IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2020, No. 4, pp. 389–415. DOI:10.13154/tches.v2020.i4.389-415

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 complex, and such 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.
This paper investigates 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 six different types of noise and countermeasures separately or combined and show that 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 cryptographic algorithms' physical implementations 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]. Machine learning-based approaches and deep learning-based approaches have proved to be powerful options when conducting profiled SCA in recent years. While such attack methods actively threaten the security of cryptographic devices, there are still severe limitations. More precisely, attack methods commonly rely on the signal's correlation characteristics, i.e., signal patterns related to the processed data. Once the correlation degrades, attacks become less effective or even useless [BCO04].

The countermeasures can be divided into two categories: masking and hiding. The masking countermeasure splits the sensitive intermediate values into different shares to decrease the key dependency [CJRR99, BDF+17]. The hiding countermeasure aims to reduce the side-channel information by adding randomness to the leakage signals or 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

Received: 2020-04-15 Accepted: 2020-06-15 Published: 2020-08-26

the used hiding approaches, 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 leakage's ground truth, then the reconstructed traces could become more vulnerable to SCA.

While considering the countermeasures as noise and 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]. Combining all these factors makes this problem very complicated, and to the best of our knowledge, there are no universal approaches to removing 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 counteract the effect of 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, the results could be lack of interpretability. 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.

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 several sources such as images [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., Gaussian noise, uniform noise, desynchronization, RDIs, clock jitters, and shuffling. We then increase the problem difficulty 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 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 to apply any attack strategy. Our approach is especially powerful when considering the white-box scenarios, but we also discuss 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 the 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]. 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. The side-channel community first 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. Specifically, CNNs were demonstrated to cope with the random delay countermeasure due to their spatial invariance property [CDP17, KPH+19]. At the same time, the fully-connected layers in MLP and CNNs 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 has not been explored before in the side-channel domain to the best of our knowledge. More precisely, we introduce a novel approach to remove the effect of countermeasures, and we propose:

1. A convolutional autoencoder architecture that requires a limited number of traces to train the model, and capable of denoising/removing 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 six 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. Additionally, we explore PCA's influence on the performance of template attack and input noise addition for CNNs (as a regularization factor). Finally, we experiment with different datasets, having either fixed or random keys, to show our approach's universality.

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 key space 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

In profiled SCAs, we assume an attacker with access to an open (keys can be chosen/or are known by the attacker) clone device. Then, the attacker uses 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 collects a few (T) traces from the attack device where the secret key is unknown. 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). Details about attack techniques used in this paper (template attack, multilayer perceptron, and convolutional neural networks) are given in Appendix A.

To assess the attacker's performance, one needs 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 $g = [g1, g2, \ldots, g_{|K|}]$ in decreasing order of probability. Then, GE is the average position of k^* in g over several experiments.

2.2 Neural Networks

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. $f : \mathcal{X} \to \mathcal{Y}$, 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.

2.3 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]. There are two datasets recorded in different conditions: fixed key encryption and random keys encryption. For the data with fixed key encryption, we use a dataset that is time-aligned in a prepossessing step [BPS+18]. There are 60 000 EM traces (50 000 training/cross-validation traces and 10 000 test traces), and each trace consists of 700 points of interest (POI). For the data with random keys 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. We use the raw traces and the pre-selected window of relevant samples per trace corresponding to masked S-box for $i = 3$.

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 such as data compression [TSCH17], anomaly detection [SY14], 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. (1)), i.e., the most representative input features are forced to be kept in the smallest layer in the network:

$$\phi, \psi = \arg\min_{\phi,\psi} \|X - (\psi \circ \phi)X\|^2. \tag{1}$$

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 latent space while neglecting other random factors. Since the original trace (without noise) can be recovered by feeding noisy traces to the autoencoder's input, one can expect that the attack efficiency will be significantly improved with the recovered traces.

3.2 Denoising Strategy

As discussed in Section 3.1, we require noisy-clean trace pairs to train a denoising autoencoder. In our context, 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 pre-process the leakage traces obtained from the device B. Finally, with "clean" (or, at least, cleaner) traces reconstructed by the denoising autoencoder, an attacker could eventually retrieve the secret information with less effort. For practical attack scenarios, the biggest challenge for this strategy is *how* to obtain clean traces:

1. **White-box setting**. For software implementations, an attacker might have a device where he can modify the code, and then turn off the countermeasures. For hardware implementations, the scenario is more complicated. Let us consider a cryptographic core on an SoC: an attacker might be able to turn off countermeasures by setting the control registers of the cryptographic engine if he has run-time control of the SoC's main processor. For signature schemes, the public key's verification procedure sometimes 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 turn off some (or all) countermeasures.
2. **Black-box setting**. Here, the attacker cannot 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. Even if we train autoencoder for different types of noise simultaneously, 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 Gaussian noise scenario. The first option is to use noisy traces and build a profiling model on such traces. Then, we use that model to attack noisy traces. Alternatively, we can use averaging on profiling traces, and then we build a profiling model. Next, we apply the averaging on attack traces and use the profiling model. While this will work, due to averaging, we reduce the number of profiling traces (less severe as we assume unlimited traces) and the number of attack traces, which could directly influence the attack performance. Now, let us consider a denoising autoencoder setup. We can use averaging on profiling traces and apply the original/averaged measurements to train the autoencoder, and then use the averaged measurements to build a profiling model. When conducting the attack, we apply

the autoencoder on attack (denoised) traces. As a consequence, there is no size reduction of the attack trace set.

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. Naturally, for some other countermeasures, like desynchronization, there is no dataset size reduction if applying specialized denoising techniques. Still, autoencoder brings the advantages of having an autoencoder model based on profiling traces. This makes the denoising process potentially faster when compared to the independent application of specialized techniques and more adapted to the profiling model, which will potentially improve the attack performance. We give experimental results for the black-box setting in Section 4.8.

Moreover, denoising autoencoder serves well as a generic denoiser technique, so that it can be used in denoising other countermeasures or types of measurements. For instance, the verification procedure could be used for training the denoising autoencoder. Although the signature generation and verification contain many operations, scalar multiplication is the most prominent one and shared by both of them. As such, we presume the training with verification procedure will work for most of the cases.

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 best performing model 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]. We display the tuning range and selected hyperparameters in Table 1. We use the *SeLU* activation function to avoid vanishing and exploding gradient problems [KUMH17].

Table 1: CAE hyperparameter tuning.

Hyperparameter	Range	Selected
Optimizer	Adam, RMSProb, SGD	Adam
Activation function	Tanh, ReLU, SeLU	SeLU
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 autoencoder architecture, we observed that an autoencoder with a shallow architecture could successfully denoise the traces when dealing with trace desynchronization. Still, when introducing other types of noise into the traces while keeping the same hyperparameters, such autoencoders cannot recover the traces' ground truth. Consequently, we decided to increase the autoencoder's depth to ensure it would be suitable for different noise types.

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

[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.

can reconstruct the input, some information from the input is lost. We aim to maximize the noise removal capability for the denoising purpose while minimizing useful information loss. By choosing a smaller size of the latent space, the signal quality will be degraded. In contrast, a larger size may introduce less critical features to the output. To better control the latent space's size, we flatten the convolutional blocks' output and introduce a fully-connected layer with 512 neurons as the middle layer in our proposed architecture.

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

The latent space's size 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}:

$$S_{latent} = \frac{S_{clean}}{\prod_{i=1}^{n} S_{pool,i}} * N_{filter0}. \tag{2}$$

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 latent space's size for different cases by changing the pooling layer's size and the number of filters.

Table 2: CAE architecture.

Block/Layer	# of filters	Filter number	Pooling stride	# of neurons
Conv block * 5	2	256	0	-
Conv block	2	256	5	-
Conv block * 3	2	128	0	-
Conv block	2	128	2	-
Conv block * 3	2	64	0	-
Conv block	2	64	2	-
Flatten	-	-	-	-
Fully-connected	-	-	-	512
Fully-connected	-	-	-	S_{latent}
Reshape	-	-	-	-
Deconv block	2	64	2	-
Deconv block * 3	2	64	0	-
Deconv block	2	128	2	-
Deconv block * 3	2	128	0	-
Deconv block	2	256	5	-
Deconv block * 5	2	256	0	-
Deconv block	2	1	0	-

We emphasize that 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

To investigate the precise influence of different sources of noise in a fair way, we simulated six types of noise/countermeasures: Gaussian noise, uniform noise (results in Appendix B.2), desynchronization (misalignment), random delay interrupts (RDI), clock jitters, and shuffling. The simulation approaches are based on previous research and the observation or implementation of the real devices. Our experiments show that the denoising architecture

can reduce GE to 0 (or close value) within 10 000 attack traces. Note that CAE could also reduce even higher noise levels, but more measurements are required to reach GE close to 0. Additionally, we do not provide results for scenarios with less noise (i.e., smaller countermeasure effect), as our experiments consistently show those cases to be easier to attack. To compare CAE's denoising performance with the existing techniques commonly used by attackers, we select and benchmark well-known denoising techniques for each type of noise. First, we consider principal component analysis (PCA) [WEG87], a well-known dimensionality reduction technique. The PCA is combined with TA, where TA uses the first 20 principal components without additional POIs selection. Additionally, recent research shows that the addition of noise can enhance the robustness of the model, eventually becoming beneficial in the process of classification [KPH+19]. Thus, we also use CNNs, where we add Gaussian noise to the input layer to improve the model's classification performance. We tested several noise levels in the range from 0.05 to 0.25 (recommendations from [KPH+19]), and we select to use a noise variance of 0.1 as it provides the best performance improvement. Finally, we apply specialized techniques to denoise specific types of noise. More precisely, we use static alignment [MOP06] for the treatment of misaligned traces [BHvW12]; frequency analysis (FA) [Tiu05], which is a method to analyze leakages in the frequency domain by transferring the data to its power spectrum density, to reduce the effect of RDIs and clock jitters [PHF08, ZDZC09]. For shuffling, we use additional traces during the profiling phase [VCMKS12]. To the best of our knowledge, there is no optimal method in denoising the combined noise, and we use FA for traces with the combination of the noise and countermeasures. Additional results for the combination of noise sources are given in Appendix B.

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

$$Y(i) = \texttt{S-box}[(p[i] \oplus k[i]) \oplus r_{out}]. \tag{3}$$

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 depends 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 CAE's trace length; the only limit is that longer traces need more processing 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

We selected 20 POIs from the traces according to the trace variation of the intermediate data (S-box output) for the template attack. 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 $RMSProb$ optimizer is used for both models with the learning rate of 1e-5, while

the batch size equals 128. During the training phase, the MLP and CNN are trained 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 show how denoising autoencoders can help improve various attacks' performance. 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 key ranks 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 could 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 use the original traces as in the ASCAD dataset with no added noise or countermeasures. Still, note that the traces are not perfectly clean as the noise still exists. In this case, the CAE input and output 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. 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.

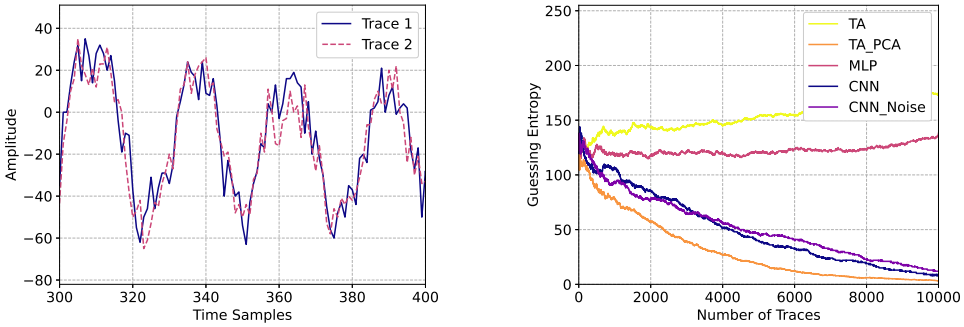
We use the CNN_{best} model [BPS+18] for the attack. Interestingly, the SNR [Fis22] value for the traces reconstructed by CAE slightly increases by 0.05, which confirms that CAE can discard random features (such as noise) and focus on the distinguishing ones and eventually make the reconstructed trace 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, we require 831 traces for the correct key for CNN if we use the original traces, while this value decreases to 751 after the traces are reconstructed with CAE. Similar performance could be observed when adding the noise to the input layer: the required number of traces decreases from 742 to 647. Note that CNN's performance with added noise is slightly better than the version without noise, proving that noise, as a regularization factor, could improve the attack efficiency. For MLP, the required traces reduced from 1 930 to 1 084. For TA, the required traces reduced from 5 928 to 4 667; when applying PCA to the traces, 615 and 635 traces are required to obtain the correct key for two datasets. The detailed results are presented in Figure 15, Appendix B.

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 parallel operations or a dedicated noise engine. In terms of trace leakage, the noise level's increment hides the correlated patterns and reduces the signal-to-noise (SNR) ratio. Consequently, the noise influences an attack's effectiveness, i.e., more traces are needed to obtain the attacked intermediate data.

To demonstrate the influence of the Gaussian noise, we add normal-distributed random value with zero mean and variance of eight 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 1a. PCA-based TA shows the best performance with GE equal to 3 after applying 10 000 attack traces. CNN and CNN

with added noise from the input layer (CNN_Noise) also converge to low guessing entropy, while TA and MLP do not succeed in the attack. 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 1).



(a) Gaussian noise: zoom-in view.

(b) Gaussian noise: guessing entropy.

Figure 1: Gaussian noise: demonstration and its influence on guessing entropy.

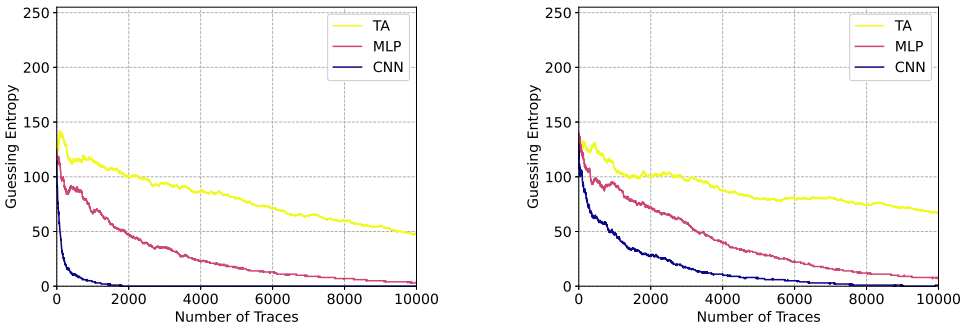
Algorithm 1 Add Gaussian Noise.

```
function ADD_GAUSSIAN_NOISE(trace, mean, variance)
    new_trace ← []                                  ▷ container for new trace
    i ← 0
    while i < len(trace) do
        level ←randomNumber(mean, variance)
        new_trace[i] ← traces[i] + level            ▷ add noise to the trace
        i ← i + 1
    return new_trace
```

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 2a and 2b, respectively. From the attack perspective, GE converges in both cases when the number of trace increases. After denoising with either averaging or denoising autoencoder, CNN attack performance is significantly improved over the noisy version: 1 754 averaged traces, or 8 751 denoised traces are sufficient to reach GE of 0. Interestingly, TA and MLP perform similarly regardless of the pre-processing method, while CNN introduces differences in attacking performance. It is worth noting that GE for averaged traces is lower than GE for CAE, confirming that trace averaging is a successful method in removing the Gaussian noise. Still, we confirm that CAE is capable of removing the Gaussian noise and improve the attacking efficiency.

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. However, 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. A good reference that meets both limitations is not always easy to find



(a) GE: denoise with averaging.

(b) GE: denoise with CAE.

Figure 2: Guessing entropy: denoising Gaussian noise with averaging (a) and CAE (b).

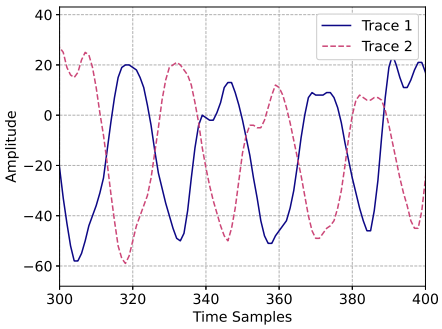
from a practical perspective. Even with an unprotected device, sometimes the traces synchronization can be a challenging task.

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 3a, while attack results are shown in Figure 3b. 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. Considering that the original "clean" traces only needed 831 traces on average to retrieve the key, the desynchronization degraded the attack's performance. Additionally, one can expect that performance to become even worse with an increased desynchronization level. Note that TA-PCA results do not converge at all, as PCA breaks the information's spatial ordering.

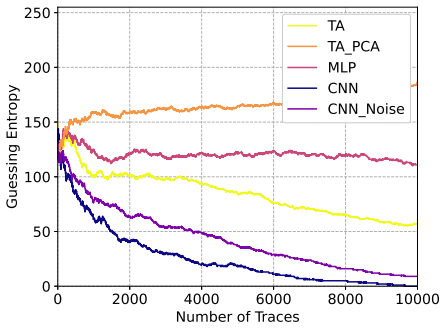
Algorithm 2 Add Desynchronization.

```
1: function ADD_DESYNC(trace, desync_level)
2:     new_trace ← []                          ▷ container for new trace
3:     level ←randomNumber(0, desync_level)
4:     i ← 0
5:     while i + level < len(trace) do
6:         new_trace[i] ← traces[i + level]    ▷ add desynchronization to the trace
7:         i ← i + 1
8:     return new_trace
```

Next, we attack the denoised traces pre-processed by static alignment or CAE. The GE are shown in Figures 4a and 4b. 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 automatically align the traces. 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). For MLP and TA, the number of required traces reduces from 8 905 to 7 168, and more than 10 000 to 6 398. Note that if attacking traces with desynchronization (Figure 3b), we are successful with CNN only with more than 9 000 attack traces to reach GE of 0.

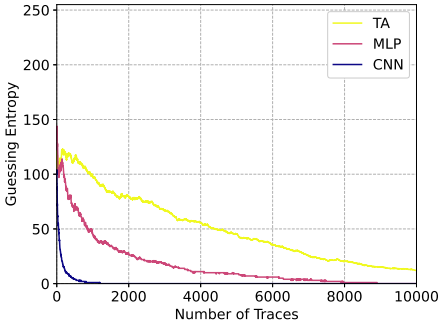


(a) Desynchronization: zoom-in view.

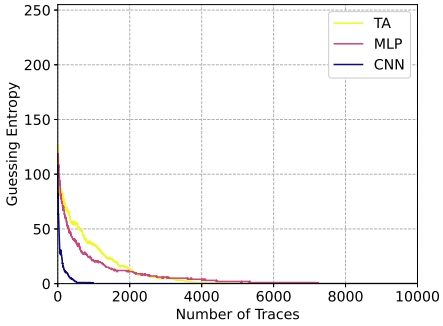


(b) Desynchronization: guessing entropy.

Figure 3: Desynchronization: demonstration and its influence on guessing entropy.



(a) GE: denoise with static alignment.



(b) GE: denoise with CAE.

Figure 4: Guessing entropy: denoising desynchronization with static alignment (a) and CAE (b).

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 typically 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.

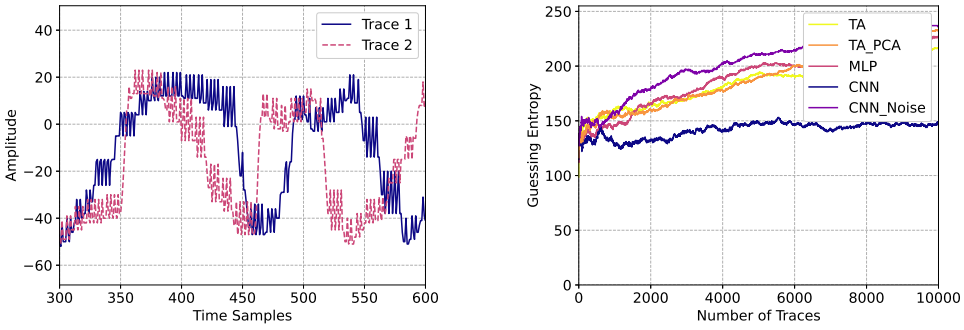
We simulate RDIs based on the Floating Mean method (with parameters a=5 and b=3) introduced in [CK09]. The RDIs implemented in such a way can provide more variance to the traces when compared with the uniform RDI distribution. To further increase the randomness of the injected RDIs, a random number (uniformly distributed between 0 and 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 RDIs in each feature equals 50%. Moreover, in real implementations, instructions, such as *nop*, are used to generate the random delay. This implementation will introduce specific patterns, such as peaks, in the power traces whenever a random delay occurs. 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. The manipulated traces are then padded with zero to keep the traces the same length.

A zoom-in view of two example traces with random RDIs is shown in Figure 5a. The number of injected RDIs can be obtained by counting the number of peaks. From the

Algorithm 3 Add Random Delay Interrupts.

```
function ADD_RDIS(traces, a, b, rdi_amplitude)
    new_trace ← []                                   ▷ container for new trace
    i ← 0
    while i < len(trace) do
        new_trace[i] ← new_trace[i].append(trace[i])
        rdi_occurrence ←randomNumber(0, threshold * 2)
        if rdi_occurrence > threshold then
            m ←randomNumber(0, a − b)
            rdi_num ←randomNumber(m, m + b)          ▷ number of RDIs to be added
            j ← 0
            while j < rdi_num do                     ▷ add RDIs to the trace
                new_trace[i] ← new_trace[i].append(trace[i])
                new_trace[i] ← new_trace[i].append(trace[i] + rdi_amplitude)
                new_trace[i] ← new_trace[i].append(trace[i + 1])
                j ← j + 1
        i ← i + 1
    return new_trace
```

traces, we observe that more randomness was introduced locally to the traces compared to the traces with desynchronization, which further influenced the attack result of guessing entropy. From Figure 5b, the best correct key rank of the traces with RDIs is 147 when using 10 000 traces, indicating that even the CNN_{best} model (with or without adding noise to the input layer) 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. Note that CNN's performance (with or without added noise) drastically differs from that reported in [KPH+19]. There are several possible reasons for such a difference: 1) we implement more difficult RDIs countermeasure, 2) we do not use as deep CNN architecture, and 3) we do not conduct a detailed tuning of the noise level when considering CNN with noise at the input. TA and TA with PCA perform similarly and do not converge.

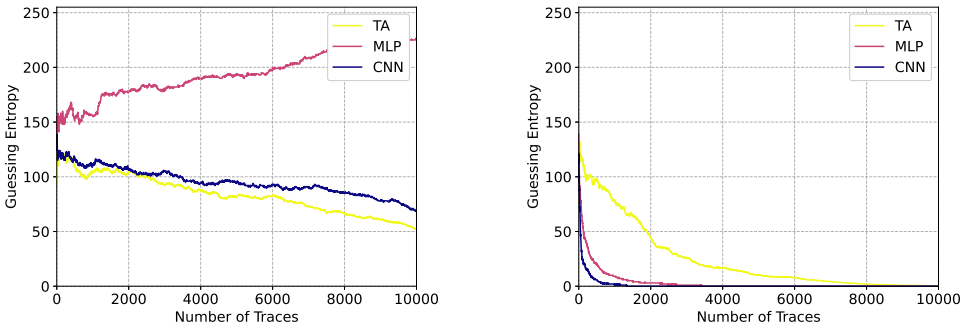


(a) RDIs: zoom-in view.

(b) RDIs: guessing entropy.

Figure 5: RDIs: demonstration and its influence on guessing entropy.

The attack results with the frequency analysis (FA) and CAE are shown in Figures 6a and 6b. With FA, GE slowly decreases when using CNN and TA for the attack, while the key rank reaches 52 for TA's best case. On the other hand, the effect of RDIs 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. Note the attack results with CNN and MLP are close to the one with the original dataset. Therefore, we can conclude that CAE can effectively recover the original traces



(a) GE: frequency analysis.

(b) GE: denoised with CAE.

Figure 6: Guessing entropy: denoise Random Delay Interrupts with frequency analysis vs CAE.

from the noisy traces with RDIs countermeasure.

4.5 Clock Jitters

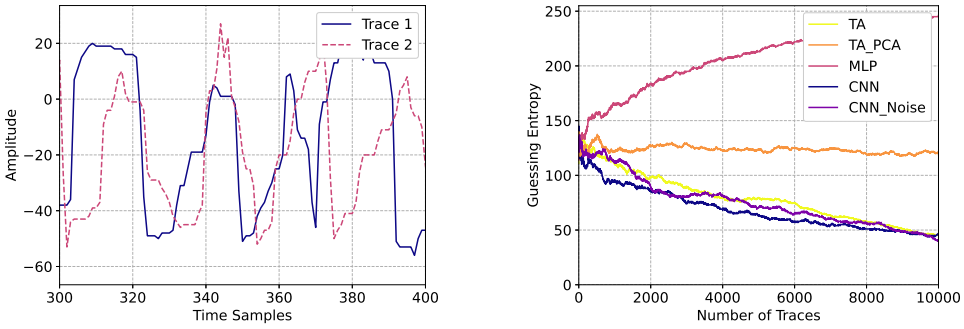
Clock jitters is a classical hardware countermeasure against side-channel attacks, realized by introducing the instability in the clock [CDP17]. Comparable to the Gaussian noise that introduces randomness to every point in the amplitude domain, the clock jitters increase the randomness for each point in the time domain. The accumulation of the deforming effect increases the misalignment of the traces and decreases the intermediate data correlation. 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 larger 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. The manipulated traces are then padded with zero to keep the traces the same length.

Algorithm 4 Add Clock Jitters.

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

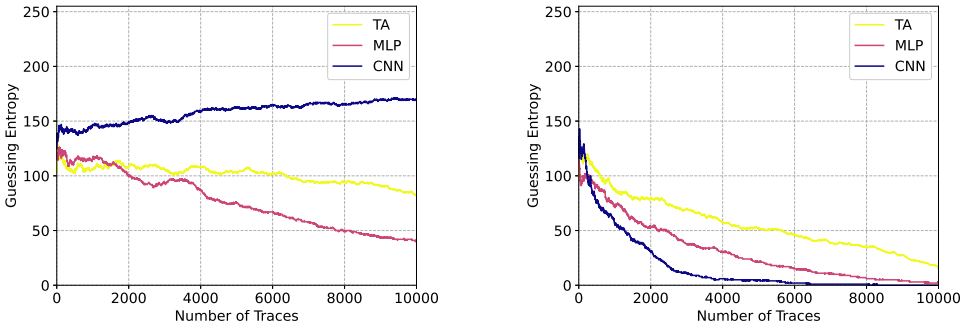
Zoom-in viewed traces with clock jitters are shown in Figure 7a. From Figure 7b, it is clear that no classifiers are successful in retrieving the key with 10 000 attack traces. The best results are achieved for CNN (with and without noise), followed closely by TA. A

comparison of the attack results for FA and denoised traces with CAE is shown in Figure 8. Like the previous attack results with the RDIs countermeasure, FA cannot 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 7: Clock Jitters: demonstration and its influence on guessing entropy.



(a) GE: frequency analysis. **(b)** GE: denoised with CAE.

Figure 8: Guessing entropy: denoising clock jitters with frequency analysis vs CAE.

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 related to 16 S-box accesses and then cluster them into 16 groups. Next, for 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]

Note that shuffling does not change the shape of the traces dramatically, so we do not demonstrate the shape of the traces here. Figure 9 shows the attack results for the

[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.

Algorithm 5 Add shuffling.

```
1: function ADD_SHUFFLING(trace, sbox_seg)
2:     new_trace ← []                                   ▷ container for new trace
3:     i ← 0
4:     while i < len(trace) do
5:         sbox_idx ←randomNumber(3, 16)
6:         new_trace[i] ← traces[i].replace(sbox_seg[sbox_idx])   ▷ replace sboxs
7:         i ← i + 1
8:     return new_trace
```

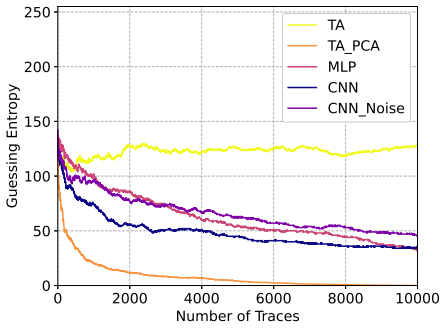
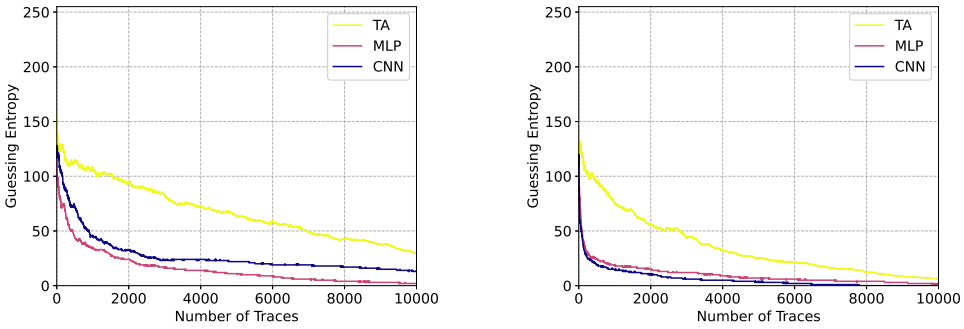


Figure 9: Shuffling: guessing entropy.

shuffling countermeasure. PCA-based TA shows the best performance with 9 885 attack traces required to reach the correct key. Compared with the baseline traces, the shuffling countermeasure increases the attack difficulty. Although GE is slowly converging for all attack methods except TA, (rank 32 for the best case with PCA-based TA), none of the attacks reach GE equal to 0 within 10 000 traces.

The results improve when we use additional 10 000 traces for profiling. As shown in Figure 10a, 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 10b), as only 7 754 traces are needed for the correct key when using CNN (MLP behaves only marginally worse). TA reaches rank equal to 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.



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

(b) GE: denoised with CAE.

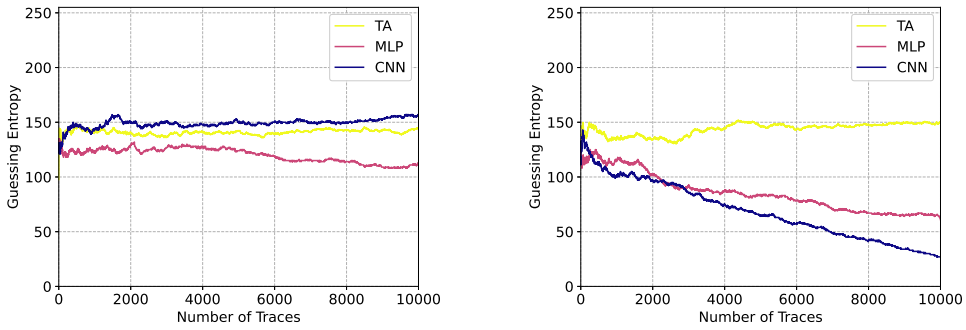
Figure 10: Guessing entropy: denoising shuffling by applying more traces (a) or CAE (b)

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

4.7 Combining the Effects of Gaussian Noise and Countermeasures

In the previous section, we add and denoise different types of noise individually. Next, we investigate an extreme situation by adding all five noise/countermeasures discussed in the previous section and verifying the CAE approach's effectiveness. To maximize the effectiveness of each type of noise and 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 proposed CAE's performance 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, use averaging after frequency analysis as then, we do not have enough measurements to conduct a successful attack.

Like the previous sections' procedure, we calculated the GE of the noisy and denoised traces and made a comparison. A demonstration of the manipulated traces with all types of noise and countermeasures is presented in Figure 18, Appendix B. As expected, the attack methods used in the paper cannot obtain the correct key within 10 000 traces. More precisely, the noisy traces do not converge with the increasing number of traces.



(a) GE: frequency analysis.

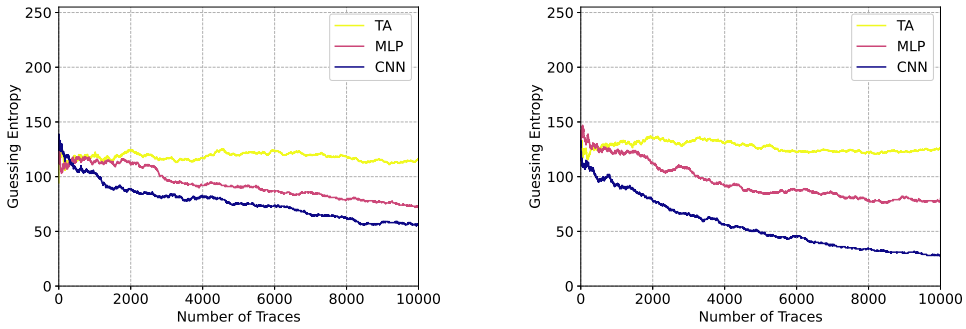
(b) GE: denoised with CAE.

Figure 11: Guessing entropy: denoising combined noise with frequency analysis (a) and CAE (b).

As shown in Figure 11a, FA is not working when dealing with the combination of noise and countermeasures (which is not surprising as we now use noise sources where this technique is insufficient). The GE of denoised traces with CAE (Figure 11b), on the other hand, reaches 27 with 10 000 traces when using CNN. Somewhat worse is MLP, and it reaches rank 61 after 10 000 traces. 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 countermeasures.

4.7.1 AES with Random Keys

Finally, we verify the CAE's performance 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 random but known keys, then use the trained model to attack the leakages and try to retrieve the unknown key. In terms of attack settings, there are 1 400 features in every trace. The attacked intermediate data is kept the same.



(a) GE: frequency analysis.

(b) GE: denoised with CAE

Figure 12: Guessing entropy: denoising combined noise with frequency analysis vs CAE.

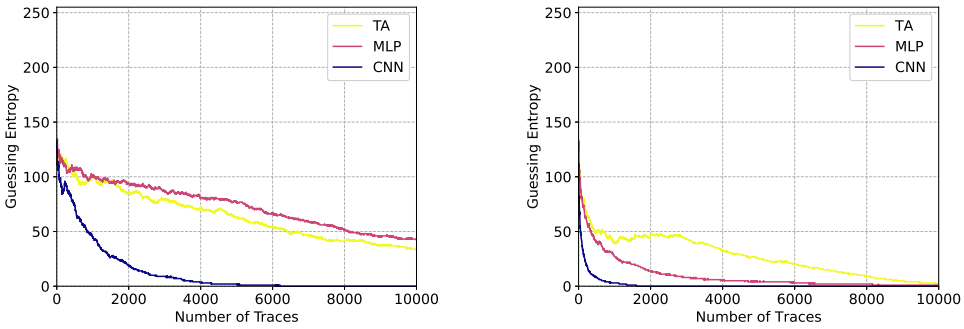
From the attack results, the GE of the noisy traces fluctuates above 100 regardless of the number of traces (Figure 19, Appendix B). On the other hand, guessing entropy indicates improved performance as a result of FA and CAE. For the best cases shown in Figure 12, GE value converges to 56 with 10 000 traces with FA and CNN, and 28 with CAE and CNN. Finally, we conclude that the proposed CAE can denoise the leakage in fixed and random key scenarios where the results are especially good when using CNNs as the attack mechanism.

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 black-box settings' countermeasures. Fortunately, CAE can denoise the traces even when the reference traces are not entirely clean. 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. The traces denoised by averaging (for Gaussian noise) and static alignment (for desynchronization) are used as the "clean" traces at the CAE output to handle the noisy traces. To quantify the remaining noise, CNN-based attacks were performed on these traces. There, 901 traces are required for realigned traces and 1 054 traces for averaged traces. Compared with the original traces with 831 attack traces, the traces denoised by classical methods are not perfectly denoised; one could expect a larger deviation of the attack traces value with simpler attack methods such as TA and MLP. 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 13.

Compared with the denoised traces using the original clean traces as the reference, the noise-to-less-noise cases' attack performance is degraded. Specifically, 8 751 traces are required to retrieve the correct key when using the clean traces to denoise the Gaussian noise (white-box setting), while this reduces to 6 073 when denoised with averaged traces,



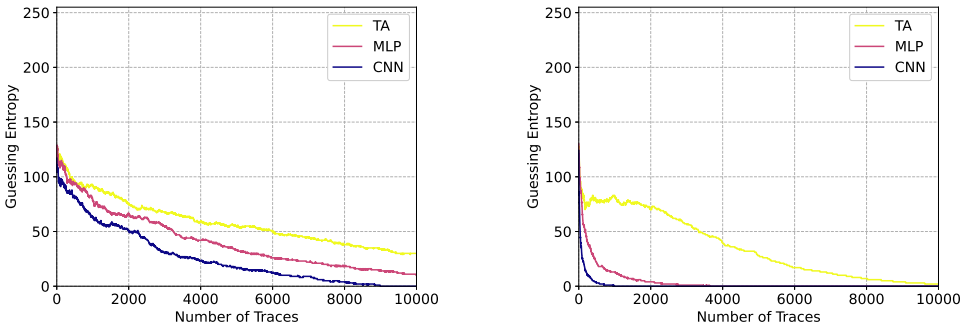
(a) GE (Gaussian noise): train CAE from noisy to averaged traces.

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

Figure 13: Guessing entropy: denoising Gaussian noise/desynchronization from less noisy traces.

indicating that the averaged traces contain even less noise than the original clean traces. The attack performance degradation is different when 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 better 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 CAE's denoising performance can be improved with cleaner traces (e.g., more traces for averaging).

Finally, we denoised the traces with Gaussian noise and desynchronization in a combined setting. More precisely, 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 14.



(a) GE (Gaussian noise).

(b) GE (desynchronization).

Figure 14: Guessing entropy: denoised Gaussian noise and desynchronization by combined training of CAE.

The joint training method leads to comparable (or even better) performance than the previous results on a single noise source. To be specific, 8 989 traces are needed for the Gaussian noise and 942 for the desynchronization. This result again shows that the 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

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, uniform 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. We consider two types of leakage traces (one encrypted with fixed and another with random keys) and three attacks (CNN, MLP, and TA). It is interesting to note that in our experiments, adding noise to the input of CNN does not provide as beneficial results as reported in [KPH+19]. Still, note that our CNN architecture is much simpler and does not work as well as the architecture presented by Kim et al. Consequently, it is not surprising that noise addition does not work as well, and that we require less noise at the input. The obtained results show that the proposed CAE can remove/reduce the noise and determine the underlying ground truth and 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. Our results show that autoencoders reliably remove noise/countermeasures even if the measurements do not contain all the noise sources the autoencoder used in the training process.

Denoising autoencoder provides an attacker with a powerful tool to pre-process the traces. We expect this technique could help with problems like portability [BCH+19]. There, the biggest obstacle stems from the variance among different devices. These variances introduce the trace variation, making the attack model generated for one device challenging to transfer to another. 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. This setting is similar to the scenario with added Gaussian noise, which indicates that the CAE approach should be very useful in portability. Additionally, the trained denoising autoencoder could be used for transfer learning. Then, the encoder part of the autoencoder could be further trained and used to launch attacks. Finally, we plan to investigate whether denoising autoencoder could also work for the masking countermeasures.

Availability

Implementations for reproducing our results are available at `https://github.com/AISyLab/Denoising-autoencoder`.

Acknowledgement

We thank Jorai Rijsdijk and anonymous reviewers for their valuable comments. We thank the shepherd for the suggestions on how to improve the paper.

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.

[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.

[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.

[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.

[SY14] Mayu Sakurada and Takehisa Yairi. Anomaly detection using autoencoders with nonlinear dimensionality reduction. In *Proceedings of the MLSDA 2014 2nd Workshop on Machine Learning for Sensory Data Analysis*, pages 4–11, 2014.

[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.

[WEG87] Svante Wold, Kim Esbensen, and Paul Geladi. Principal component analysis. *Chemometrics and intelligent laboratory systems*, 2(1-3):37–52, 1987.

[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 Attack Methods

A.1 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.

A.2 Multilayer Perceptron

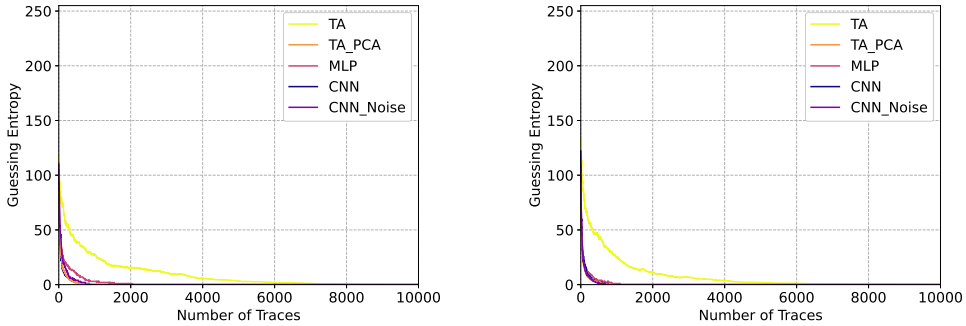
The multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs [GD98]. MLP consists of multiple layers of nodes in a directed graph, where each layer is fully connected to the next one, and training of the network is done with the backpropagation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

A.3 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.

B Additional Results

B.1 Denoising the "Clean" Traces



(a) GE: Original traces.

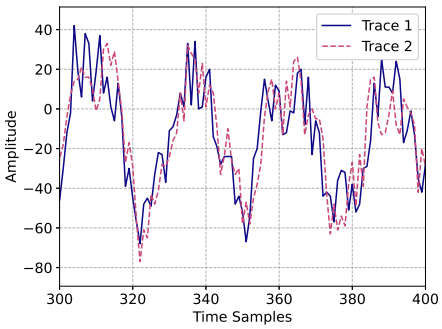
(b) GE: traces denoised by CAE.

Figure 15: Guessing entropy: original traces vs "cleaned" traces by CAE.

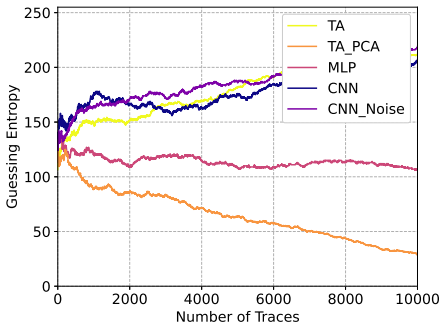
B.2 Uniform Noise

Besides analyzing the denoising performance with Gaussian noise, we also consider the uniform noise. A uniform-distributed random values ranging from -20 to 20 are added to each point of the trace to simulate the uniform noise. The pseudocode for constructing traces with uniform noise is shown in Algorithm 6. An example of the zoom-in view of two manipulated traces is shown in Figure 16a; the attack results are shown in Figure 16b. Similar to Gaussian noise, PCA-based TA performs the best with the correct key ranking reaching 29. We also observe that adding noise to the input of CNN improves the attack performance. Still, the uniform noise significantly increases the difficulties in obtaining the correct key (Figure 16).

Next, we denoise the traces with averaging as well as CAE. The GE of denoised traces with 10-trace averaging and CAE are shown in Figures 17a and 17b, respectively. From the attack perspective, GE converges in both denoising cases when the number of trace increases: 3 584 averaged traces or 4 880 denoised traces are sufficient to reach GE of 0. Following this observation, we again confirm that trace averaging is a successful method in removing the uniform noise. Additionally, TA is better than MLP in dealing with uniform noise. Indeed, TA is a generic method that follows Bayes' theorem and is resilient to the noise's interference. As the noise still exists in the denoised traces, we believe that the



(a) Uniform noise: zoom-in view.



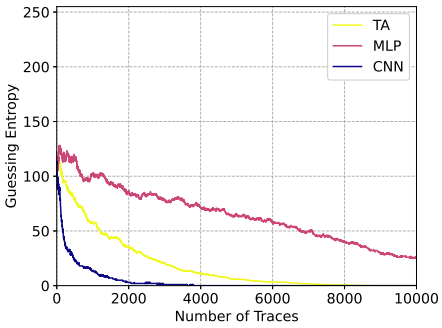
(b) Uniform noise: guessing entropy.

Figure 16: Uniform noise: demonstration and its influence on guessing entropy.

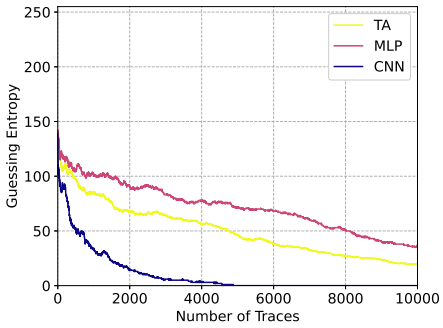
Algorithm 6 Add Uniform Noise.

```
1: function ADD_UNIFORM_NOISE(trace, range)
2:     new_trace ← []                          ▷ container for new trace
3:     i ← 0
4:     while i < len(trace) do
5:         level ←randomNumber(−range, range)
6:         new_trace[i] ← traces[i] + level    ▷ add noise to the trace
7:         i ← i + 1
8:     return new_trace
```

MLP model we used is less robust than TA in dealing with fluctuation from the amplitude level. Compared with the denoising performance with the Gaussian noise, the uniform noise seems easier to counteract.



(a) GE: denoised with averaging.



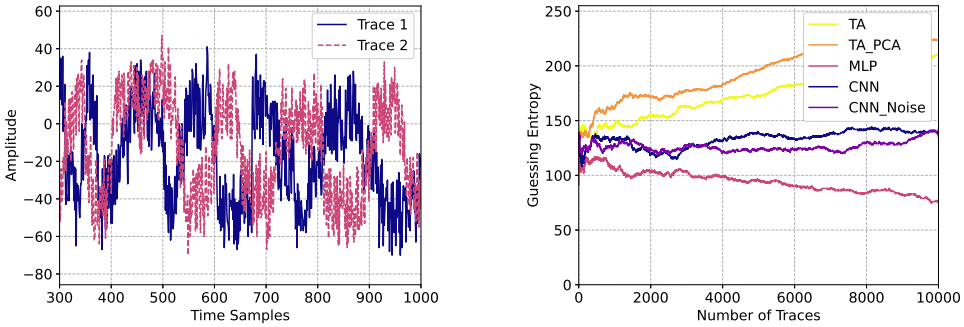
(b) GE: denoised with CAE.

Figure 17: Guessing entropy: denoise uniform noise with averaging vs CAE.

B.3 Results With Combined Noise

In Figure 18, we depict results for the AES with the fixed key where five noise sources are combined. Observe that 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 19). 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 CNN performs the best. Finally, the best performing algorithm's

key rank is slightly lower when considering a 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 18: All in one (fixed key): demonstration and its influence on guessing entropy.

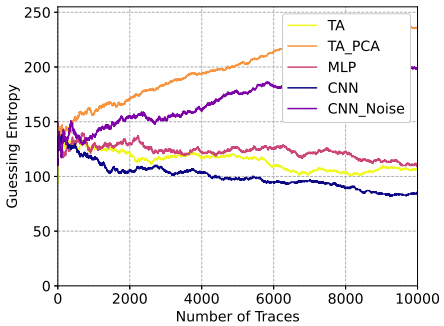


Figure 19: All in one (random key): guessing entropy.