10) when injecting the random delays to the traces. The pseudocode for constructing traces with RDIs is shown in Algorithm 3.

Algorithm 3 Add Random Delay Interrupts.

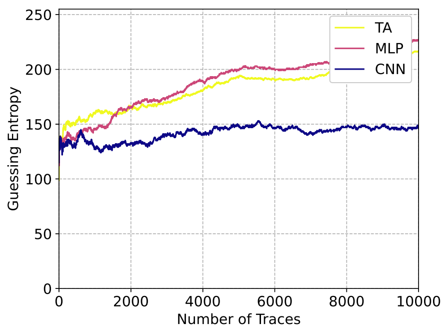
```
1:  function ADD_RDIS($traces, A, B, rdi\_occurrence, threshold, rdi\_amplitude$)
2:      $a \leftarrow A$                                                    ▷ maximum number of RDIs
3:      $b \leftarrow B$                                                    ▷ a value smaller than A
4:      $new\_trace \leftarrow []$                                          ▷ container for new trace
5:      $i \leftarrow 0$
6:      while $i < len(trace)$ do
7:          $new\_trace[i] \leftarrow new\_trace[i].append(trace[i])$
8:          if $rdi\_occurrence > threshold$ then
9:              $m \leftarrow$randomNumber$(0, a - b)$
10:             $rdi\_num \leftarrow$randomNumber$(m, m + b)$               ▷ number of RDIs to be added
11:             $j \leftarrow 0$
12:             while $j < rdi\_num$ do                                     ▷ add RDIs to the trace
13:                 $new\_trace[i] \leftarrow new\_trace[i].append(trace[i])$
14:                 $new\_trace[i] \leftarrow new\_trace[i].append(trace[i] + rdi\_amplitude)$
15:                 $new\_trace[i] \leftarrow new\_trace[i].append(trace[i + 1])$
16:                 $j \leftarrow j + 1$
17:         $i \leftarrow i + 1$
18:     return $new\_trace$
```

A zoom-in view of two example traces with random RDIs is shown in Figure 6a. The number of injected RDIs can be obtained by counting the number of peaks. From the traces, We observe that more randomness was introduced locally to the traces when compared to the traces with desynchronization, which further influence the attack result of guessing entropy. From Figure 6b, the best correct key rank of the traces with RDIs is 147 when using 10 000 traces, indicating that even the CNN_{best} model is not powerful enough to extract the useful patterns and retrieve the key. We can conclude that RDIs implemented in this way dramatically increase the attack difficulty.

The attack result with the frequency analysis (FA) and CAE are shown in Figures 7a and 7b. With FA, GE slowly decreases when using CNN and TA for the attack, while the key rank reaches 52 for the best case with TA. On the other hand, the effect of RDIs

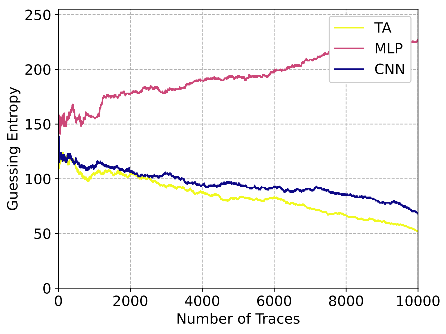


(a) RDIs: zoom-in view.

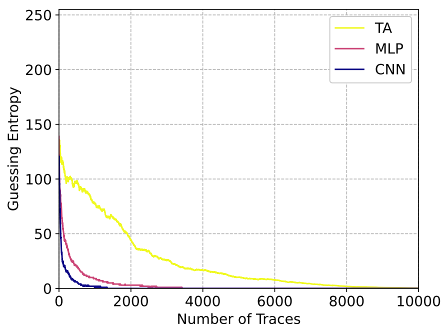


(b) RDIs: guessing entropy.

Figure 6: RDIs: demonstration and its influence on guessing entropy.



(a) GE: frequency analysis.



(b) GE: denoised with CAE.

Figure 7: Guessing entropy: frequency analysis vs CAE.

has been reduced dramatically with the help of CAE: GE converges significantly faster when attacking with TA, MLP, and CNN. CNN performance is especially good as it needs only 1 322 traces on average to reach GE of 0, while TA needs 8 952 traces and MLP 3 398 traces. We can conclude that CAE can recover the original traces from the noisy traces with RDIs countermeasure.

4.5 Clock Jitters

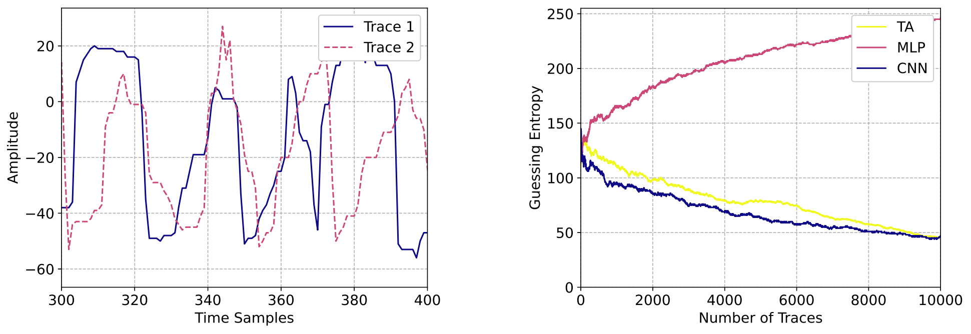
Clock jitters is a classical hardware countermeasure against side-channel attacks, realized by introducing the instability in the clock [CDP17]. Comparable to the Gaussian noise that introduces randomness to every point in the amplitude domain, the clock jitters increase the randomness for each point in the time domain. The accumulation of the deforming effect increases the misalignment of the traces and decreases the correlation of the intermediate data. As a consequence, the attacked intermediate data become more difficult to retrieve. Here, we simulate the clock jitters by randomly adding or removing points with a pre-defined range. Similar approaches are used in [CDP17]. More precisely, we generate a random number r that is uniformly distributed between -4 to 4 to simulate the clock variation in a magnitude of 8. When scanning each point in the trace, r points will be added to the trace if r is greater than zero. Otherwise, the following r points in the trace are deleted. The pseudocode for constructing traces with clock jitters is shown in Algorithm 4.

Zoom-in viewed traces with clock jitters are shown in Figure 8a. From Figure 8b, it is

Algorithm 4 Add Clock Jitters.

```
1: function ADD_CLOCK_JITTERS($trace, clock\_jitters\_level$)
2:     $new\_trace \leftarrow []$                                  ▷ container for new trace
3:     $i \leftarrow 0$
4:     while $i < len(trace)$ do
5:         $new\_trace[i] \leftarrow new\_trace[i].append(trace[i])$
6:         $level \leftarrow$randomNumber$(0, clock\_jitters\_level)$   ▷ level of clock jitters
7:         if $level < 0$ then
8:             $i \leftarrow i + level$                              ▷ skip points
9:         else
10:            $j \leftarrow 0$
11:            $average\_amplitude \leftarrow (trace[i] + trace[i+1])/2$
12:            while $j < level$ do
13:                $new\_trace \leftarrow new\_trace.append(average\_amplitude)$   ▷ add points
14:                $j \leftarrow j + 1$
15:        $i \leftarrow i + 1$
16:    return $new\_trace$
```

clear that all three classifiers are not successful in retrieving the key. A comparison of the attack results for FA and denoised traces with CAE is shown in Figure 9. Similar to the previous attack results with the RDIs countermeasure, FA is not able to retrieve the key within 10 000 traces even for the best attack (MLP with rank 41). The proposed CAE, on the other hand, successfully reduces the effect of clock jitters. Specifically, with the best setting for CNN, 8 045 traces are sufficient to obtain the correct key.



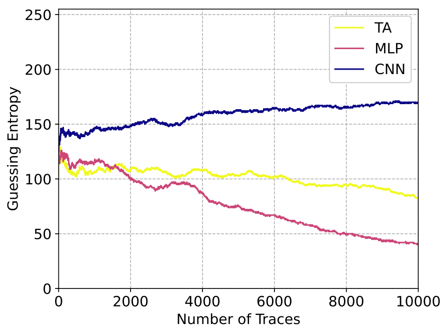
(a) Clock Jitters: zoom-in view. (b) Clock Jitters: guessing entropy.

Figure 8: Clock Jitters: demonstration and its influence on guessing entropy.

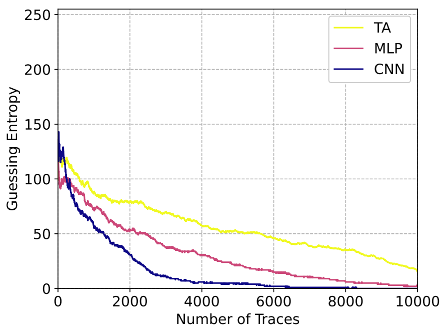
4.6 Shuffling

As a hiding countermeasure, a classical approach to realize shuffling is by randomizing the access to the S-box [VCMKS12]. With this method, it becomes more difficult for attackers to select points of interest or locate part of the traces that are correlated to the S-box-related intermediate data. Here, we simulate the shuffling effect by gathering the traces segments that are related to 16 S-box accesses and then cluster them into 16 groups. Next, for each traces to be manipulated, we randomly select one group and replace the attack traces part (related to the S-box processing) with the segment in the group. The pseudocode is shown in Algorithm 5[2]

[2]We acknowledge that the described algorithm may not produce the same effect as the actual shuffling, but we consider it to be a valid showcase for the experimental evaluation, and the closest option to simulate the effect of shuffling.



(a) GE: frequency analysis.



(b) GE: denoised with CAE.

Figure 9: Guessing entropy: frequency analysis vs CAE.

Algorithm 5 Add shuffling.

```
1: function ADD_SHUFFLING($trace, sbox\_seg$)
2:     $new\_trace \leftarrow []$                                          ▷ container for new trace
3:     $i \leftarrow 0$
4:     while $i < len(trace)$ do
5:         $sbox\_idx \leftarrow$randomNumber$(3, 16)$
6:         $new\_trace[i] \leftarrow traces[i].replace(sbox\_seg[sbox\_idx])$     ▷ replace sboxs
7:         $i \leftarrow i + 1$
8:     return $new\_trace$
```

Note that shuffling does not change the shape of the traces dramatically, so we do not demonstrate the shape of the traces here. Figure 10 shows the attack results for the shuffling countermeasure. Although the GE is slowly converging for both three attack methods (rank 32 for the best case with MLP), none of them reaches zero within 10 000 traces.

The results improve when we use additional 10 000 traces for profiling. As can be seen in Figure 11a, for the best case with MLP, we reach rank two after 10 000 traces, indicating that deep learning attacks can combine complex features as well as to handle the trace randomness. CNN is slightly worse, while TA does not manage to converge to a successful attack (rank 30). The traces denoised with CAE give the best results (Figure 11b), as only 7 754 traces are needed for the correct key when using CNN. TA is somewhat worse, and its rank equals six after 10 000 traces. We emphasize that with CAE, we use only 10 000 traces and we get better results than with 20 000 traces without using CAE.

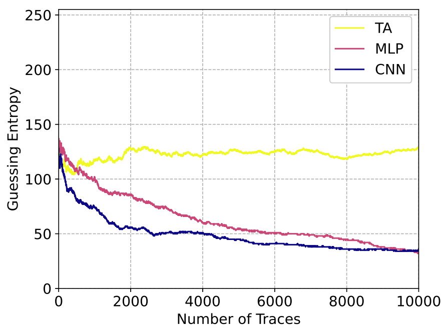
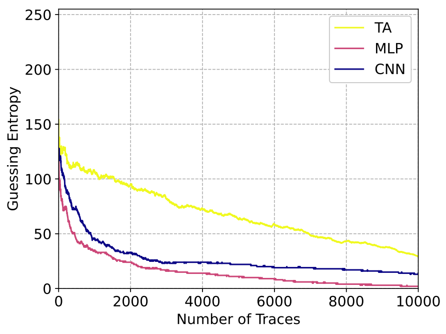
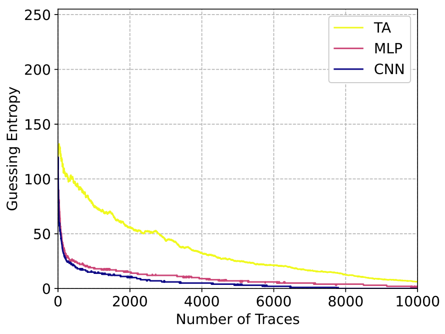


Figure 10: Shuffling: guessing entropy.



(a) GE: profiling with additional 10 000 traces.



(b) GE: denoised with CAE.

Figure 11: Guessing entropy: more traces vs CAE

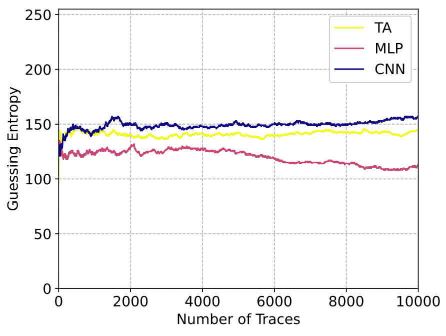
To conclude, the proposed CAE proves its ability to limit the effect of the Gaussian noise, desynchronization, random delay interrupts, clock jitters, and shuffling. Also, traces denoised with CAE shows comparable, and in many cases, even better results compared to specific denoising/signal processing approaches. Finally, denoising autoencoder works for TA, MLP, and CNN attacks, but for most of the cases, CNN's performance is the best.

4.7 Combining the Effects of Gaussian Noise and Countermeasures

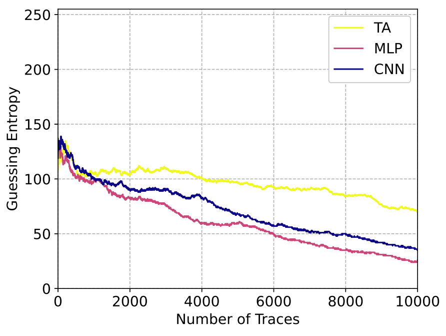
In the previous section, we add and denoise different types of noise individually. Now, we investigate an extreme situation by adding all five discussed noise/countermeasures together and verifying the effectiveness of the CAE approach. To maximize the effectiveness of each type of noise as well as keep the simulated traces close to the realistic, we added the noise in the order: shuffling - desynchronization - RDI - clock jitters - Gaussian noise. Note there would be fewer countermeasures combined in the traces in realistic settings. In such cases, we expect that the performance of the proposed CAE would be better, as evident from scenarios when handling only a single countermeasure. We test two different datasets: AES with a fixed key and AES with random keys. Since there are no specific approaches in reducing the effect of combined noise sources, we evaluate GE of the noisy traces and traces after applying frequency analysis and CAE. Note that we do not, for instance, apply averaging after frequency analysis as then, we do not have enough measurements to conduct a successful attack.

Similar to the procedure of the previous sections, we calculated the GE of the noisy and denoised traces and made a comparison between them. A demonstration of the manipulated traces with all types of noise and countermeasures is presented in Figure 16. As expected, the three attack methods used in the paper are not able to obtain the correct key within 10 000 traces. In fact, we observe that the noisy traces do not converge with the increasing number of traces.

Frequency analysis is not working when dealing with the combination of noise and countermeasures (which is not surprising as we now have noise sources where this technique is not sufficient). The GE of denoised traces with CAE, on the other hand, reaches 26 with 10 000 traces when using MLP. Somewhat worse is CNN and it reaches rank 35 after 10 000 traces. This again confirms that the denoising autoencoder can work regardless of the applied attack technique. What is more, we see that a simpler attack technique in combination with CAE can outperform more complicated attack techniques (*cf.* TA with CAE, and CNN with the frequency analysis). Finally, we can observe that the attack performance converges slower than for the denoised traces with a single type of noise, but CAE still proves its capability in removing the combined effect of noise and



(a) GE: frequency analysis.



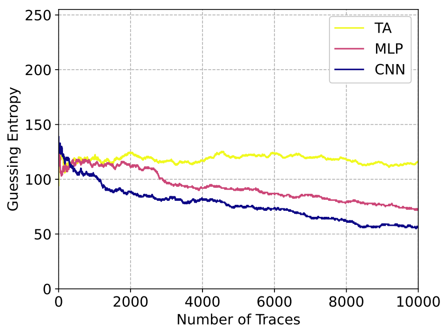
(b) GE: denoised with CAE.

Figure 12: Guessing entropy: frequency analysis vs CAE.

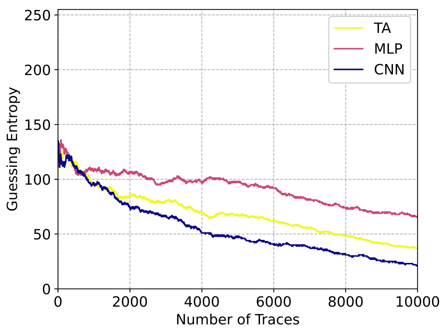
countermeasures.

4.7.1 AES with Random Keys

Finally, we verify the performance of the CAE by trying to denoise the AES traces with random keys. To retrieve the correct key from the leakage traces, we first train the model with leakage with a random but known key, then use the trained model to attack the leakages and try to retrieve the unknown key. When comparing with the fixed-key traces, the randomness of the key introduces more variance into the traces, thus further increasing the difficulties in denoising the traces. In terms of attack settings, there are 1 400 features in every trace. The attacked intermediate data was kept the same.



(a) GE: frequency analysis.



(b) GE: denoised with CAE

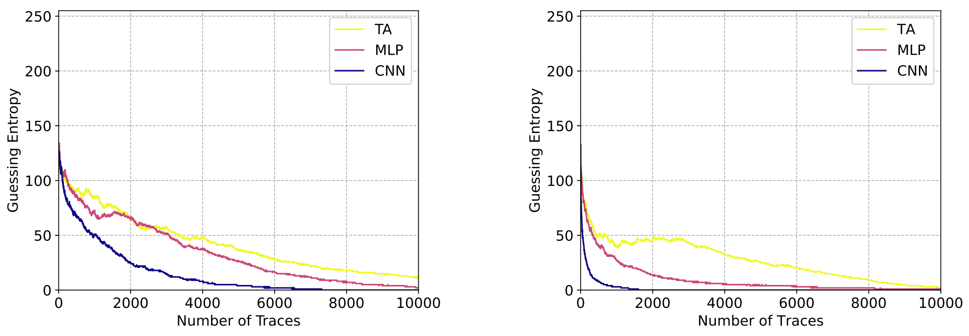
Figure 13: Guessing entropy: frequency analysis vs CAE.

From the attack results, the GE of the noisy traces fluctuates above 100 regardless of the number of traces (Figure 17). On the other hand, guessing entropy indicates improved performance as a result of FA and CAE. For the best cases, GE value converges to 56 with 10 000 traces with FA and CNN, and to 21 with CAE and CNN. What is more, again, we can see that TA with CAE is better than CNN with FA, which confirms that a strong pre-processing can more than compensate for a "weaker" attack mechanism. Finally, we conclude that the proposed CAE can denoise the leakage in both fixed key and variable key scenarios where the results are especially strong if using CNN as the attack mechanism. What is especially interesting is to observe how even less powerful attacks can produce better results if denoised with CAE, then a more powerful attack when using the noisy traces (e.g., TA after CAE, and CNN without CAE).

4.8 Case Study: From Noisy to Less Noisy – The Black-box Setting

The denoising strategy we proposed in this paper is orientated more toward white-box settings, as the evaluator has the full control of the device so that the clean traces can be easily obtained by turning off the countermeasures. The denoising strategy cannot be directly applied when considering the difficulties in disabling the countermeasures for the black-box settings. Fortunately, CAE can denoise the traces even when the reference traces are not perfectly clean. In other words, the less noisy traces generated by the traditional denoising methods can also be used as the "clean" traces for CAE training.

We investigate noisy-to-less-noisy scenarios with Gaussian noise and desynchronization. To handle the noisy traces, the traces denoised by averaging (for Gaussian noise) and static alignment (for desynchronization) are used as the "clean" traces at the output of the CAE. Note that the traces denoised by these two methods are not perfectly denoised. Still, CAE can reduce noise levels by mapping the noisy traces to less noisy traces. First, we denoise the traces with Gaussian noise and desynchronization separately. The results are given in Figure 14.



(a) GE (Gaussian noise): train CAE from noisy to averaged traces.

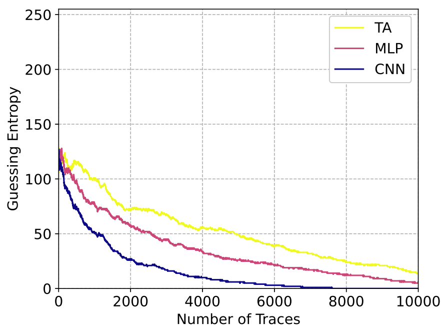
(b) GE (desynchronization): train CAE from noisy to static aligned traces.

Figure 14: Guessing entropy: denoised from less noisy traces.

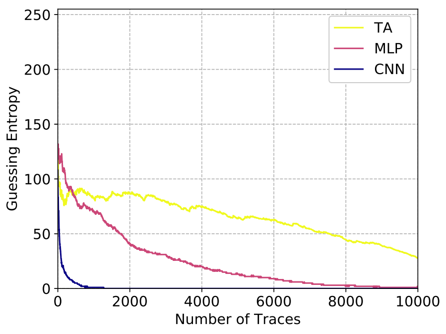
Compared with the denoised traces using the original clean traces as the reference, the attack performance for the noise-to-less-noise cases is degraded. Specifically, 3 584 traces are required to retrieve the correct key when using the clean traces to denoise the Gaussian noise (white-box setting), while this increases to 7 258 when denoised with averaged traces. The attack performance degradation is similar in terms of removing desynchronization: 822 traces to attack when denoised from the clean traces and 1 604 traces when denoised from the static aligned traces. One can expect that with deeper (or improved) CAE models with improved denoising ability, the variation of the attack performance between different clean references can be further minimized. Also, note that we do not specifically optimize the denoising approach, so the denoising performance of the CAE can be further improved with cleaner traces (e.g., more traces for averaging).

Finally, we denoised the traces with Gaussian noise and desynchronization in a combined setting. Specifically, 10 000 trace pairs with Gaussian noise (noisy-averaged) and 10 000 trace pairs with desynchronization (noisy-static aligned) are combined and used for training the CAE. The results are presented in Figure 15.

Interestingly, the joint training method leads to similar performance compared with the previous results on a single noise source. To be specific, 7 576 traces are needed for the Gaussian noise and 1 275 for the desynchronization. This again shows that the CAE



(a) GE (Gaussian noise).



(b) GE (desynchronization).

Figure 15: Guessing entropy: combined training of CAE.

model can learn and remove different types of noise simultaneously. More precisely, we can train CAE to remove various types of noise, and it will work even if using traces that do not have all noise sources.

5 Conclusions and Future Work

In this paper, we introduce a convolutional autoencoder to remove the noise and countermeasure from the leakage traces. We consider different types of noise and countermeasures: Gaussian noise, desynchronization, random delay interrupts, clock jitters, and shuffling. Additionally, we simulate the scenario where all noise types and countermeasures are combined into the measurements. To strengthen our experimental results, we consider two types of leakage traces (one encrypted with fixed and another with random keys) and three attack strategies (CNN, MLP, and TA). The obtained results show that the proposed CAE can still remove/reduce the noise and find out the underlying ground truth and thus significantly improve the attack performance.

Our approach is especially powerful in the white-box settings, but we demonstrate it has potential also in black-box settings. We believe it is especially interesting to consider denoising autoencoders as a generic denoiser technique since our results indicate it gives good results, while it is easy to apply it. What is more, our results show that autoencoders reliably remove noise/countermeasures even if the measurements do not contain all the noise sources the autoencoder was trained with.

Denoising autoencoder provides an attacker with a powerful tool to pre-process the traces. We expect this technique could be used to help solve other problems like portability [BCH+19]. There, the biggest obstacle stems from the variance among different devices. These variances introduce the variation of the trace, making the attack model generated for one device challenging to transfer to another one. With the help of an autoencoder, this problem can be solved by considering the traces variation as noise and use denoising autoencoder to remove it. We note that this setting is similar to the scenario with added Gaussian noise, which indicates that the CAE approach should be very useful in portability. Finally, in future work, we aim to investigate whether denoising autoencoder could also work for the masking countermeasures.

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[AC15] Jinwon An and Sungzoon Cho. Variational autoencoder based anomaly detection using reconstruction probability. *Special Lecture on IE*, 2(1), 2015.

[Bal12] Pierre Baldi. Autoencoders, unsupervised learning, and deep architectures. In *Proceedings of ICML workshop on unsupervised and transfer learning*, pages 37–49, 2012.

[BCH+19] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/661, 2019. https://eprint.iacr.org/2019/661.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 16–29. Springer, 2004.

[BDF+17] Gilles Barthe, François Dupressoir, Sebastian Faust, Benjamin Grégoire, François-Xavier Standaert, and Pierre-Yves Strub. Parallel implementations of masking schemes and the bounded moment leakage model. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 535–566. Springer, 2017.

[BHvW12] Lejla Batina, Jip Hogenboom, and Jasper GJ van Woudenberg. Getting more from pca: first results of using principal component analysis for extensive power analysis. In *Cryptographers' track at the RSA conference*, pages 383–397. Springer, 2012.

[BPS+18] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ascad database. *ANSSI, France & CEA, LETI, MINATEC Campus, France. Online verfügbar unter https://eprint. iacr. org/2018/053. pdf, zuletzt geprüft am*, 22:2018, 2018.

[C+15] François Chollet et al. Keras. https://github.com/fchollet/keras, 2015.

[CCD00] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 252–263. Springer, 2000.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In *International Conference on Cryptographic Hardware and Embedded Systems*, pages 45–68. Springer, 2017.

[CJRR99] Suresh Chari, Charanjit S Jutla, Josyula R Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In *Annual International Cryptology Conference*, pages 398–412. Springer, 1999.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An Efficient Method for Random Delay Generation in Embedded Software. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 156–170, 2009.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template Attacks. In *CHES*, volume 2523 of *LNCS*, pages 13–28. Springer, August 2002. San Francisco Bay (Redwood City), USA.

[Fis22] Ronald 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):309–368, 1922.

[GD98] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[GHO15] Richard Gilmore, Neil Hanley, and Maire O'Neill. Neural network based attack on a masked implementation of aes. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111. IEEE, 2015.

[Gon16] Lovedeep Gondara. Medical image denoising using convolutional denoising autoencoders. In *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*, pages 241–246. IEEE, 2016.

[Gur14] Kevin Gurney. *An introduction to neural networks*. CRC press, 2014.

[HPGM16] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In *Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers*, pages 91–104, 2016.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.

[KPH$^+$19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[KUMH17] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in neural information processing systems*, pages 971–980, 2017.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A Machine Learning Approach Against a Masked AES. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2013. Berlin, Germany.

[LPB$^+$15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 20–33. Springer, 2015.

[MOP06] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks: Revealing the Secrets of Smart Cards*. Springer, December 2006. ISBN 0-387-30857-1, http://www.dpabook.org/.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[PHF08] Thomas Plos, Michael Hutter, and Martin Feldhofer. Evaluation of side-channel preprocessing techniques on cryptographic-enabled hf and uhf rfid-tag prototypes. In *Workshop on RFID Security*, pages 114–127, 2008.

[PHJ$^+$17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 4095–4102, 2017.

[PHJ$^+$19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. 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.

[PHJB19] S. Picek, A. Heuser, A. Jovic, and L. Batina. A systematic evaluation of profiling through focused feature selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12):2802–2815, Dec 2019.

[PSK$^+$18] Stjepan Picek, Ioannis Petros Samiotis, Jaehun Kim, Annelie Heuser, Shivam Bhasin, and Axel Legay. On the performance of convolutional neural networks for side-channel analysis. In Anupam Chattopadhyay, Chester Rebeiro, and Yuval Yarom, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 157–176, Cham, 2018. Springer International Publishing.

[QS01] Jean-Jacques Quisquater and David Samyde. Electromagnetic analysis (ema): Measures and counter-measures for smart cards. In Isabelle Attali and Thomas Jensen, editors, *Smart Card Programming and Security*, pages 200–210, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[RHW85] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[SMY09] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 443–461. Springer, 2009.

[TGWC18] Hugues Thiebeauld, Georges Gagnerot, Antoine Wurcker, and Christophe Clavier. Scatter: A new dimension in side-channel. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 135–152. Springer, 2018.

[Tiu05] Chin Chi Tiu. *A new frequency-based side channel attack for embedded systems*. PhD thesis, University of Waterloo, 2005.

[TSCH17] Lucas Theis, Wenzhe Shi, Andrew Cunningham, and Ferenc Huszár. Lossy image compression with compressive autoencoders. *arXiv preprint arXiv:1703.00395*, 2017.

[TV03] Kris Tiri and Ingrid Verbauwhede. Securing encryption algorithms against dpa at the logic level: Next generation smart card technology. In *International Workshop on Cryptographic Hardware and Embedded Systems*, pages 125–136. Springer, 2003.

[VCMKS12] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 740–757. Springer, 2012.

[WLL$^+$18] Lingxiao Wei, Bo Luo, Yu Li, Yannan Liu, and Qiang Xu. I know what you see: Power side-channel attack on convolutional neural network accelerators. In *Proceedings of the 34th Annual Computer Security Applications Conference*, pages 393–406. ACM, 2018.

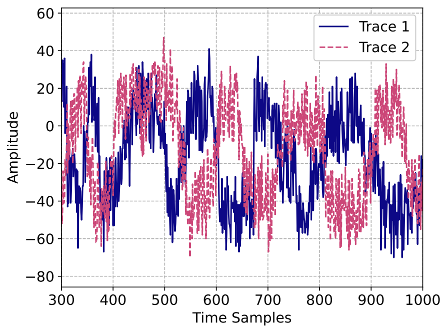
[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.

[ZDZC09] Peng Zhang, Gaoming Deng, Qiang Zhao, and Kaiyan Chen. Em frequency domain correlation analysis on cipher chips. In *2009 First International Conference on Information Science and Engineering*, pages 1729–1732. IEEE, 2009.

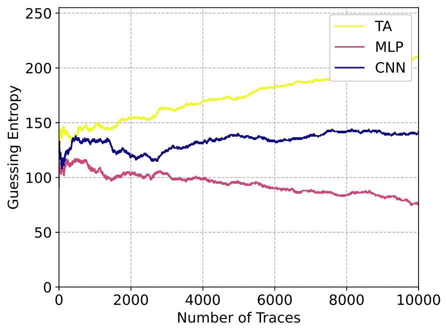
[ZZY$^+$15] Yingxian Zheng, Yongbin Zhou, Zhenmei Yu, Chengyu Hu, and Hailong Zhang. How to compare selections of points of interest for side-channel distinguishers in practice? In Lucas C. K. Hui, S. H. Qing, Elaine Shi, and S. M. Yiu, editors, *Information and Communications Security*, pages 200–214, Cham, 2015. Springer International Publishing.

A Additional Results

In Figure 16, we depict results for the AES with the fixed key where all five sources of noise are combined. Clearly, using 10 000 traces is not enough to break the target with any of the considered attacks. Similar behavior we observe when attacking AES with random keys (Figure 17. Interestingly, when considering a fixed key, we see that MLP is by far the best performing algorithm, while for the scenario with random keys, all three algorithms perform similarly, where the best performance is seen for CNN. Finally, key rank for the best performing algorithm is slightly lower when considering fixed key setting, but the differences are small enough to indicate that both scenarios are too difficult for tested attacks.



(a) All in one: zoom-in view.



(b) All in one: guessing entropy.

Figure 16: All in one (fixed key): demonstration and its influence on guessing entropy.

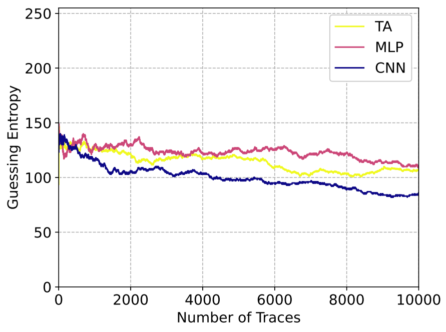


Figure 17: All in one (random key): guessing entropy.