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. What is more, various countermeasures make attacks significantly more difficult, and those countermeasures can be further combined to make the attacks even more challenging. From the other side, to improve the performance of attacks, an intuitive solution 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 four 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, Hiding, 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 and sometimes even useless [BCO04].

In some cases, the low signal-to-noise ratio of the leakage increases the difficulties of identifying these patterns. Additionally, there are various countermeasures in both hardware and software that make the attacks even more difficult. Such 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. 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., 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 the 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 be separated perfectly if we do not know their characterizations. In addition, in realistic settings, we must also consider the portability and the differences among various devices [BCH+19]. The combination of all these factors makes this problem very complex, 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 using low-pass filters [WLL+18], conducting trace alignments [TGWC18, TGWC18], and various feature engineering methods [ZZY+15, PHJB19]. More recently, the SCA community used deep learning techniques that can conduct implicit feature selection and fight countermeasures [CDP17, KPH+19, ZBHV19]. While such techniques are useful, they are usually aimed either against a single source of noise or in cases when they can handle more sources of noise, they do not offer 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 point). Finally, we emphasize that being able to reduce the noise comprehensively could bring several advantages 1) understanding the attack techniques better, 2) understanding the noise better and consequently, (hopefully) being able to design stronger countermeasures, 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, and clock jitters. 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.

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 most interest in techniques like random forest [LMBM13, HPGM16] and support vector machines [HZ12, PHJ+17]. More recently, multilayer perceptron [GHO15, PHJ+19] and convolutional neural networks [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 not noise removal, and they report a poor performance when compared to CNNs [MPP16].

1.2 Our Contributions

In this paper, 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.

To conduct experimental analysis, we consider four separate sources of noise or their combination. We investigate the performance of both multilayer perceptron and convolutional neural networks for classification after noise removal. Finally, we experiment with several different noise levels, as well as datasets, having either fixed or random keys, to show the universality of our approach.

This paper is organized as follows. In Section 2, we provide details about profiled SCAs, machine learning techniques we use, and the dataset we investigate. In Section 3, we provide details about denoising autoencoders and their convolutional version. Section 4 gives experimental results when considering the effects of noise either separately or combined. Finally, in Section 5, we conclude the paper and present possible future research directions.

2 Background

In this section, we start by introducing the notation we follow. Afterward, we discuss profiled side-channel analysis and neural networks. Finally, we give details about the ASCAD dataset we use in the rest of the paper.

2.1 Notation

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 leakages/traces (with noise and countermeasures) as noisy leakages/traces while the unprotected leakages are denoted as clean leakages/traces.

2.2 Profiled Side-channel Analysis

In the context of implementation attacks, they target physical leakages from the insecure implementation of otherwise theoretically secure cryptographic algorithms. 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 power 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. 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).

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.

2.3 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.

2.3.1 Multilayer Perceptron

The multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs. MLP consists of multiple layers (at least three - one input layer, one output layer, and one or more hidden 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.

2.3.2 Convolutional Neural Networks

Convolutional neural networks (CNNs) are a type of neural network originally designed for 2-dimensional convolutions as inspired by the biological processes of animals' visual cortex [LB+95]. They are commonly used for image classification, but in recent years, they have shown their strengths for time series data [ODZ+16, CFSC17], speech [PC+15], but also security applications [SIJW18].

CNNs resemble ordinary neural networks (e.g., multilayer perceptron) from the architecture perspective: they consist of several layers where each layer is made of neurons. CNN usually consists of three types of layers: convolutional layers, pooling layers, and

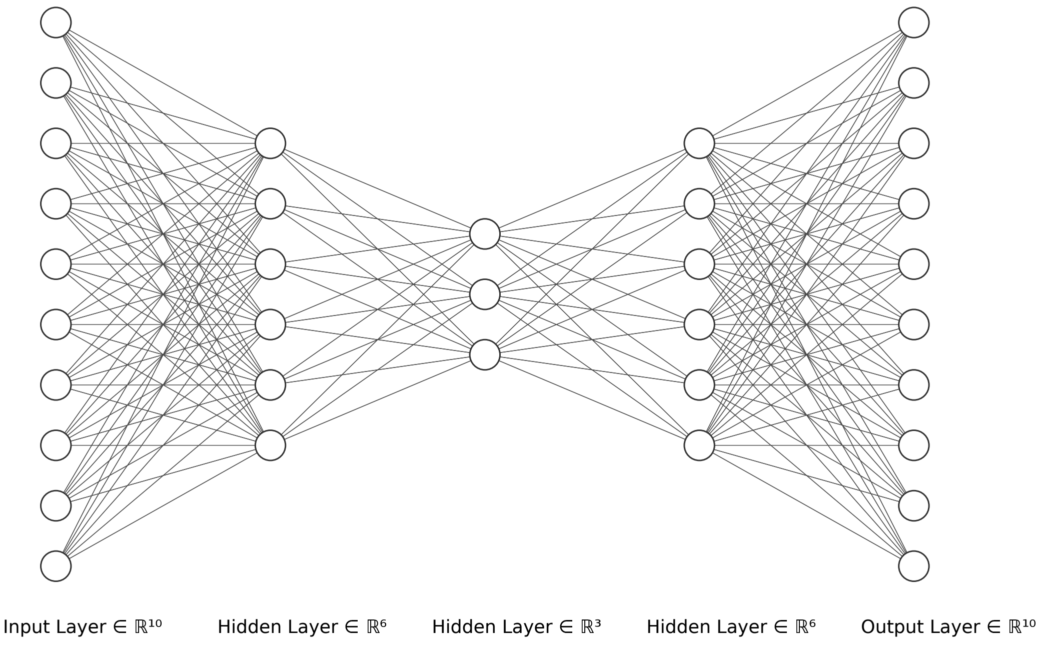


Figure 1: An example of an autoencoder network with three hidden layers (created with NN-SVG [LeN19]).

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.

2.3.3 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 aims to transform inputs into outputs with the least possible amount of distortion [Bal12]. Benefiting from its unsupervised learning characteristic, autoencoder has been used in many applications such as data compression [TSCH17], anomaly detection [AC15], and image recovery [Gon16]. The basic structure of an MLP-based autoencoder is given in Figure 1.

An autoencoder consists of two parts: encoder (ϕ) and decoder (ψ). Intuitively, the encoder squeezes the input with more features to its bottleneck with fewer features, while the goal of the decoder is to reverse this process. More precisely, the goal of the encoder is to transfer the input to its latent space $\mathcal{F}$, i.e., $\phi : \mathcal{X} \rightarrow \mathcal{F}$. The decoder, on the other hand, reconstructs the input from the latent space, which is equivalent to $\psi : \mathcal{F} \rightarrow \mathcal{X}$. 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.

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

When applying the autoencoder for the denoising purpose, the input and output are not identical but represent the noisy-clean data pairs. Ideally, a well-trained autoencoder can keep the most representative information (e.g., characteristics of the original data) in its bottleneck while neglecting the less important features (e.g., random noise). Consequently, the original data (without noise) can be recovered by feeding noisy data to the input of the autoencoder.

A similar idea can also be applied to remove the countermeasures from the leakage traces. Since the noise and countermeasures existing in the leakage are random in the amplitude or time domain, we could also consider them as noise. For example, we can refer to the white noise as the noise in the amplitude domain; the global jitters (desynchronization), random delay interrupts, and clock jitters, on the other hand, can be considered as the noise in the time domain. Since the autoencoder is good at extracting the important features while neglecting randomness, we use it to filter out the noise and recover the ground truth of the traces. Then, one can expect that with the recovered traces, the attack efficiency will be dramatically improved.

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 ATMega8515 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 data sets 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.

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.:

$$Y(i) = \texttt{Sbox}[(p[i] \oplus k[i])]. \tag{2}$$

Note that the model given in Eq. (2) does not leak information directly as it is first-order protected. Consequently, we do not state a model-based SNR. The SNR for the ASCAD dataset is ≈ 0.8 under the assumption we know the mask (shown in Figure 5a) while it is almost 0 with the unknown mask.

3 Denoising with Convolutional Autoencoder

In this section, we discuss the attacker model and the way to obtain clean/noisy traces. Afterward, we give details about the convolutional autoencoder architecture used in our experiments.

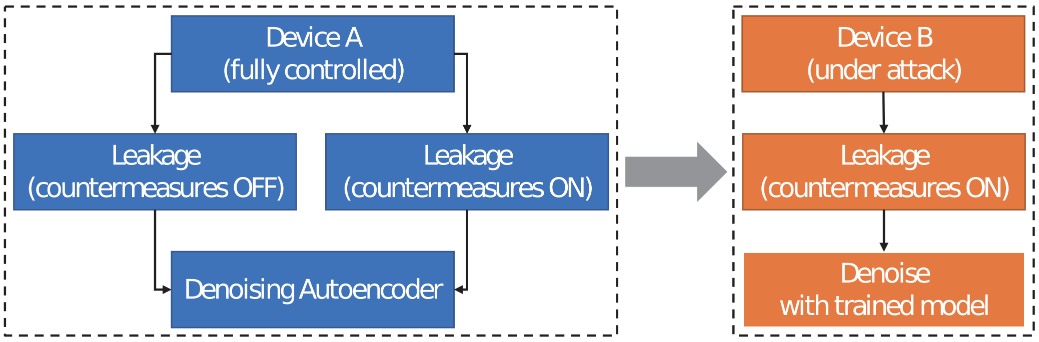


Figure 2: Denoising strategy.

3.1 Denoising Strategy

As discussed in Section 2.3.3, noisy-clean trace pairs are required to train a denoising autoencoder. Inspired by the profiled side-channel analysis method, we devise a denoising strategy shown in Figure 2.

We assume an attacker has 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 several 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.

3.2 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). Since related works indicate that CNN outperforms MLP in dealing with the side-channel leakages [MPP16,PHJ$^+$19], we use the convolution layer as the basic element for denoising purpose.

To maximize the denoising ability of the proposed architecture, we tune the hyperparameters by evaluating the CAE performance towards different types of noise. The tuning range and selected hyperparameters are shown in Table 1. We use the *SeLU* activation function to avoid vanishing and exploding gradient problems [KUMH17] and *He Uniform* initialization to improve weight initialization [HZRS15].

Table 1: CAE hyperparameter tuning.

Hyperparameter	**Range**	**Selected**
Optimizer	Adam, RMSProb, SGD	RMSProb
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	1 000, 2 000	2 000

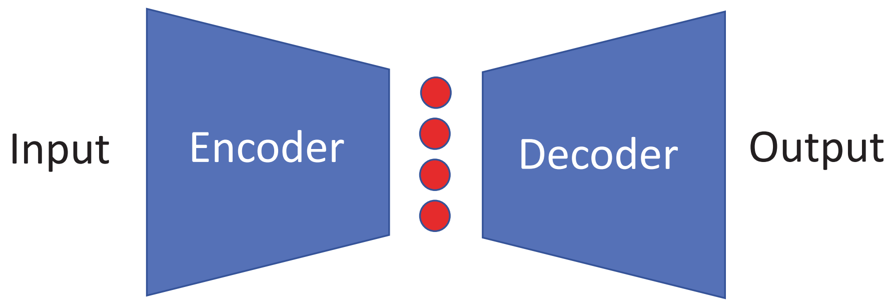


Figure 3: High-level CAE structure

In terms of the autoencoder architecture, we observed that when dealing with trace desynchronization, an autoencoder that has 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, 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.

A high-level CAE structure is shown in Figure 3. The convolutional encoder converts the leakages to its latent representation by passing through multiple convolution blocks. Then, these compressed high-dimensional features are further processed through a fully-connected layer, which is then reshaped to the convolution size and reconstructed to the target traces. Note that the size of the latent space (represented by four red circles) 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}:

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

S_{clean} is the size of the target clean traces, S_{pooli} 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. ‘

We emphasize that from an attacker 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.

Table 2: CAE architecture.

Block/Layer	**Filter size**	**Filter number**	**Pooling stride**
Conv block * 2	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 * 2	2	256	0
Deconv block	2	1	0

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 four types of noise/countermeasures that commonly exist in the devices: Gaussian noise, desynchronization (misalignment), random delay interrupts (RDI), and clock jitters with different noise levels. The simulation approaches are based on previous researches as well as the observation or implementation of the real devices. The pseudocode for the noise simulation is available in the Appendix C.

There is a question of whether the denoising procedure can help regardless of the noise amount. In our experiments, we show the results where the denoising architecture is able to reduce the GE to 0 (or close value) within 5 000 attack traces. Higher noise levels are still affected by CAE, but we require more measurements to reach GE of close to 0. As expected, smaller noise levels are simpler for CAE, and we can reach GE of 0 with even fewer attack traces. We demonstrate this in Appendix A. Finally, we consider what would happen if one applies denoising autoencoder to measurements that do not have noise. In Appendix B, we show that in such a case, a part of the useful signal is removed (as expected), but the attack still works, albeit somewhat worse.

To compare the denoising performance of CAE with the existing techniques commonly used by attackers, we also evaluate the denoising performance of trace averaging. Here, ten traces are averaged to obtain a single trace.

Throughout the experiments, we use the ASCAD dataset (with fixed key and random key) as the training datasets. Note that the ASCAD dataset is masked and there is no first-order leakage. To obtain an intuition of the properties of the reconstructed traces, we evaluate the SNR of the masked S-box output (Eq. (4)).

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

The SNR is sometimes named F-Test to refer to its original introduction in [Fis22]. For a noisy observation L_t at each time sample t of an event Z, it is defined in Eq. (5). The numerator denotes the signal magnitude and the denominator denotes the noise magnitude estimation.

$$\mathrm{SNR}\,[t] \triangleq \frac{\mathrm{Var}\,\mathbb{E}[L_t|Z]}{\mathbb{E}\,\mathrm{Var}[L_t|Z]}. \tag{5}$$

Two models, CNN_best and MLP_best given in the ASCAD paper [BPS+18], are used to attack the traces. The architectures are listed in Table 3 and Table 4. Additionally, the average pooling layer is used in the Conv block for CNN architecture. In this work, 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].

Table 3: CNN architecture used for attacking.

Layer	**Filter size**	**Filter number**	**Pooling stride**	**Neuron number**
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

Table 4: MLP architecture used for attacking.

Layer	**Neuron number**
Fully-connected *4	200

Finally, the selected hyperparameters for both deep learning models are shown in Table 5. Since these two models have different complexities, we can evaluate the denoising performance in a fair way. We emphasize that we do not aim to find the best attack models but to show how denoising autoencoders can help improving performance for 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.

Table 5: Hyperparameters for MLP and CNN classifiers.

Hyperparameter	**Value**
Optimizer	RMSProb
Weight initialization	Uniform distribution
Activation function	ReLU
Learning Rate	1e-5
Batch size	200
Epochs	100 (CNN) / 500 (MLP)
Training sets	35 000
Validation sets	5 000

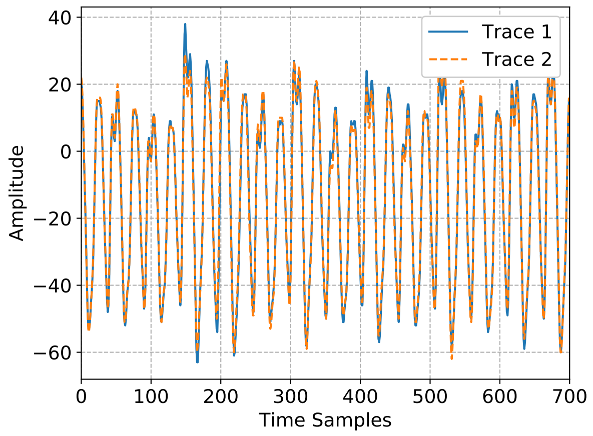
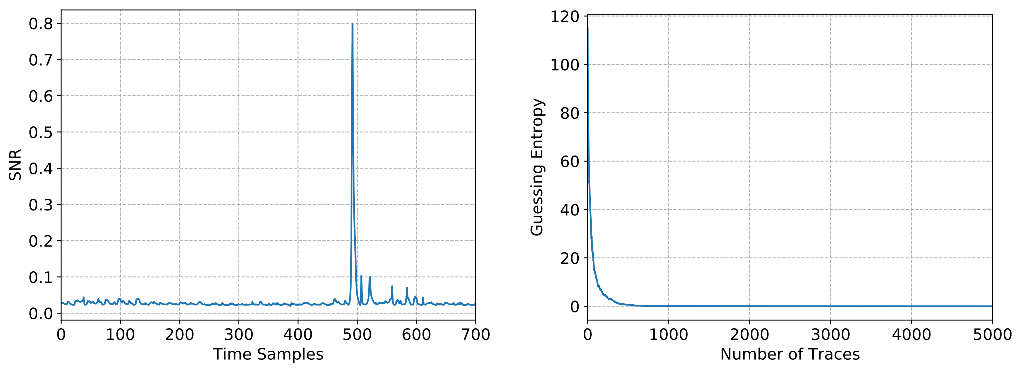


Figure 4: Baseline: example traces.

4.1 Baseline

We first attack the synchronized ASCAD dataset (ASCAD.h5) as the baseline in comparison with different scenarios. The detailed attack approach can be found in the original paper [BPS+18]. An example of two traces to be attacked is presented in Figure 4. From the comparison of those two traces, we see that most of the parts are identical, but there are still some variations (especially on the peaks).

Here, we use the CNN_best model for the attack. The SNR of the masked S-box output and the resulting guessing entropy are presented in Figure 5.



(a) SNR: masked S-box output.

(b) Guessing entropy.

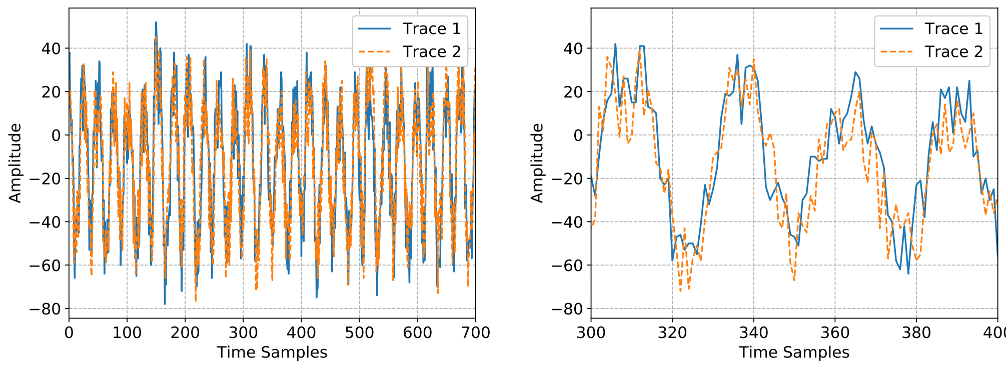
Figure 5: Baseline: SNR and guessing entropy.

Without noise and countermeasures in the traces, guessing entropy reaches zero quickly after 405 traces, indicating that the real key has been successfully predicted. The outcomes of this attack are used as the baseline and the following attacks on noisy and denoised traces are compared based on these results.

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. The noise can also be artificially 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.

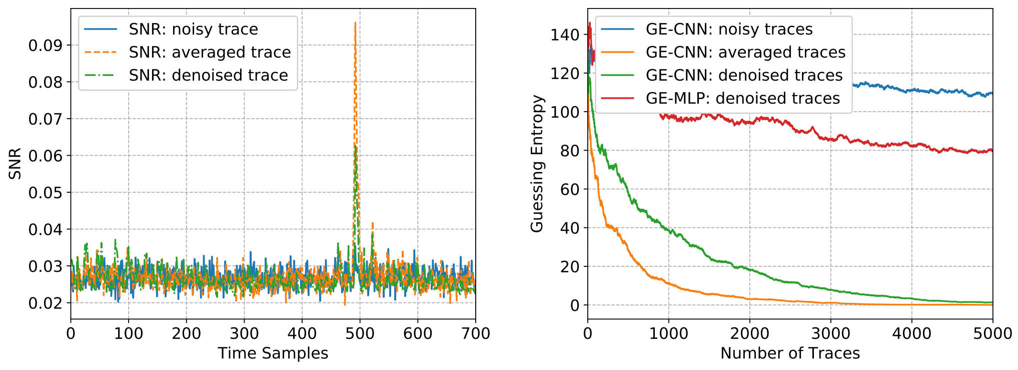
To demonstrate the influence of the Gaussian noise, we add a uniformly distributed random value between -20 to 20 to each point of the trace. An example of the manipulated trace and its zoom-in view is shown in Figure 6. Compared with the baseline traces, the Gaussian noise significantly distorted the shape of the original traces in the amplitude domain.



(a) Gaussian noise: example traces. (b) Gaussian noise: zoom-in view.

Figure 6: Gaussian noise: example traces and its zoom-in view.

Next, we denoise the Gaussian noise with trace averaging as well as CAE proposed in this paper. The SNR of the noisy and denoised traces are shown in Figure 7a, while GE values after deep learning attacks are shown in Figure 7b.



(a) SNR. (b) Guessing entropy.

Figure 7: Noisy and denoised traces (Gaussian noise): SNR and guessing entropy.

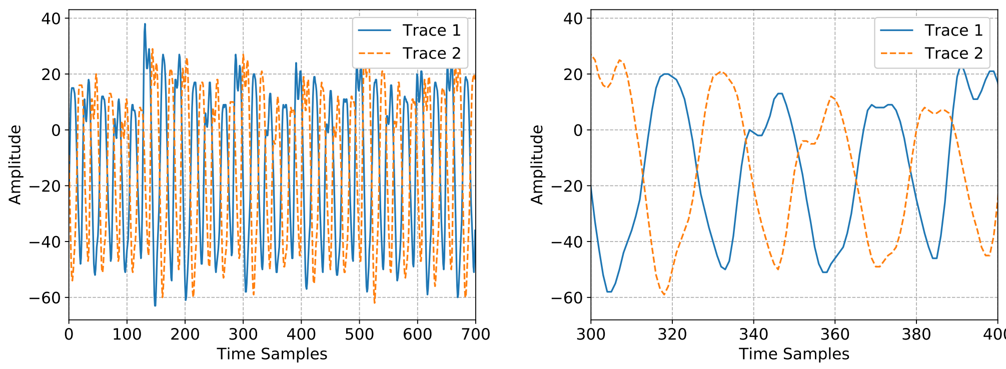
For the baseline attack with “clean” traces, the existence of the Gaussian noise reduces the leakage of the attacked intermediate data: the SNR drops from 0.8 to 0.03. The SNR of denoised traces clearly shows that the CAE has filtered out a part of the Gaussian noise, as the SNR peak of the denoised traces is more than three times higher than the noisy one.

From the attack (GE) perspective, GE converges in both cases when the number of trace increases. For the noisy traces, 5 000 traces reduce GE from 121 to only 109 on average. When considering a multilayer perceptron, we observe that its behavior is better than CNN when no denoising is done. This indicates that the denoising autoencoder works 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. CNN attack performance after denoising with either averaging or denoising autoencoder is significantly improved over the noisy version: 5 000 traces are sufficient to reach GE of 0. It is worth noting that GE of averaged traces is slightly better than GE of CAE, proving that trace averaging is a good candidate in removing the Gaussian noise. Still, we can conclude that CAE can remove the Gaussian noise and consequently 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. 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.

We consider the desynchronization as a type of noise existing in the traces. Different from the Gaussian noise, the desynchronization noise 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 (ASCAD_desync50 [BPS+18]). An example of the traces with desynchronization is shown in Figure 8.

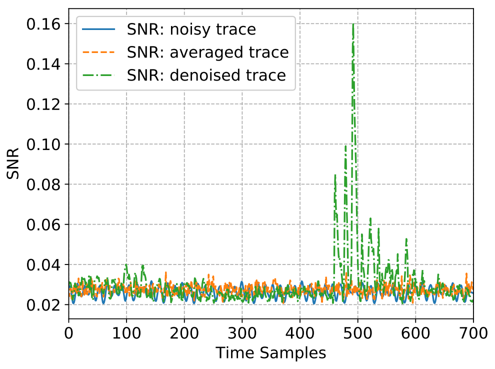


(a) Desynchronization: example traces. (b) Desynchronization: zoom-in view.

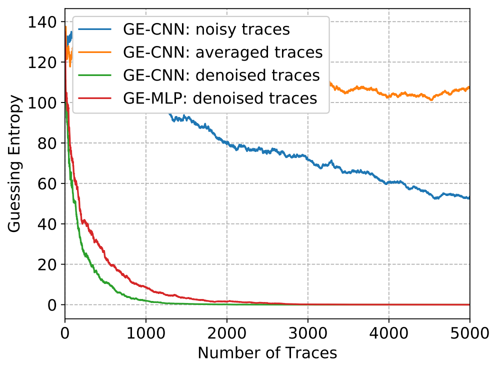
Figure 8: Desynchronization: example traces and its zoom-in view.

Next, we attack the misaligned traces as well as the denoised traces from CAE. Figure 9a shows that only CAE can increase the SNR of the intermediate data (0.16). The averaged traces, on the other hand, have similar SNR (0.03) with the noisy ones, indicating that traces averaging is ineffective in dealing with desynchronization.

As evident from Figure 9b, GE of the noisy traces converges faster than for the averaged traces. Consequently, we conclude that the averaging of the desynchronized traces is not



(a) SNR.



(b) Guessing entropy.

Figure 9: Noisy and denoised traces (desynchronization): SNR and guessing entropy.

helping to recover the traces. Indeed, the averaging of the point from different time locations further degrades the data correlation. On the other hand, CNN proves its ability to delimit the desynchronization effect, as GE converges to 53 with 5 000 traces when attacking the noisy traces. Still, considering that the original "clean" traces only needed 405 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.

CAE provides a simple alternative approach in synchronizing the traces. By training a CAE with desynchronized–synchronized traces pair, the model can align the traces automatically. Consequently, the number of required traces to retrieve the key reduces to 1 189 with CNN and 2 329 with MLP on average.

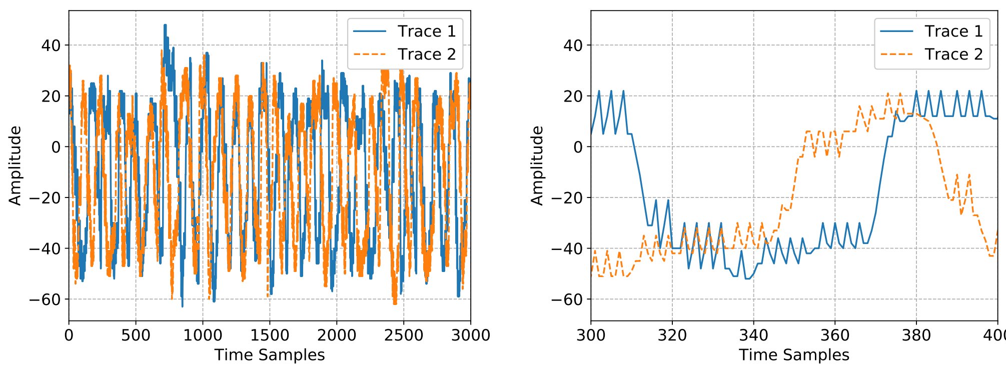
4.4 Random Delay Interrupts (RDIs)

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 parameter 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, thus further increasing the attack difficulty. The probability of the occurrence of RDIs is fixed to 50%. Moreover, 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, a specific pattern, such as peak, is shown in the power trace. We consider this effect by generating a small peak with a certain amplitude when injecting the random delays to the traces.

An example of the traces with RDIs with its zoom-in view is shown in Figure 10. We observe that more randomness was introduced locally to the traces when compared to the traces with desynchronization.

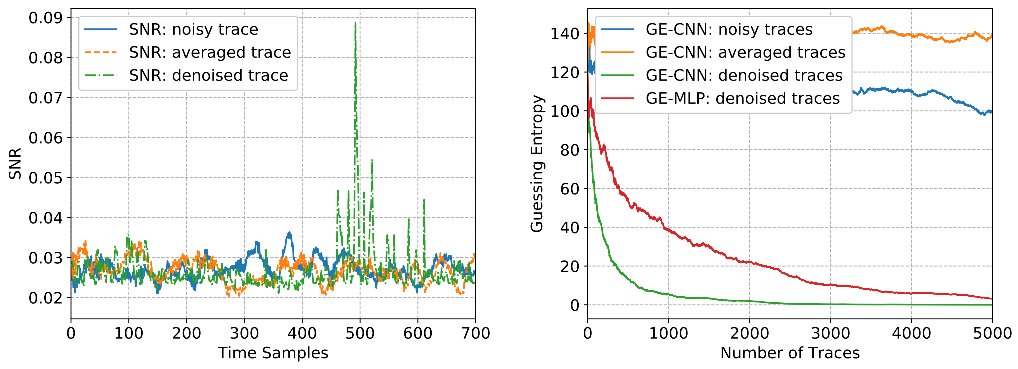
As a result, the SNR and rank of the original traces with RDIs and denoised traces with averaging and CAE are shown in Figure 11. The guessing entropy of the traces with RDIs converges slowly (129 to 98 with 5 000 traces), indicating that the CNN_best model we are using is not powerful enough to extract the useful patterns and retrieve the key with



(a) Random Delay Interrupts: example traces. (b) Random Delay Interrupts: zoom-in view.

Figure 10: Random Delay Interrupts: example traces and its zoom-in view.

5 000 traces. We can conclude that RDIs implemented in this way dramatically increase the attack difficulty.



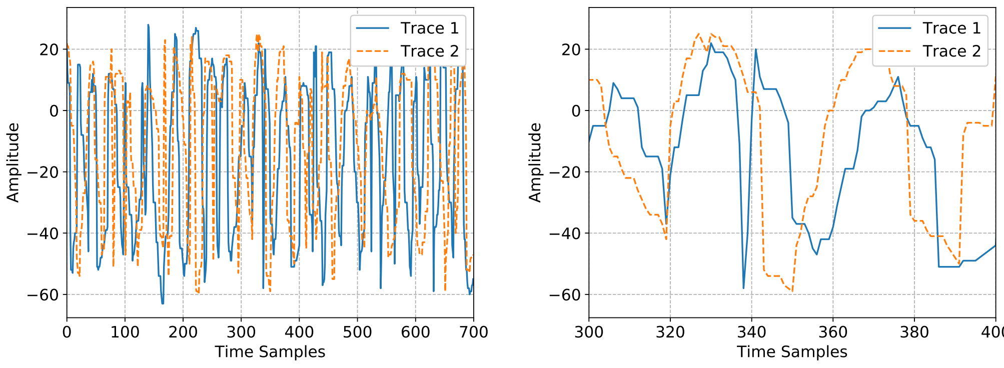
(a) SNR. (b) Guessing entropy.

Figure 11: Noisy and denoised traces (RDIs): SNR and guessing entropy.

Trace averaging is again not helpful to remove the countermeasure and the GE value fluctuates around 140. By applying the CAE, the effect of RDIs has been reduced dramatically: the SNR increases three times to 0.09 and GE converges significantly faster both when attacking with MLP and CNN. CNN performance is especially good as it needs only 2 264 traces on average to reach GE of 0 (while MLP requires around double the amount of traces). We can conclude that CAE can recover the original traces from the noisy traces with RDIs countermeasure.

4.5 Clock Jitters

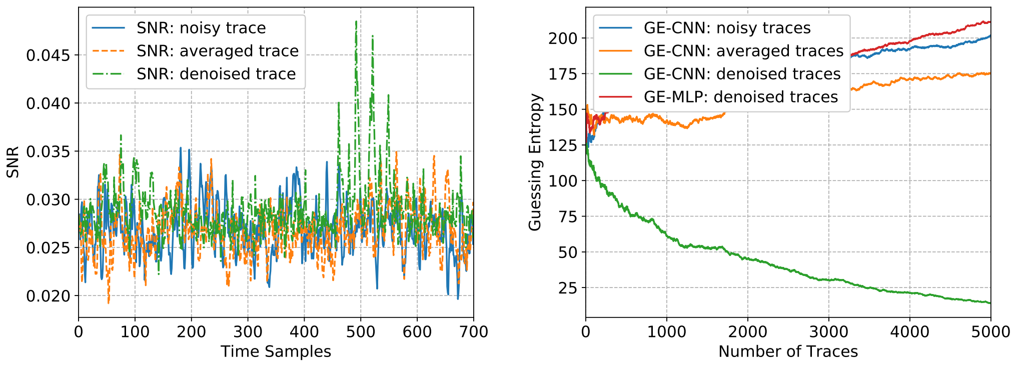
Clock jitters is a classical hardware countermeasure against side-channel attacks, realized by introducing the instability in the clock [CDP17]. Comparable to the Gaussian noise that introduces randomness to every point in the amplitude domain, the clock jitters increase the randomness for each point in the time domain. Indeed, 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. An example of the traces with clock jitters is shown in Figure 12.



(a) Clock Jitters: example traces. (b) Clock Jitters: zoom-in view.

Figure 12: Clock Jitters: example traces and its zoom-in view.

We denoise the clock jitters with CAE and a comparison of the attack results for the noisy and denoised traces with averaging and CAE is shown in Figure 13. As in the previous settings, traces averaging is not efficient in dealing with time-based noise and countermeasures. The proposed CAE, on the other hand, successfully reduces the effect of clock jitters. From the SNR perspective, although the denoised SNR is the lowest (0.044) for all four different types of noise, it still outperforms its noisy and averaged counterparts. When compared with noisy traces, the GE of the denoised traces with CNN converges to 13 after applying 5 000 traces[1].



(a) SNR. (b) Guessing entropy.

Figure 13: Noisy and denoised traces (clock jitters): SNR and guessing entropy.

To conclude, the proposed CAE proves its ability to delimit the effect of the Gaussian

[1]Note, we see that for other attacks, GE increases with the increase in the number of traces. We can invert the key guess to obtain the ranking as now the least likely guess would be correct. Still, CNN's performance after DAE is the best one

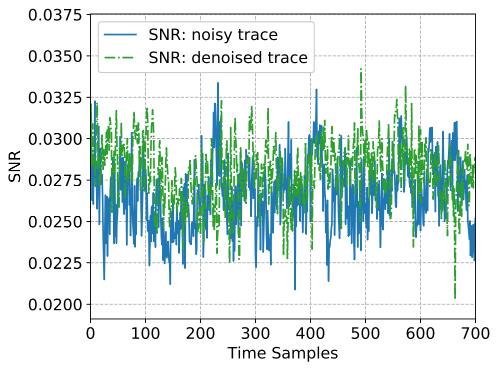
noise, desynchronization, random delay interrupts, and clock jitters. Traces averaging performs well in removing the Gaussian noise but is ineffective in dealing with the noise and countermeasures in the time domain. Denoising autoencoder works for both MLP and CNN attacks, but CNN's performance is better in comparison.

4.6 Combining the Effects of Gaussian Noise and Countermeasures

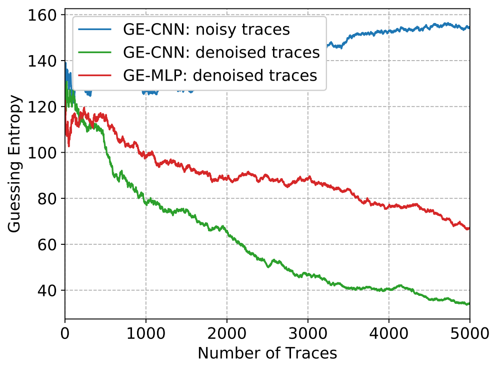
In the previous section, we add and denoise different types of noise individually. Now, we investigate a more realistic situation by adding all of the discussed noise types together and then verifying the effectiveness of the CAE approach. Here, we test two different datasets: AES with a fixed key and AES with random keys. Since trace alignment is proved to be inefficient in dealing with time-based noise and countermeasure, we only evaluate the SNR and GE of the noisy and denoised traces with CAE.

4.6.1 AES with the Fixed Key

Similar to the procedure of the previous sections, we calculated the SNR and GE of the noisy and denoised traces and made a comparison between them. The SNR comparison is presented in Figure 14a. Compared with the noisy traces, the SNR of denoised traces is increased slightly (0.0345). This observation indicates that the combination of the noise types degraded the performance of the CAE.



(a) SNR.



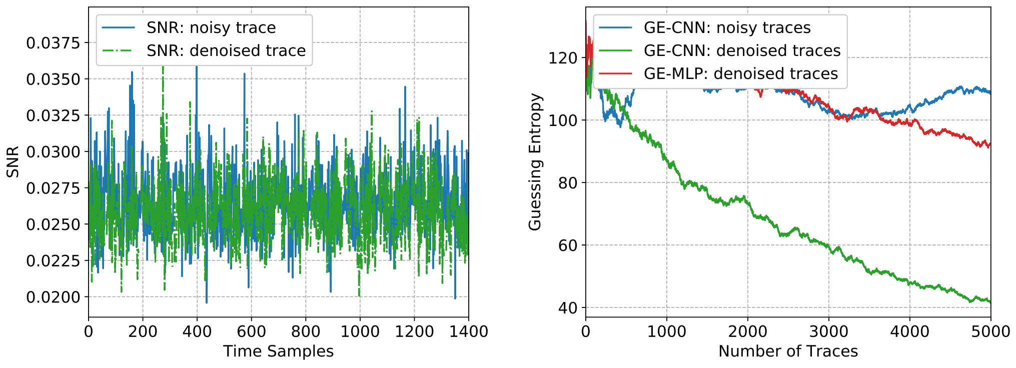
(b) Guessing entropy.

Figure 14: Noisy and denoised traces (all): SNR and guessing entropy.

From the GE plot (Figure 14b), the noisy traces do not converge with the increasing number of traces. The GE of denoised traces, on the other hand, reaches 34 with 5 000 traces when using CNN classifier. Interestingly, the MLP behavior after denoising is significantly better than CNN's performance without denoising. Still, GE equals 66 after applying 5 000 traces. Finally, we can observe that the attack performance converges slower than the denoised traces with a single type of noise, but CAE still proves its capability in removing the combined effect of noise and countermeasures.

4.6.2 AES with Random Keys

Next, 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) SNR. (b) Guessing entropy.

Figure 15: Noisy and denoised traces with variable key (all): SNR and guessing entropy.

Based on the results, there is no significant improvement from the SNR perspective. Still, guessing entropy indicates the improved performance as a result of CAE: for CNN, GE value converges to 42 with 5 000 traces. The GE of the noisy traces, on the other hand, fluctuates above 100 regardless of the number of traces. When considering MLP, we see that denoising helps but reduces GE to only 93 after 5 000 traces. Still, we can conclude that the proposed CAE can denoise the leakage in both fixed key and variable key scenarios where the results are especially impressive if using CNN as the attack mechanism.

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, such as Gaussian noise, desynchronization, random delay interrupts, and clock jitters. Additionally, we simulate the scenario where all noise types and countermeasures are combined into the measurements. There, the noisy and denoised traces are compared from two different perspectives: SNR and guessing entropy. To strengthen our experimental results, we consider two types of leakage traces (one encrypted with fixed and another with random keys), two attack strategies (CNN and MLP), and various levels of noise/countermeasure strengths. 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.

Denoising autoencoder provides an attacker with a powerful tool to pre-process the traces. Besides that, we expect it 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 difficult to transfer to another one. With the help of an autoencoder, this problem can be solved in another way: we can consider the traces variation as noise and use denoising autoencoder to remove it. The basic idea is to select the traces from one device as a reference and then train an autoencoder to convert the traces from other devices to the selected device. Since the variation of the traces between devices should be diminished with the help of the autoencoder, the attack model from one device could be applied to other devices as it should generalize better. Besides that, in this paper, we considered some common hiding countermeasures. 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.

[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.

[CFSC17] Keunwoo Choi, György Fazekas, Mark Sandler, and Kyunghyun Cho. Convolutional recurrent neural networks for music classification. In *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 2392–2396. IEEE, 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.

[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.

[LB$^+$95] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

[LeN19] LeNail. NN-SVG: Publication-Ready Neural Network Architecture Schematics. *Journal of Open Source Software*, 4(33):747, 2019.

[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.

[ODZ$^+$16] Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[PC$^+$15] Dimitri Palaz, Ronan Collobert, et al. Analysis of cnn-based speech recognition system using raw speech as input. Technical report, Idiap, 2015.

[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.

[SIJW18] Payap Sirinam, Mohsen Imani, Marc Juarez, and Matthew Wright. Deep fingerprinting: Undermining website fingerprinting defenses with deep learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 1928–1943. ACM, 2018.

[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.

[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.

[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.

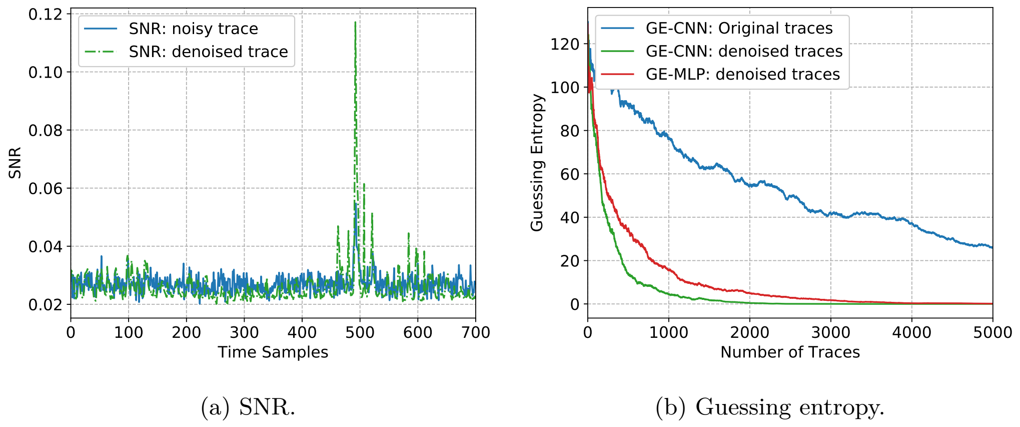
[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.

[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 with Smaller Levels of Noise

To further explore the behavior of the purposed CAE, we explore different levels of noise/countermeasures. For the Gaussian noise, the noise level is reduced from ± 20 to ± 10.

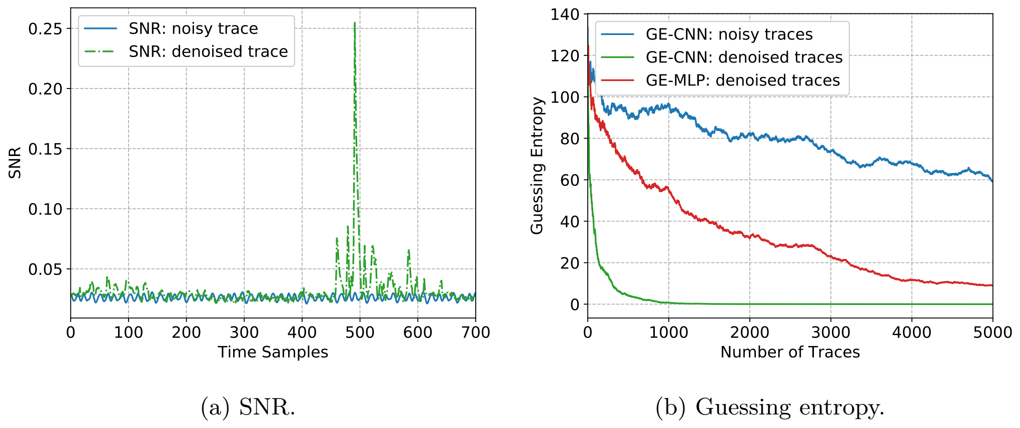
The SNR and GE results are shown in Figure 16. From the results, the GEs of denoised traces with both MLP and CNN shows better performance than their noisy counterpart. As expected, since now the noise level is reduced, the GE performance is improved over the case with a noise equal to 20.



(a) SNR. (b) Guessing entropy.

Figure 16: Noisy and denoised traces (Gaussian noise): SNR and guessing entropy.

For the desynchronization, the noise level is decreased from 50 to 30. The SNR and GE results are shown in Figure 17. In general, in both cases, fewer traces are needed to retrieve the key. The GE of the clean traces attacked with CNN is unchanged, indicating that in both noise levels (high:50/low:30), the maximum denoising capability of CAE is reached (also shown in Appendix B).

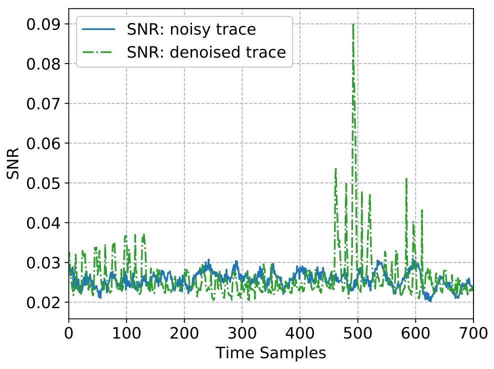


(a) SNR. (b) Guessing entropy.

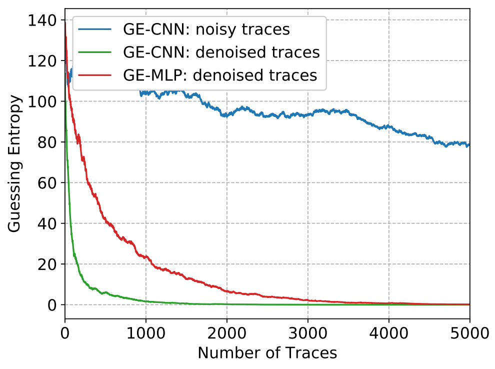
Figure 17: Noisy and denoised traces (desynchronization): SNR and guessing entropy.

For the random delay interrupts, the noise level is reduced from a=5/b=3 to a=4/b=2. The SNR and GE results are shown in Figure 18. As expected, the GE converges faster for all three attack cases.

Finally, for the clock jitters, the noise level was decreased from 4 to 2. The SNR and GE results are shown in Figure 19. Observe that CNN performs the best and that with MLP, we need to invert the key guess from the best to the worst one.

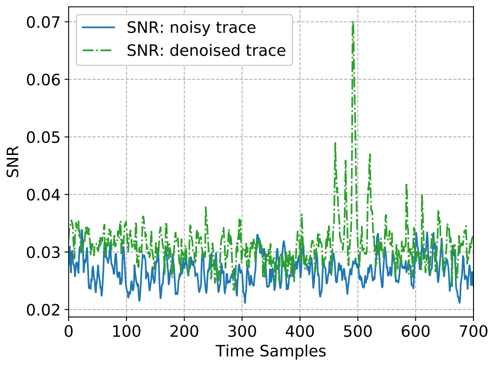


(a) SNR.

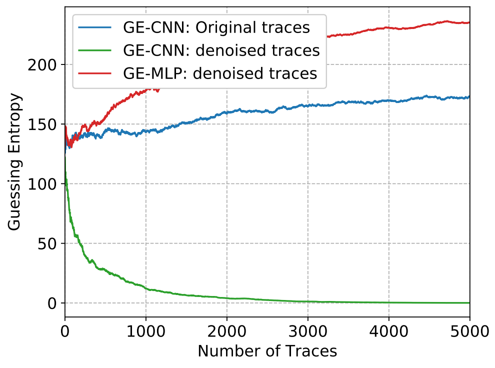


(b) Guessing entropy.

Figure 18: Noisy and denoised traces (RDIs): SNR and guessing entropy.



(a) SNR.



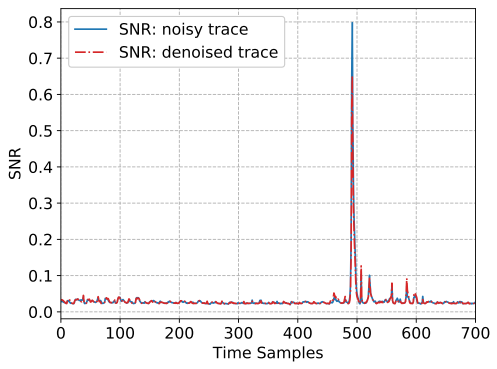
(b) Guessing entropy.

Figure 19: Noisy and denoised traces (clock jitters): SNR and guessing entropy.

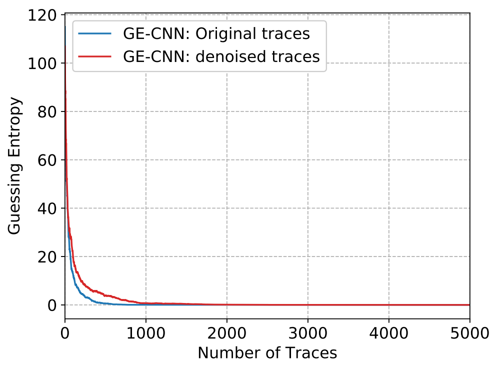
B Cleaning the Clean Traces

One should notice that the traces regenerated by CAE have information loss because of the bottleneck in the middle of the architecture. In an ideal case, CAE keeps the most critical information while neglecting the less important features. To evaluate the reconstruction capability, we now use CAE to denoise the clean traces. In this case, the input and output of the CAE are identical. While this does not represent a realistic case (as there is no reason to apply denoising autoencoder to traces that do not have noise), we conduct this experiment to 1) show that CAE removes mostly noise information, and 2) validate that even if the evaluator applies by mistake CAE, the performance of the attack will not be significantly reduced.

Figure 20 depicts results for the scenario when denoising autoencoder would be applied to already clean traces (ASCAD dataset with the fixed key). As can be expected, SNR reduces for those "cleaned" traces as CAE removes some of the information from the signal. For GE, there is no significant difference if we use clean traces or clean traces after CAE. In fact, only if the number of traces in the attack set is very limited, there will be a slight difference in the performance. More precisely, when compared with original traces, there is a 0.15 drop in terms of SNR; the number of traces to obtain the real key increase from 405 to 891.



(a) SNR.



(b) Guessing entropy.

Figure 20: Original clean traces and denoised clean traces: SNR and guessing entropy.

C Pseudocode for the Noise Simulation Techniques

In Algorithm 1 to 4, we give pseudocode for constructing traces with Gaussian noise, desynchronization, random delay interrupts, and clock jitters, respectively.

Algorithm 1 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 $new\_trace$
```

Algorithm 2 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 $new\_trace$
```

Algorithm 3 Random Delay Interrupts.

```
1: function ADD_RDIS($traces, A, B, rdi\_probability, rdi\_threshold, rdi\_amplitude$)
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 $rdi\_probability > rdi\_threshold$ 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 $new\_trace$
```

Algorithm 4 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 $new\_trace$
```