Diffuse Some Noise: Diffusion Models for Measurement Noise Removal in Side-channel Analysis

Sengim Karayalcin[1], Stjepan Picek[2] and Guilherme Perin[1]

[1] Leiden University, Leiden, The Netherlands,
s.karayalcin,@liacs.leidenuniv.nl,guilhermeperin7@gmail.com
[2] Radboud University, Nijmegen, The Netherlands, stjepan.picek@ru.nl

Abstract. Resilience against side-channel attacks is an important consideration for cryptographic implementations deployed in devices with physical access to the device. However, noise in side-channel measurements has a significant impact on the complexity of these attacks, especially when an implementation is protected with masking. Therefore, it is important to assess the ability of an attacker to deal with noise. While some previous works have considered approaches to remove (some) noise from measurements, these approaches generally require considerable expertise to be effectively employed or necessitate the ability of the attacker to capture a 'clean' set of traces without the noise. In this paper, we introduce a method for utilizing diffusion models to remove measurement noise from side-channel traces in a fully non-profiled setting. Denoising traces using our method considerably lowers the complexity of mounting attacks in both profiled and non-profiled settings. For instance, for a collision attack against the ASCADv2 dataset, we reduced the number of traces required to retrieve the key by 40%, and we showed similar improvements for ESHARD using a state-of-the-art MORE attack. Furthermore, we provide analyses into the scenarios where our method is useful and generate insights into how the diffusion networks denoise traces.

Keywords: Side-Channel Analysis · Deep Learning · Diffusion Models

1 Introduction

While standard cryptographic algorithms are generally considered (or, at least, believed after sufficient public analysis and scrutiny) theoretically secure, as retrieving the secret key from only inputs and outputs in a reasonable time is impossible, their real-world deployment poses additional attack surfaces. Deployments of these algorithms will unintentionally leak some information about their computation to the outside world through power consumption, timing, or electromagnetic emanation. These information leakages, or side channels, can allow an attacker to recover secret information from a device efficiently. Since being introduced by Kocher [Koc96], significant research has been done into side-channel attacks (SCA) and their countermeasures. We can broadly categorize SCA into two categories: 1) non-profiled attacks, where an attacker collects side-channel leakages and uses statistical distinguishers to extract the secret key [KJJ99, BCO04] and 2) profiled attacks, where the attacker builds a model for the leakage using a copy (clone) device they have full control over [CRR02]. From the machine learning perspective, we can divide the algorithms into generative and discriminative ones.[1]

[1] A common division in machine learning-based SCA is into supervised and unsupervised machine learning, but that relates to the task and whether there are labels available and not how the algorithm

While countermeasures for side-channel attacks exist, over recent years, a significant rise in deep learning-based SCA (DLSCA) has shown that these countermeasures can, in some cases, be circumvented, see, e.g., [LZC+21, PWP22, PCP20]. In the profiled setting, straightforward applications of discriminative models allow progressively more efficient attacks [MPP16, KPH+19, ZBHV19, WPP22b]. Similar approaches based on discriminative models have also been applied in the non-profiled setting [Tim19, DLH+22]. To a lesser degree, there are approaches based on generative models that allow for pre-processing of side-channel traces to simplify/improve attacks [WP20, WPP22a, ZBC+23].

While generative models can be a "natural" setting for SCA (for instance, the template attack [CRR02] is generative), we see fewer developments with generative models-based SCA in the last years compared to the discriminative ones. A part of the reason for this is that discriminative models excel at distinguishing among classes, which is a common setup for SCA (since we commonly consider the classification task). On the other hand, generative models generate new data, which is a natural option for data augmentation, a direction already explored in SCA.

In this work, we propose a novel approach to denoise traces based on Denoising Diffusion Probabilistic Models (DDPM). Using these models, we can effectively remove environmental (Gaussian) noise from side-channel traces without requiring a reference set of 'clean' traces. We experimentally validate our approach against several datasets and show improved attack performance for non-profiled collision attacks, non-profiled attacks using deep learning, higher order correlation power analysis (HO-CPA), and horizontal attacks. Additionally, when we consider profiling attacks, our technique can be used to improve the profiling complexity and ease the difficulty of finding good model architectures using hyperparameter search. Our main contributions are:

- We showcase the first use cases for DDPM models to pre-process traces in SCA.
- We provide an analysis of the trained DDPM models that explains how traces are denoised and gives insights into situations where denoising is possible.
- We showcase improvements in attack performance for state-of-the-art non-profiled attacks after denoising using the proposed model. For collision attacks against ASCADv2, the required number of traces to retrieve the key is reduced by approximately 40%, and for MORE attacks against ESHARD, we show similar gains in performance.
- We show significantly decreased difficulty in finding model architectures and hyperparameter configurations for profiling attacks, especially in settings with low numbers of profiling traces.

The source code to reproduce the experiments is publicly available.[2]

2 Background

2.1 Side-channel Analysis

Side-channel attacks [Koc96, KJJ99] are a class of attacks aiming at the implementation of cryptographic algorithms. The idea is that (physical) side-effects, e.g., timing [Koc96], power [Koc96], or the electromagnetic emanation [AARR02] of the execution of the algorithm can leak information about secret internal values. An attacker then captures a (large) number of traces of the algorithm's execution by measuring one of these side channels and utilizes these to mount the attack.

We can broadly categorize side-channel attacks into two threat models. First, non-profiled attacks where an attacker utilizes statistical distinguishers to differentiate the correct (sub)key candidate from the wrong ones. Techniques here generally compute the

works.

[2] https://anonymous.4open.science/r/diff_release-57F1

hypothetical intermediate value for all possible (sub)key candidates and attempt to find a connection between these labels and the side-channel traces [KJJ99, BCO04].

The second category includes profiled attacks. In this case, an attacker has access to (and full control of) an open copy of the device to be attacked. This allows the attacker to characterize the (physical) leakage using traces captured from the copy device, significantly improving the efficiency of attacks against the target [CRR02, SLP05].

Both of these categories of attacks rely on the fact that values that are operated on during the algorithm's execution are related to the measured traces. This relation is modeled by using a leakage model. A leakage model $f : Y \mapsto R$ mapping from an intermediate value $y \in Y$ to the leakage is generally composed of a part that relates to the hypothetical leakage of the value and a noise part. Common ways to model the leakage of this value are the Hamming Weight (HW) (the number of ones in the binary representation of y) or the Hamming Distance (HD) (the Hamming weight of the bitwise difference between y and the value it overwrites in a register).

To leverage this leakage model for key retrieval, the intermediate value an attacker targets needs to be related to the key and some known values. For AES implementations, the Sbox output in the first round is commonly used (for the Hamming weight and Identity leakage models). In this case, $y = \text{Sbox}(p_i \oplus k_i)$ where p_i and k_i are the i-th byte of the known plaintext and secret key. As these values are bytes, it is computationally feasible to calculate the hypothetical values for all 256 possible values of k_i and "match" those to the measured leakage. In this way, each key byte can be attacked separately, eventually leading to the recovery of the full key.

2.1.1 Signal-to-Noise Ratio (SNR)

SNR is a leakage assessment metric that quantifies the amount of leakage that is present in a random value. For a set of traces X with intermediate values Y at sample i, it is defined as:

$$\text{SNR}(X^i, Y) = \frac{\text{Var}_{y \in \mathcal{Y}}(E(X_i|y)}{E_{y \in \mathcal{Y}}(\text{Var}((X_i|y))}.$$

Here, E is the mean, Var is the variance of a random variable, and $\mathcal{Y}$ is the set of possible values in Y. We generally compute SNR for secret shares that leak directly (e.g., masks or masked sensitive values). In this work, we always compute SNR with 20 000 traces and the Identity leakage model.

2.2 Algorithmic Noise vs. Measurement Noise

We consider algorithmic noise to be the parts of the computation that are happening in parallel with the intermediate values we target. For example, an optimized hardware implementation of AES might execute several Sboxes in parallel, resulting in the Hamming weight of all output bits leaking together. If we want to target only one byte, the contribution to the leakage of the other bytes is considered noise. Measurement noise is then the part of the trace that is part of taking the physical measurements. This could be due to imperfections in the measurement setup or environmental factors. We generally assume this noise follows or is similar to, a Gaussian distribution [MP18].
The main difference between these types of noise for the purposes of unsupervised pre-processing of side-channel traces is that the algorithmic noise is part of the signal and is, therefore, not removed. An illustrative example is that if we take several measurements during the computation of a larger state, the algorithmic noise will stay the same for each of these samples, while the measurement noise will vary.

2.3 Datasets

ESHARD. The ESHARD dataset[3] contains EM measurements of an AES implementation protected with first-order Boolean masking. The dataset contains 1 400 sample points corresponding to the loading of the mask values and 100 000 traces with a fixed key. We target the Sbox output in the first round for all attacks. Note that we use the non-shuffled variant for all our analyses, as the shuffling was implemented by manipulating plaintexts a posteriori.

ASCADf. The ASCAD fixed key dataset (ASCADf)[4] contains EM measurements from an AES implementation protected with first-order Boolean masking. The dataset contains 60 000 traces with 100 000 samples each. We focus on a pre-selected window of 700 samples containing leakages for the masked Sbox computation in the first round for the 3rd key byte (which is the first masked byte). The dataset has a fixed key for all traces.

ASCADv2. The ASCADv2 dataset[5] contains power measurements of an AES implementation protected with an affine masking scheme and shuffling. The dataset contains 800 000 traces with 1 million sample points each. We take smaller part of the 15 000 sample extracted dataset used in [MS23], which contain 2 000 samples corresponding to a concatenation of indices 0-1 000 (loading masks), 6 040-6 540 (processing masked Sbox for third byte), and 11 250-11 750 (removing additive mask). Note that for this analysis, we disable shuffling by manipulating plaintexts a posteriori.

AES_HD. The AES_HD dataset[6] is an unprotected AES implementation on an FPGA board. The dataset contains 500 000 power traces using a fixed key. Each trace consists of 1 250 sample points. We target the Hamming Distance of register writing in the last round $(\text{Sbox}^{-1}[C_i \oplus k*] \oplus C_j)$.

AES_HD_MM. The AES_HD_MM dataset[7] is an AES implementation on an FPGA board protected with first-order Boolean masking. The dataset contains 5 600 000 traces using a fixed key. The measurements contain 3 125 samples per trace. We target the same intermediate value as for AES_HD.

ASCON. The ASCON dataset[8] is an unprotected software implementation of the ASCON cipher in authenticated encryption mode [DEMS21]. The dataset consists of 200 000 traces where 100 000 traces use random keys for profiling and 100 000 traces use a fixed key. Each trace consists of 772 sample points corresponding to the first round of the initialization phase of the authenticated encryption protocol.

2.4 Discriminative vs. Generative Models

Machine learning algorithms can be divided into two categories: generative and discriminative. Discriminative algorithms are primarily concerned with simulating the conditional probability distribution of the output labels given the input features. The goal is to understand the decision boundary. On the other hand, generative algorithms are designed to simulate the joint probability distribution of the input features (possibly conditioned on labels). To create new samples, their goal is to learn the underlying data distribution.

2.5 Denoising Diffusion Probabilistic Models (DDPMs)

Denoising Diffusion Probabilistic Models were first introduced by Ho et al. [HJA20]. Over the next few years, models based on the DDPM paradigm have outperformed state-of-the-art generative models on image generation and other tasks [YZS+24]. DDPM training is

[3] https://gitlab.com/eshard/nucleo_sw_aes_masked_shuffled
[4] https://github.com/ANSSI-FR/ASCAD/tree/master/ATMEGA_AES_v1/ATM_AES_v1_fixed_key
[5] https://github.com/ANSSI-FR/ASCAD/tree/master/STM32_AES_v2
[6] https://github.com/AISyLab/AES_HD_Ext
[7] https://chest.coe.neu.edu/?current_page=POWER_TRACE_LINK&software=ptmasked
[8] https://zenodo.org/records/10229484

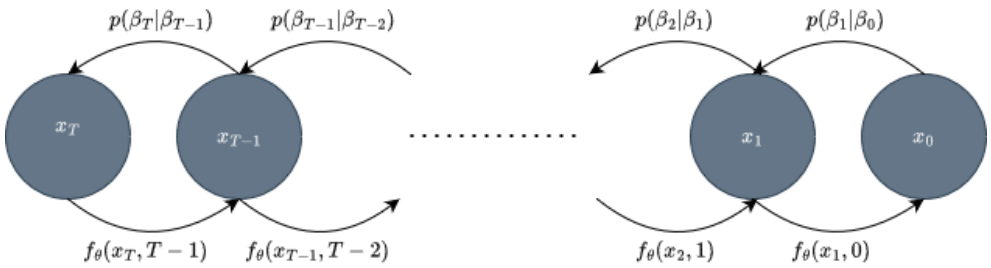


Figure 1: Diagram depicting the forward and backward process for training DDPMs.

based on a relatively straightforward paradigm: during training, we iteratively add some noise to an image (or some other type of data) for T steps; this is referred to as the forward process (left direction in Figure 1). Then, for an image x_t where noise has been added t times, we train the model to predict x_{t-1} and thereby remove noise. This is called the backward process (right direction in Figure 1). The central idea here is that when we start from fully random noise and iteratively remove noise, we can generate realistic-looking images as the diffusion models try to amplify patterns in the noise.

More formally, the forward process is defined using a Markov chain from x_0 (the original images) to x_T (Gaussian noise) and transitions $q(x_t|x_{t-1})$. We then have a noise schedule $\beta_0, \beta_1, \ldots, \beta_T$ and corresponding values $\alpha_0, \alpha_1 \ldots, \alpha_T$ (with $\alpha_0 = 0$ increasing to $\alpha_T = 1$). These α_i allow us to generate pairs x_t, x_{t-1} for arbitrary $1 < t \leq T$ using $x_t = (1 - \alpha_t)x_0 + \alpha_t Z$ and $x_{t-1} = (1 - \alpha_{t-1})x_0 + \alpha_{t-1} Z$ where $Z = \mathcal{N}(0, 1)$. These pairs can then be used to minimize the squared error of our diffusion model parameterized with weights θ, i.e., $\arg\min_\theta (f_\theta(x_t, t-1) - x_{t-1})^2$ using uniformly sampled t from $[0, T]$ for each mini-batch. For a more detailed description of diffusion models, see [HJA20].

3 Related Work

The profiling side-channel analysis started with the template attack [CRR02]. A few years later, the stochastic attack was also introduced [SLP05]. Interestingly, both of those attacks build generative models. With the introduction of "classical" machine learning in SCA, the community moved the attention to discriminative models. Still, deep learning-based generative models have been used in the last few years, with the primary goals to either pre-process the side-channel traces or generate synthetic traces.

3.1 Pre-processing using Neural Networks

While classical techniques for pre-processing side-channel traces have been explored, such techniques often require a significant domain expertise and error-prone manual intervention to achieve optimal results [LCSL07, OP12, PS15, MP18]. As such, the focus of the SCA community has recently moved to automated techniques utilizing deep learning. A first approach to using denoising autoencoders for removing noise from side-channel traces was proposed by Yang et al. [YLMZ19]. There, the authors used trace averaging to imitate a 'clean' set of traces, which can then be used to train an autoencoder to remove noise from the original traces. Subsequently, Wu and Picek [WP20] extended the approach to cover more hiding countermeasures like desynchronization and random delays. Berg et al. [vdBSB+23] further investigated hyperparameter configurations for these networks. Finally, Hu et al. [HSV24] included additional training objectives to improve the performance of autoencoders for removing noise from traces. Beyond autoencoders,

Wu et al. [WPP22a] utilized triplet networks to extract representations from traces that can be used to mount template attacks.

More recently, several studies have explored generative approaches for pre-processing. Genevey-Metat et al. [GHG21] utilized a GAN to translate traces between side-channel domains. Cao et al. [CZG+22] used a GAN approach to tackle portability challenges by transforming measurements from the attack device to the profiling device. Karayalcin et al. [KKW+23] investigated a Conditional Generative Adversarial Networks (CGANs)-based framework to emulate feature selection for masked implementations without access to mask values. Finally, Zaid et al. [ZBC+23] used variational autoencoders to model the physical leakage and subsequently leverage these models for attacks.
The main limitation of these approaches for pre-processing is that they only work in settings with additional assumptions over the standard non-profiled setting. The approaches using autoencoders in [YLMZ19, WP20, vdBSB+23] require a set of 'clean' traces that serve as a target for the networks. When considering Gaussian noise, this clean set can be emulated by averaging, but for masked implementations, this requires access to mask values [YLMZ19]. For the methods in [WPP22a, CZG+22, KKW+23, ZBC+23, HSV24], a labeled profiling set (or access to masks values for [ZBC+23]) is necessary for training the models. The trace translation in [GHG21] requires paired measurements in different side-channel domains, and it necessitates that the target side-channel domain is easier to attack, essentially mimicking the 'clean' set of traces in [WP20]. None of these approaches can effectively pre-process traces in a fully non-profiled setting.

3.2 Other Approaches using Generative Models in SCA

Several works have looked at applications of generative models for SCA. To generate additional traces, Wang et al. [WCL+20] considered CGANs to expand the size of the profiling set. Subsequently, Mukhtar et al. [MBPK22] improved upon the network architecture used in [WCL+20]. Finally, Yap and Jap [YJ24] proposed the use of diffusion models to generate additional traces. In all of these works, the authors relied on having access to a profiling device to label traces for training the networks. The resulting networks are then utilized to generate traces while providing label information to the network to control the trace generation. Note that while Transformers are often used in generative contexts in natural language processing, the Transformer architectures in [HSAM22, BIK+23, KVPB23, HCM24] are used as classification models (i.e., in a discriminative setting).

4 Denoising Diffusion Probabilistic Models for SCA

While utilizing DDPMs for data augmentation to improve side-channel attacks is a straightforward direction, the results from Yap and Jap [YJ24] suggest that there do not seem to be any significant advantages to using DDPMs over previously used (C)GAN methods [WCL+20, MBPK22]. Additionally, using DDPMs for trace generation requires profiling labels (or even mask knowledge) to allow for useful trace generation, which limits the applicability to profiled settings.

On the other hand, we utilize DDPMs to denoise traces. The key idea here is to take a diffusion model $f_\theta : X^m \times T \mapsto X^m$, where X^m is a side-channel trace with m samples and $T = \mathbf{Z}_n$ that we train using standard diffusion model training (see Section 2.5) on our measured traces. We then input actual traces (or x_0) and try to remove noise (or predict x_{-1}) from these traces. The reasoning here is that we can view side-channel traces as a combination of signal S and environmental/measurement noise Z. If we write $x_i = q_i * x_i + (1 - q_i) * Z$, where q_i is some schedule for the forward process (note that $q_i < q_{i-1}$), the optimization objective becomes $min_\theta(f_\theta(x_i, i) - x_{i-1})^2$. As the trained model has been trained to remove Gaussian noise, the model should then be able to

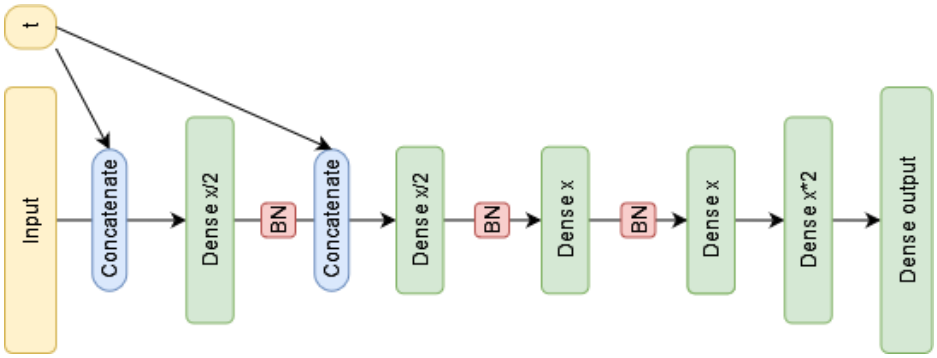


Figure 2: General model architecture for the input of size X.

remove some of the noise Z present in the original trace. Notably, this assumes that the distribution of the environmental/measurement noise Z is Gaussian, but this is a common assumption in the SCA domain [CRR02, MP18, WP20].

The main advantage of this approach is that the models are trained to be agnostic of implementation specifics. The only requirement is that the noise we are trying to remove follows a Gaussian distribution. As the diffusion model tries to reconstruct the trace, its output will also still follow the original trace's structure in terms of intermediate values. While this also holds for the denoising autoencoders in [YLMZ19, WP20], the training procedure of the autoencoders necessitates access to a reference set of 'clean' traces, which requires the ability to disable countermeasures on a profiling device. Note that while in [WP20], the authors also emulate this reference set by averaging traces, this still requires a large number of additional measurements, complicating the process and making it only possible for unmasked implementations. Indeed, in masked implementations, an attacker cannot know in which traces the same intermediate values are processed (i.e., which traces to average) without mask knowledge [YLMZ19]. If an attacker tries to average traces with the same label but different mask values, then the attacker will average traces that leak different intermediate values.

4.1 Network Architecture

To keep the focus of this work on the viability of DDPMs for denoising traces in an unsupervised context, we only use synchronized traces. This allows us to restrict our architecture to shallow MLPs as these have been shown to be effective for processing synchronized side-channel traces [PWP22]. As such, the design choices for our architecture are relatively simple: we follow the general structure of a U-Net [RFB15] where we first downsample for several layers, then keep the same dimensions for some layers to induce a compressed latent representation, and finally upsample again to the original trace dimensions. We utilize batch-normalization layers in the downsampling section of our network to stabilize training. The network architecture can be seen in Figure 2.

4.2 Hyperparameter Setup

To train the DDPM models across all experiments, we use the Adam optimizer [KB15]. The learning rate is scheduled according to an exponential decay[9] schedule with an initial learning rate of 0.001, the decay rate of 0.96, and 10 000 decay steps. We train all models for 200 epochs using batch size 200. We use the *tanh* activation function for every intermediate

[9] https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/schedules/ExponentialDecay

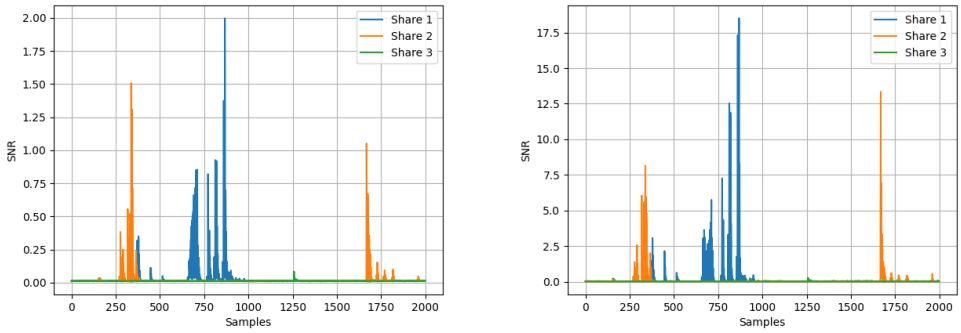
layer and the linear activation for the output. We use $T = 16$ for all of the experiments. These hyperparameters perform well and allow for reasonably effective denoising against the considered targets. We arrived at these values after some preliminary testing. Note that these are not optimal, but we refrain from further optimization as this setup can already show the merits of our approach. We provide further experiments to show the effects of some hyperparameter variations in Appendix A.

For the denoising of the traces after training, we observe in the initial set of experiments that predicting traces with $t = 15$ ($f_\theta(x_0, 15)$) works significantly better than using $t = 0$ ($f_\theta(x_0, 0)$). The intuition behind this is that for higher t, the model gets noisier inputs during training, which forces it to find patterns in its input data more aggressively. As such, we use $t = 15$ for all experiments unless otherwise specified.

4.3 Proof of Concept

To provide a proof of concept for our method, we first look at testing against (relatively) noisy trace sets from software targets. First, we consider the ASCADv2 target as this is currently the most difficult public software target. There, the leakage of the masked output is noisy (SNR around 0.08), which potentially allows for significant benefits. Second, the ESHARD target provides measurements of a software implementation where both the mask and masked Sbox output leak with relatively low SNRs (see Figure 4a). Note that in this section, we only consider software targets where the measurements contain at least a moderate amount of measurement noise.

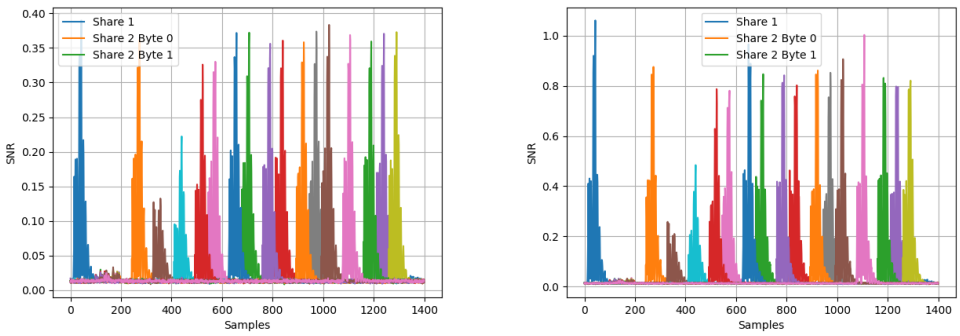
We train the models using all available traces and subsequently use the trained model to obtain denoised traces. We use the SNR of the secret shares in the traces (using the Identity leakage model) as a measure of how successful the noise removal was. In Figure 3, we see the SNR results for a trimmed version of the ASCADv2 dataset. The results clearly show that the SNR peaks for all three shares are significantly improved. In fact, we see that for share 2, the improvement is almost 10×. In Figure 4, we again see significant improvements in SNR. These results show that our networks are effective at amplifying the side-channel signal. However, there are significant differences in the magnitude of the improvements. The SNR of the first share of ASCADv2 is improved by a factor of 9, while the improvement for the third share has a factor of 2.5 only. While these differences are not problematic, they raise questions about how the models create these improvements. In Section 4.4, we aim to explain these differences using simulated traces.



(a) Original (b) Diffused

Figure 3: SNR values for secret shares for ASCADv2.

In Figure 5, we showcase histograms of the values in the highest SNR point for the secret shares. We can observe that, especially for share 1 of ASCADv2 and for ESHARD,



(a) Original (b) Diffused

Figure 4: SNR values for secret shares for Eshard.

the distributions are much smoother, and the separation between classes is clearer for the diffused traces. Additionally, the separation between classes is increased. These results indicate that the diffusion networks can effectively smooth out the noise from side-channel traces while maintaining the leakages.

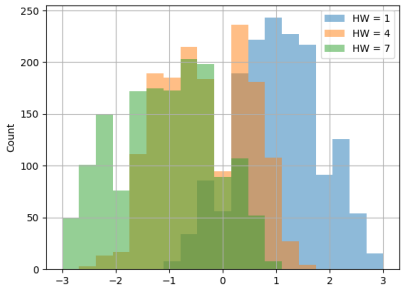
4.4 Simulations

Next, we explore in what situations we can improve the SNR of side-channel measurements using our DDPMs. To accomplish this, we utilize simulations with varying noise levels and a varying number of informative points. We follow the procedure:

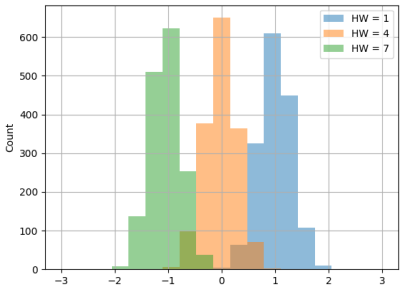
1. we generate traces of 100 points noise following a normal distribution, and then,
2. for $0 < n \leq 40$ of these points (to allow different settings), we include the Hamming weight of an 8-bit intermediate value y uniformly sampled from the range $[0, 255]$.

The main purpose of varying the number of leaky points is to determine how the DDPMs are amplifying the side-channel signal. While the results in Section 4.3 show clear improvements in terms of SNR for individual features, the networks cannot provide more information than what is present in the original trace. As such, the increased SNR in individual features must come from other trace points. Intuitively, combining information from several points leaking the same value is straightforward to amplify the signal in each of these points. As can be seen in Figure 6, there is a clear link between the number of leaky features and the level of SNR achieved. Notably, for both the high and low noise scenarios, the model does not increase the SNR if only one leaky feature is present. In the low-noise scenario, the model already shows significant improvement over baseline SNR when two leaky features are included, improving further with more leaky features. In the high-noise scenario, more leaky features are required before the SNR levels are improved over the baseline.

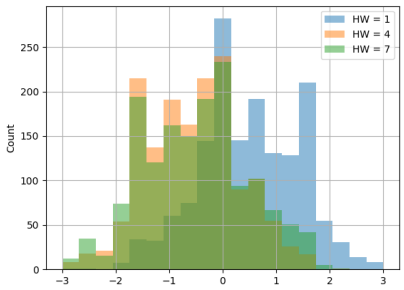
Overall, these results strongly suggest that diffusion models learn to differentiate the side-channel signal from noise by looking for correlated features in the trace. By finding and combining information from those related points, the model can decrease the error in its output. This is relevant for real-world side-channel traces when we take several measurements during an operation that leaks some sensitive value, e.g., our oscilloscope has a high sampling rate or if some sensitive value is manipulated in several trace points.



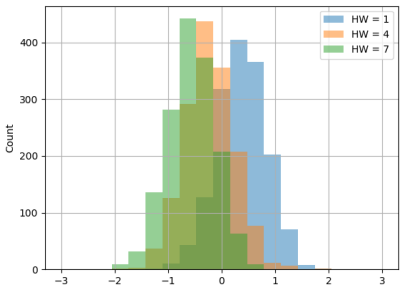
(a) Original ASCADv2 share 1



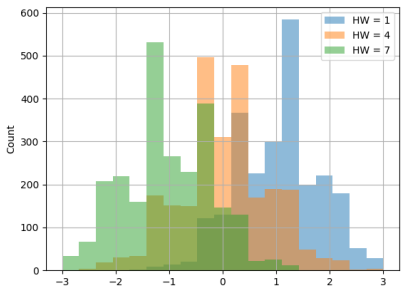
(b) Diffused ASCADv2 share 1



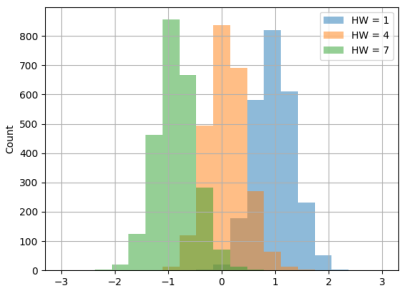
(c) Original ASCADv2 share 3



(d) Diffused ASCADv2 share 3



(e) Original ESHARD

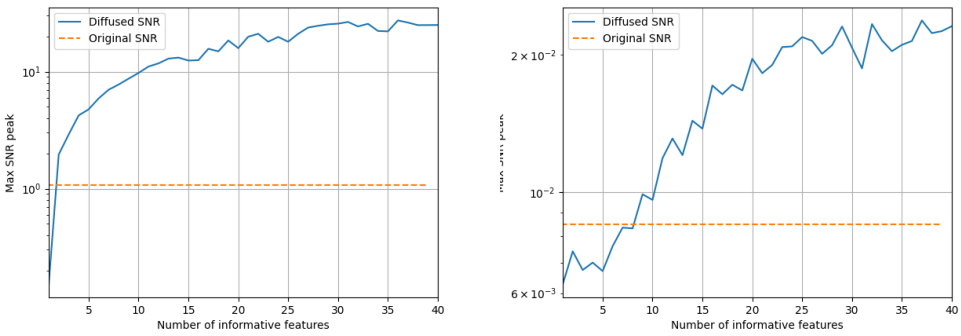


(f) Diffused ESHARD

Figure 5: Histogram for the HW values in the highest SNR samples.

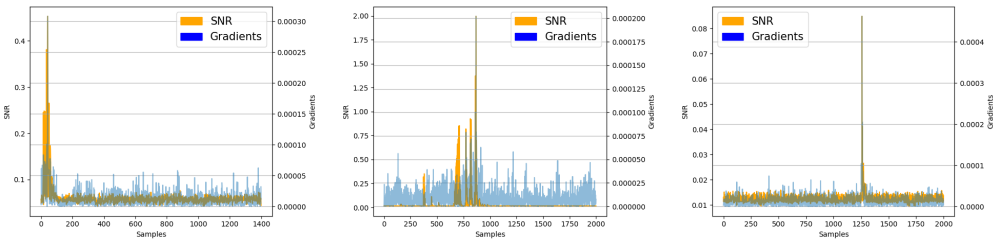
4.5 Gradient Visualization

To show that the DDPMs learn to combine information across correlated features in real-world settings, we visualize what features contribute to one of the output features. We use gradient visualization techniques that have been previously used in SCA [MDP20]. In Figure 7, we provide the gradient visualizations of the highest SNR features. We clearly see that sample points correlated with the most informative sample influence the model's output. Notably, these results explain the significant differences in the magnitude of the SNR increases for different secret shares of ASCADv2 we saw in Figure 3. As can be seen in Figure 7c, the diffusion model can only utilize a small number of samples to combine information for share 3, while in the case of share 1, there are significantly more samples to learn from. These differences follow the demonstrated trends in Figure 6.



(a) Low-noise scenario (b) High-noise scenario

Figure 6: Maximum SNR value for diffused traces simulations for varying numbers of informative points.



(a) ESHARD (b) ASCADv2 share 1 (c) ASCADv2 share 3

Figure 7: Gradients vs. SNR values.

5 Experimental Results

In this section, we present results for state-of-the-art attacks to explore the practical usefulness of our DDPMs.

5.1 Non-profiled Attacks

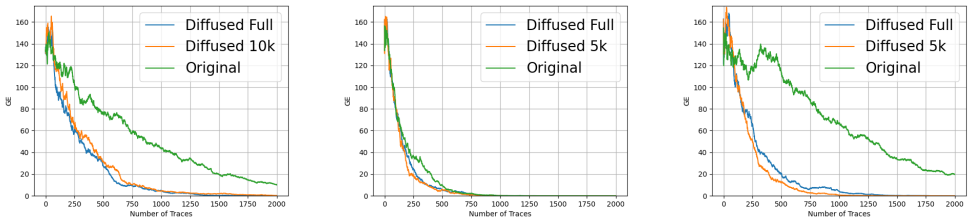
5.1.1 Correlation Attacks

First, we investigate improvements to second-order CPA-based attacks. As these attacks utilize one sample for each secret share, the expectation is that we will see significant improvements as the diffusion models allow for the implicit utilization of information leaked across several correlated features. We showcase scenarios for ASCADf and ESHARD to show the effects of a diffusion model on the attack performance when SNR is improved (ESHARD) and when it is not (ASCADf). For ASCADf, we also present results where Gaussian noise[10] was added to the traces before training the diffusion model, as the original traces are relatively low noise. Note that as peak SNR is not improved for other targets, diffusing traces has no impact on the performance of CPA-based attacks [Man04]. We select the highest SNR samples for each share in the original traces for these attacks and combine them using absolute difference as a shortcut to avoid testing all possible feature combinations.

The results in Figure 8 indicate that clear improvements in attack performance are

[10]After standardization, with mean 0 and standard deviation 1. We use the same setup in Section 6.

achieved. Note that for these attacks, the CPA results are not entirely representative of real-world attacks. In fact, we require more traces than are available in the attacks to train the diffusion models. Thus, to simulate representative attacks, we should train diffusion models for every subset of traces we attack in each of the attack simulations, which is impractical, especially when training diffusion models using low trace counts. However, for noisier targets where (very) large numbers of traces are necessary for key retrieval, this limitation is not an objection, as the diffusion model can be trained using the larger set. As such, the results in Figure 8 indicate that trained diffusion models provide significant benefits for improving CPA attacks (or other attacks that represent the leakage of a secret share using a single sample point).



(a) ESHARD (b) ASCADf (c) ASCADf with noise

Figure 8: CPA results.

5.1.2 Multi Output Regression Enhanced (MORE)

This section provides results for state-of-the-art non-profiled attacks using DL [SKP+24]. The basic idea of this attack is to train one model labeled for every possible key and conduct the regression task. As the labels generated using the correct key are the only ones that are related to the trace, the model should then most accurately predict labels of the correct key. A ranking for key candidates can then be created by measuring the network error for each candidate. We only show results against ESHARD as breaking the ASCADv2 target is still infeasible using the MORE methodology, while for ASCADf and the hardware targets, diffusing traces does not make a difference in terms of attack performance. We generate a distribution of key ranks using 40 separate random models following the hyperparameter ranges used in [SKP+24]. We choose this method as it allows us to assess the impact of diffusing traces on the difficulty of defining an appropriate model configuration, and it reflects directly on the effectiveness of the ensemble-based attacks that use these random models. The diffusion model in this case is the same as in Section 5.1.1 using 10 000 traces. We use the HW leakage model and target the third key byte. The results in Figure 9 showcase that diffusing the traces helps significantly. Attacks using diffused traces perform similarly at 20 000 traces to the attacks using 50 000 original traces. This indicates that diffusion models significantly help the consistency of training discriminative models, especially in more restricted settings.

5.1.3 Collision Attack against ASCADv2

To demonstrate the practical relevance of our approach, we first showcase attacking results in a non-profiled context. We focus on the collision attacks as described by Wu et al. [WPP24], which aim to recover the bitwise difference between sub-keys (key-deltas). These key-deltas can then be used to brute-force one key byte, leading to full key recovery (given correct key-deltas). We include this attack as it is the only attack that can break

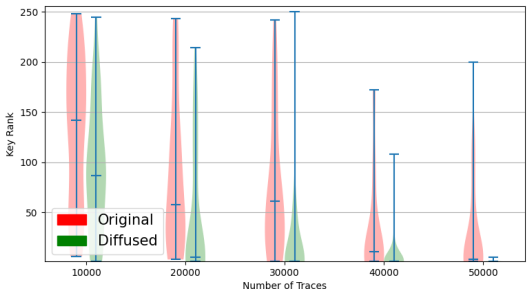
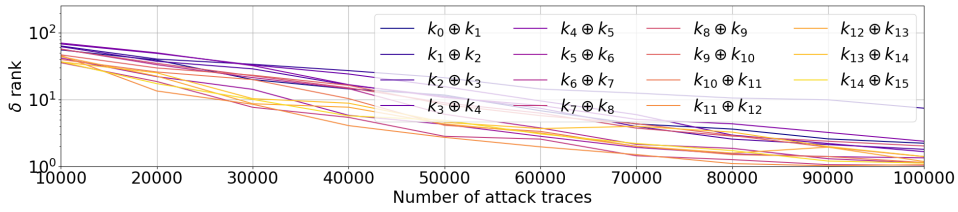
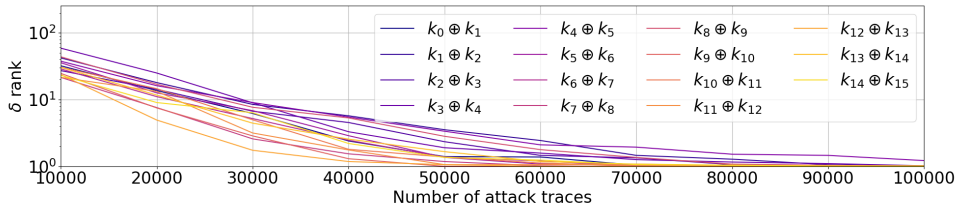


Figure 9: MORE results for ESHARD.



(a) Original



(b) Diffused

Figure 10: ASCADv2 collision attacks.

the ASCADv2 dataset in a non-profiled setting[11]. Note that Cristiani et al. [CLHM22] also showcased successful attacks against the same implementation, but they require a different acquisition campaign with significantly more traces.

For training the DDPM, we use the intervals given in [WPP24]. Note that for these attacks, the shuffling countermeasure is disabled, and we simulate a fixed attack key for the profiling set (see [WPP24] for details). To simplify the analysis, we concatenated the used 100 sample intervals into one 1 600 sample trace and trained the diffusion model on 20 000 such traces to limit computational overhead. We then executed 50 runs on randomly sampled traces from the 500 000 profiling traces and averaged key-delta ranks to achieve a GE estimate. The results in Figure 10 clearly favor the diffused traces. In fact, using diffused traces can successfully reduce GE for all key-delta candidates below 1 using 60 000 traces, while three of the deltas are not fully recovered using 100 000 traces for the original traces.

[11]With shuffling disabled.

Table 1: Average/Max single trace accuracy for cswap_arith using the one neuron perceptron form [BA23] and CNN setups from [PCBP21].

	One neuron	CNN	CNN +Dropout	Random CNN	Random CNN + Dropout
Original	70.9%/79.2%	63.6%/73.7%	55.2%/75.7%	71.7%/80.0%	98.6%/99.6
Diffused	96.3%/99.2%	70.8%/87.1%	50.1%/83.5%	62.6%/81.1%	99.6%/100%

Table 2: Hyperparameter search ranges for MLP architecture as a profiling attack model.

Hyperparameter	Options
Dense layers	1, 2, 3, 4
Neurons	10, 20, 50, 100, 200, 300, 400, 500
Activation Function	selu, relu,
Learning Rate	0.005, 0.001, 0.0005, 0.0001
Optimizer	Adam, RMSprop
Batch Size	100, 200, 300, 400, 500, 600, 700, 800, 900, 1000
Weight Initialization	random uniform, he uniform, glorot uniform

5.1.4 Horizontal Attacks against Public Key Implementation

To illustrate that our method is generally useful for analyzing side-channel traces, we showcase improvements to the horizontal attack from [PCBP21]. In this attack, initial labeling that is only slightly better than random guessing (around 52%) is iteratively improved upon using CNNs. In this work, results are presented using both an optimized fixed CNN architecture and a new random CNN at each iteration (for more details, see [PCBP21]). Note that in subsequent work, it was shown that similar attack performance could be achieved in some cases using only one neuron instead of larger CNNs [BA23]. We only show results on the cswap_arith dataset as the cswap_pointer is significantly easier to attack, and almost every network setup achieves 100% accuracy on both the original and diffused traces. The diffusion model is trained in the standard way using the 63 750 traces. For this dataset, we use batch size 1 000 and obtain denoised traces with $t = 0$ instead of $t = 15$ as this achieved better results.

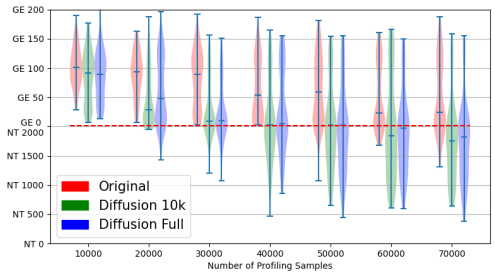
The results in Table 1 again show significant improvements to accuracies by using diffused traces. In all cases, the maximum accuracy using diffused traces is higher than using original traces. Especially in the case where only the simplest perceptron from [BA23] is used, we obtain 99.2% maximum accuracy using diffused traces while only 80% using the original traces. For the CNNs, we see that the fixed CNN improves between 10% and 15%, and using random CNNs, we find the only attack achieving 100% maximum accuracy uses diffused traces.

5.2 Profiling Attacks

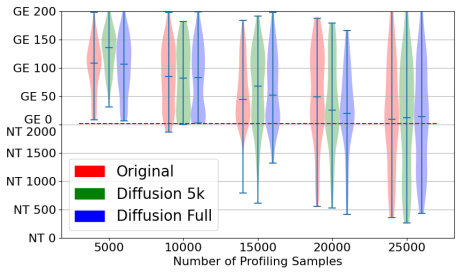
In this section, we explore the impact of using diffusion models to denoise traces in a profiled setting. We report the distribution of the attack performance of random models to assess the impact of using diffused traces on the difficulty of finding good model configurations.

5.2.1 Experimental Setup

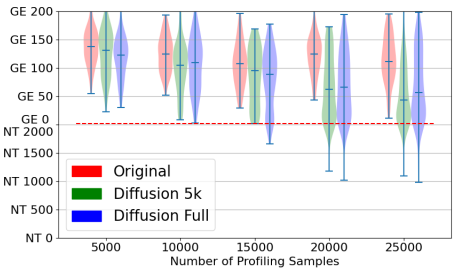
To investigate the impact of using denoised traces for profiling attacks, we will examine the profiling complexity of attacks against several datasets. To do this, we randomly search small MLP models using the ranges in Table 2. This search is run using a varying number of profiling traces for original and diffused traces. We use two diffusion models, one trained with the maximum considered number of profiling traces and one with the minimum considered number of traces (5 000 and 25 000, and 10 000 and 70 000 for ASCADf and ESHARD, respectively).



(a) ESHARD HW leakage model



(b) ASCADf ID leakage model



(c) ASCADf with added noise ID leakage model

Figure 11: Distribution of GE/number of traces to reach $GE = 1$ for 100 random MLPs in various scenarios.

5.2.2 Results

Figure 11a shows the distribution of attacking results for the 100 random MLPs against ESHARD. The attack performance is significantly improved by utilizing diffused traces. For all the tested settings, we see that more of the models trained on diffused traces result in successful attacks. In fact, the distribution of attack performances at 30 000 diffused profiling traces is already better than the distribution using 70 000 original profiling traces. Additionally, in settings with lower numbers of profiling traces, only attacks using diffused traces can successfully recover the key in 2 000 traces.

In Figure 11b, the results for ASCADf are less impressive. In fact, in this case, there does not seem to be any difference between using diffused and original traces. When we simulate a noisier measurement setup by adding Gaussian noise to the traces, we see that the benefits of using DDPMs are restored. In Figure 11c, none of the models using original traces can successfully recover the key byte in 2 000 attack traces, while for diffused traces, a number of the networks is successful starting at 15 000 profiling traces. These results indicate that for datasets where the diffusion models successfully improve SNR, it becomes significantly easier to define and train profiling models that can retrieve the key, while for cases where the models do not improve SNR, the difficulty remains the same.

6 Datasets with Algorithmic Noise

In previous sections, we have shown significantly improved attack performance against targets that have (almost) no algorithmic noise. In these cases, it is clear from the results in Section 4.3 that the measurement noise is mitigated by using diffusion models. However, when we consider targets that process larger states, the denoising is less relevant.

In Figure 12, we see that the DDPMs cannot improve the peak SNR for any of the

Table 3: Max SNR peaks for denoising autoencoder approach.

	Original	DDPM	DAE CNN from [WP20]	Our MLP trained as AE
ASCADf	1.30	1.24	0.76	1.17
Eshard	0.39	1.06	0.20	0.03
ASCADv2 share 1	2.00	8.90	1.35	1.93
ASCADv2 share3	0.08	0.19	0.06	0.03

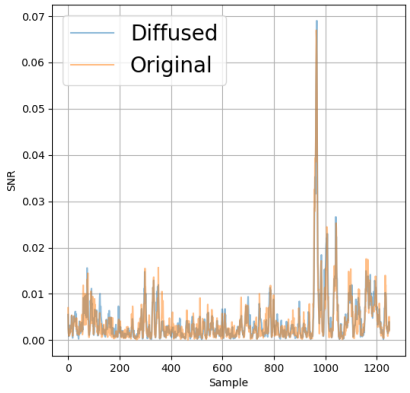
targets without added noise. Only for AES_HD_MM in Figure 12c, we see that the SNR of the other samples is increased by a marginal amount. Note that this does not seem to affect attack performance; for profiling attacks using 100 random MLP models using the ranges from Table 2, GE is 60.78 ± 23.95 and 59.85 ± 21.00 for original and diffused traces, respectively. Looking at traces with added Gaussian noise, we see improved SNR for ASCON and AES_HD_MM. For AES_HD, SNR does not improve, presumably, as the noise level is relatively high and the intermediate value is only leaked in a very small number of samples following the results on simulations in Section 4.4.

These results indicate that the diffusion models are not as useful for measurements with mostly algorithmic noise. However, even for hardware targets, we can see some improvements to SNR in specific samples, indicating that removing (some) measurement noise is still achievable in these cases. Additionally, when we add Gaussian noise, we see clear improvements in SNR for both the ASCON and the AES_HD_MM targets, reinforcing the usefulness of our diffusion models in scenarios with noisier measurements. Since noisier settings are more relevant from a practical perspective, and even the current results with deep learning perform well in scenarios with little noise, we consider our approach highly relevant and applicable in real-world settings.

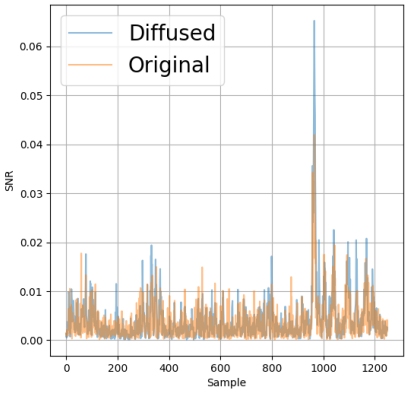
7 Comparison with Denoising Autoencoders

While several works have looked at utilizing techniques from the deep learning domain for pre-processing side-channel traces, almost none of these techniques are directly applicable in a non-profiled setting. As mentioned before, most either require profiling labels [WPP22a, KKW+23, HSV24] or the ability to capture some 'clean' target traces [YLMZ19, WP20, GHG21]. However, as mentioned in Section 4.1 of [WP20], denoising autoencoders can be used for the same purpose. In this case, the idea is that when the autoencoder is forced to compress the relevant information in the trace in a smaller representation and then reconstruct the original input. In principle, the model is forced to discard irrelevant information (noise) and maintain the side-channel signal. We compare the performance of these models in terms of the maximum SNR peaks as a simple representation of the denoising performance.

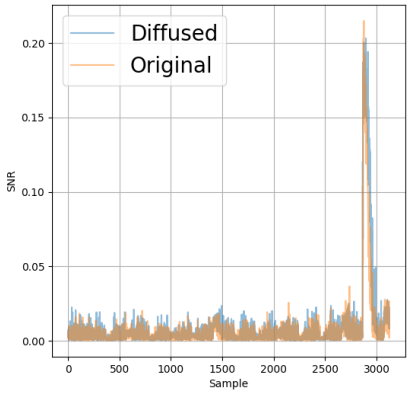
From the results in Table 3, we can clearly see that autoencoders trained without a target set of clean traces are not effective at removing measurement noise. Both the convolutional architecture from [WP20] and our DDPM architecture trained as autoencoders fail to increase SNR over the original traces. While tuning the architectures for each specific dataset could lead to improvements, we note that this is not necessary for our DDPMs. In addition, the autoencoders trained to reconstruct traces can easily overfit and memorize their training traces, increasing the difficulty of finding an appropriate architecture, especially in non-profiled contexts. Overall, it seems clear that in a scenario without a set of clean target traces (or label/mask knowledge), autoencoders are not appropriate for removing noise.



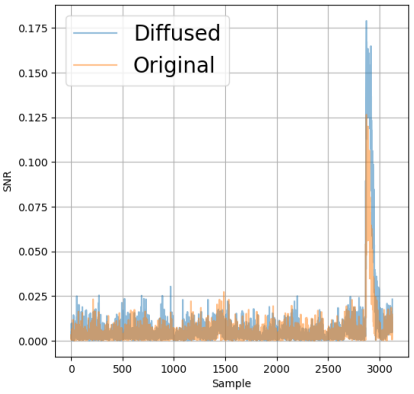
(a) Original AES_HD



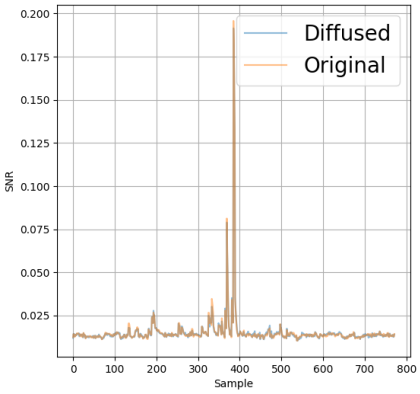
(b) Noise added AES_HD



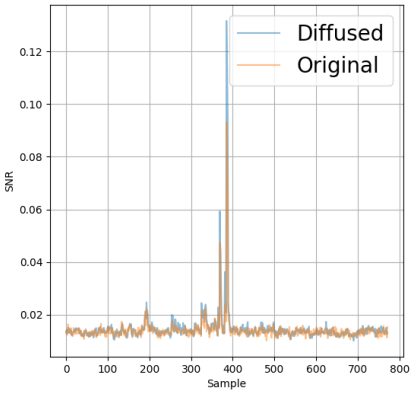
(c) Original AES_HD_MM



(d) Noise added AES_HD_MM



(e) Original ASCON



(f) Noise added ASCON

Figure 12: SNR values for intermediate values for various datasets.

8 Discussion

Our results showcase that DDPM models can learn useful representations of side-channel traces in unsupervised contexts. Additionally, the analysis in Section 4.4 shows how the networks learn these representations and the intuition for why the denoising can work. The mechanism is quite straightforward. To remove noise from a leaky sample point, the network needs more information about the leaking value. To accomplish this, it can find features that leak the same value and combine the information from these features to arrive at a less noisy version of the feature. In effect, we compress the information from several leaky samples into a singular sample.

The benefits of this for non-profiled attacks are clear. For these attacks, we often utilize only a single feature to represent each secret share [CLHM22]. In these contexts, our models allow for the implicit utilization of several leaky samples without having access to mask values. Additionally, for collision attacks, the stronger separation between classes can clearly reduce the number of traces necessary to detect key differences.

For profiled attacks, the benefits are less obvious. In principle, using LDA to reduce the dimension from a sufficiently large number of informative samples effectively compresses the information from all of those samples and thus eliminates the benefits of diffusing traces. Similarly, a well-trained neural network should implicitly combine the information available across a trace. However, in practice, we see significant benefits to denoising traces before training profiling models. Our results in Section 5.2(and in Section 5.1.2) clearly indicate that the difficulty of finding appropriate hyperparameters for neural networks and the required number of profiling traces is significantly reduced. We believe this happens because diffusion models are significantly more powerful than LDA or commonly used neural networks in SCA. While our results show significant gains for the showcased attacks against some targets (specifically ESHARD and ASCADv2), it is clear that these benefits are not universal. Our method does not improve the SNR for datasets that contain mostly algorithmic noise. However, the SNR also does not seem to be harmed by pre-processing the traces using diffusion models, limiting the downside of using our method to the (limited) computational overhead required for training the diffusion model. The artificial addition of Gaussian noise in Section 6 also showcases that the method is still effective in more difficult scenarios when the measurements are more noisy.

The main takeaway from these results is that including diffusion models for pre-processing traces within the evaluation can significantly reduce the overhead caused by environmental noise while not requiring the same level of expertise to tune as other methods from the DL domain. The tuning of diffusion models in the SCA context seems relatively straightforward, and the pre-processing can be used to simplify any subsequent analysis. Especially in contexts where individual samples are used to represent the leakage from sensitive values, like (higher-order) CPA, our method allows for the combination of information from several samples without any additional access assumptions or alterations to the attack methodology. While the practical results presented in Section 5 showcase strong benefits in terms of attack performance, these are clearly influenced by our choice of datasets. Overall, the attack improvements are only present when the SNR of secret shares is similarly improved. For targets aside from ESHARD, ASCADv2, and cswap_arithmetic, utilizing our DDPMs does not seem to make a significant difference for practical attacks in our experiments unless we artificially inject Gaussian noise to simulate noisier measurement setups. Note that there may be some benefits (minor SNR improvements we see for AES_HD_MM and ASCON), but this did not make a difference for the considered attacks. Another consideration for interpreting our results is that the network and hyperparameter settings leave significant room for improvement. We aimed to present the denoising method using DDPMs, not to optimize the denoising performance for each specific scenario. In fact, even the results in Appendix A show that larger improvements than those presented in Section 5 are relatively straightforward to

achieve by changing only one or two hyperparameters. Consequently, this suggests that future research looking into more complex network architectures and hyperparameter optimizations would still be beneficial in evaluating the full potential of our approach.

9 Conclusions and Future Works

We have presented an approach for utilizing DDPMs to remove noise from side-channel traces. As shown in Section 4.4, our approach is effective at increasing SNR levels of traces when several samples in the side-channel trace leak the same information. In these cases, DDPMs can combine information from several samples to remove noise from each of the individual samples. Notably, this is, to the best of our knowledge, the first approach that can effectively denoise traces in a fully non-profiled setting without a "clean" set of target traces. Furthermore, we showed significant improvements in attack performance for several state-of-the-art non-profiled attacks and similar improvements in the profiling complexity of deep learning models for profiled attacks. One of the main limitations of our work is that we focus on aligned traces. While directly training our models on misaligned traces does not pose any technical difficulties, achieving satisfactory performance in such a context is more difficult. In future work, we plan to investigate mechanisms for effectively applying our method to misaligned traces, and to investigate more complex network architectures. Besides this, there are a number of use cases within the SCA domain for our models, which could potentially be interesting. Some initial ideas include pre-training (parts of) classification models since diffusion models could aid in the training of profiling models and exploring whether the implicit compression of leakage from several features can be used to limit the computational overhead of subsequent attacks.

A Hyperparameter Evaluation

As we only utilize one architecture and hyperparameter configuration to achieve the results, we provide insights into the effects of varying this architecture/hyperparameter setup. First, we look at variations in the depth of our architecture and the type of activation function. Second, we look at training time parameters like the number of steps T in our diffusion process, the initial learning rate, and batch size. Finally, we consider the number of epochs and the number of traces required to train the models.

All of these results are evaluated on the maximum SNR peaks of the first secret share of ESHARD (the mask) and the third share of ASCADv2 (the masked Sbox output). We chose these two secret shares as our standard architecture works well on these targets, and they are rather different in terms of the number of features that leak the secret shares (which can be seen in Section 4.3). Note that in every table, we make the value we use in our standard configuration bold to improve readability.

A.1 Architecture

The effects of varying the number of downsampling (and corresponding upsampling) layers are quite different for different targets. When only one downsampling layer is used, Table 4 shows that the difference from the original SNR is limited for both targets. When the number is increased, we see that for ESHARD, the SNR increases over our standard configuration, while for ASCADv2, the SNR decreases. In Table 5, varying the activation function results in somewhat decreased performance for ASCADv2, while for ESHARD, there does not seem to be much of an effect. Overall, defining an appropriate architecture is mainly the question of defining an appropriate depth. Using two blocks seems like a reasonable middle-ground for the network, not ignoring certain leakages that

only contribute to a small number of features while still providing enough expressive power in the network to remove noise effectively.

Table 4: SNR peaks for varying numbers of downsampling blocks in the network.

	1	**2**	3	4
ASCADv2	0.08	0.18	0.08	0.02
ESHARD	0.63	1.13	1.61	1.48

A.2 Training Time Hyperparameters

As can be seen in Table 6, the initial learning rate is quite an important factor. Only the standard 0.001 can effectively denoise ASCADv2. For ESHARD, it matters significantly less, and while the performance is best for our standard case, varying it still results in significant improvements over the original traces. The effect of varying batch size in Table 8 is fairly limited, and while 200 seems like a good default value, increasing it to speed up training is seemingly not that harmful to the performance. Finally, varying the number of steps T in Table 7 shows significant room for improvement over our baseline model. Especially for ASCADv2, we achieve another 50% improvement in peak SNR by optimizing T. Overall, this indicates that the specifying batch size and T are not too sensitive, and values in a broad range are effective. On the contrary, defining learning rates that are inappropriate can quickly result in models that do not learn anything for some targets.

A.3 Epochs and Number of Traces

Table 10 shows that the models are surprisingly effective when given only a relatively small number of training traces. In fact, while using more traces obviously does not hurt, the benefits are only marginal, and for the fairly difficult case of ASCADv2's third share, we can already see a doubling of the SNR using only 10 000 traces. The number of epochs is somewhat more sensitive. In Table 9, for ESHARD, the model performance is already good with only 10 epochs, while for ASCADv2, good performance starts at 100 epochs.

References

[AARR02] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The EM side-channel(s). In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 29–45. Springer, 2002.

[BA23] Sana Boussam and Ninon Calleja Albillos. Keep it unsupervised: Horizontal attacks meet simple classifiers. In Shivam Bhasin and Thomas Roche, editors, *Smart Card Research and Advanced Applications - 22nd International Conference, CARDIS 2023, Amsterdam, The Netherlands, November 14-16, 2023, Revised Selected Papers*, volume 14530 of *Lecture Notes in Computer Science*, pages 213–234. Springer, 2023.

Table 5: SNR peaks for varying activation functions.

	tanh	relu	selu	linear
ASCADv2	0.18	0.15	0.09	0.08
ESHARD	1.13	1.43	1.21	1.24

Table 6: SNR peaks for varying initial learning rates.

	0.01	**0.001**	0.0001	0.0005	1e-05	5e-05
ASCADv2	0.02	0.18	0.05	0.08	0.02	0.04
ESHARD	0.02	1.13	0.98	0.99	0.98	0.97

Table 7: SNR peaks for varying number of steps T.

	4	8	**16**	32	128	512	1024
ASCADv2	0.10	0.18	0.18	0.20	0.27	0.09	0.02
ESHARD	1.27	1.28	1.13	1.15	1.16	1.53	1.56

Table 8: SNR peaks for varying batch sizes.

	100	**200**	300	400	500	600	700	800	900	1 000
ASCADv2	0.15	0.18	0.16	0.12	0.13	0.14	0.08	0.07	0.06	0.07
ESHARD	0.89	1.13	1.05	1.01	1.00	0.99	0.98	1.05	0.97	0.99

Table 9: SNR peaks for varying numbers of epochs.

	10	25	50	100	**200**	400
ASCADv2	0.03	0.08	0.11	0.15	0.18	0.20
ESHARD	1.09	1.08	1.18	1.20	1.13	1.13

Table 10: SNR peaks for varying numbers of traces.

	10 000	20 000	30 000	**40 000**	50 000	60 000
ASCADv2	0.16	0.15	0.17	0.18	0.20	0.24
ESHARD	0.89	1.02	1.04	1.20	1.13	1.26

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, volume 3156 of *Lecture Notes in Computer Science*, pages 16–29. Springer, 2004.

[BIK+23] Elie Bursztein, Luca Invernizzi, Karel Král, Daniel Moghimi, Jean Michel Picod, and Marina Zhang. Generic attacks against cryptographic hardware through long-range deep learning. *CoRR*, abs/2306.07249, 2023.

[CLHM22] Valence Cristiani, Maxime Lecomte, Thomas Hiscock, and Philippe Maurine. Fit the joint moments: How to attack any masking scheme. *IEEE Access*, 10:127412–127427, 2022.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski Jr., Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, volume 2523 of *Lecture Notes in Computer Science*, pages 13–28. Springer, 2002.

[CZG+22] Pei Cao, Hongyi Zhang, Dawu Gu, Yan Lu, and Yidong Yuan. AL-PA: cross-device profiled side-channel attack using adversarial learning. In Rob Oshana, editor, *DAC '22: 59th ACM/IEEE Design Automation Conference, San Francisco, California, USA, July 10 - 14, 2022*, pages 691–696. ACM, 2022.

[DEMS21] Christoph Dobraunig, Maria Eichlseder, Florian Mendel, and Martin Schläffer. Ascon v1.2: Lightweight authenticated encryption and hashing. *J. Cryptol.*, 34(3):33, 2021.

[DLH+22] Ngoc-Tuan Do, Phu-Cuong Le, Van-Phuc Hoang, Van-Sang Doan, Hoai Giang Nguyen, and Cong-Kha Pham. Mo-dlsca: Deep learning based non-profiled side channel analysis using multi-output neural networks. In *2022 International Conference on Advanced Technologies for Communications (ATC)*, pages 245–250, 2022.

[GHG21] Christophe Genevey-Metat, Annelie Heuser, and Benoît Gérard. Trace-to-trace translation for SCA. In Vincent Grosso and Thomas Pöppelmann, editors, *Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers*, volume 13173 of *Lecture Notes in Computer Science*, pages 24–43. Springer, 2021.

[HCM24] Suvadeep Hajra, Siddhartha Chowdhury, and Debdeep Mukhopadhyay. Estranet: An efficient shift-invariant transformer network for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2024(1):336–374, 2024.

[HJA20] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[HSAM22] Suvadeep Hajra, Sayandeep Saha, Manaar Alam, and Debdeep Mukhopadhyay. Transnet: Shift invariant transformer network for side channel analysis. In Lejla Batina and Joan Daemen, editors, *Progress in Cryptology - AFRICACRYPT 2022: 13th International Conference on Cryptology in Africa, AFRICACRYPT 2022, Fes, Morocco, July 18-20, 2022, Proceedings*, volume 13503 of *Lecture Notes in Computer Science*, pages 371–396. Springer Nature Switzerland, 2022.

[HSV24] Fanliang Hu, Jian Shen, and Pandi Vijayakumar. Side-channel attacks based on multi-loss regularized denoising autoencoder. *IEEE Trans. Inf. Forensics Secur.*, 19:2051–2065, 2024.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael J. Wiener, editor, *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer, 1999.

[KKW+23] Sengim Karayalcin, Marina Krcek, Lichao Wu, Stjepan Picek, and Guilherme Perin. It's a kind of magic: A novel conditional gan framework for efficient profiling side-channel analysis. Cryptology ePrint Archive, Paper 2023/1108, 2023. https://eprint.iacr.org/2023/1108.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, volume 1109 of *Lecture Notes in Computer Science*, pages 104–113. Springer, 1996.

[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.

[KVPB23] Praveen Kulkarni, Vincent Verneuil, Stjepan Picek, and Lejla Batina. Order vs. chaos: A language model approach for side-channel attacks. *IACR Cryptol. ePrint Arch.*, page 1615, 2023.

[LCSL07] Thanh-Ha Le, Jessy Clédière, Christine Servière, and Jean-Louis Lacoume. Noise reduction in side channel attack using fourth-order cumulant. *IEEE Trans. Inf. Forensics Secur.*, 2(4):710–720, 2007.

[LZC+21] Xiangjun Lu, Chi Zhang, Pei Cao, Dawu Gu, and Haining Lu. Pay attention to the raw traces: A deep learning architecture for end-to-end profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021.

[Man04] Stefan Mangard. Hardware countermeasures against DPA ? A statistical analysis of their effectiveness. In Tatsuaki Okamoto, editor, *Topics in Cryptology - CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23-27, 2004, Proceedings*, volume 2964 of *Lecture Notes in Computer Science*, pages 222–235. Springer, 2004.

[MBPK22] Naila Mukhtar, Lejla Batina, Stjepan Picek, and Yinan Kong. Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. In Steven D. Galbraith, editor, *Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings*, volume 13161 of *Lecture Notes in Computer Science*, pages 297–321. Springer, 2022.

[MDP20] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):348–375, 2020.

[MP18] Houssem Maghrebi and Emmanuel Prouff. On the use of independent component analysis to denoise side-channel measurements. In Junfeng Fan and Benedikt Gierlichs, editors, *Constructive Side-Channel Analysis and Secure Design - 9th International Workshop, COSADE 2018, Singapore, April 23-24, 2018, Proceedings*, volume 10815 of *Lecture Notes in Computer Science*, pages 61–81. Springer, 2018.

[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.

[MS23] Loïc Masure and Rémi Strullu. Side-channel analysis against anssi's protected AES implementation on ARM: end-to-end attacks with multi-task learning. *J. Cryptogr. Eng.*, 13(2):129–147, 2023.

[OP12] David F. Oswald and Christof Paar. Improving side-channel analysis with optimal linear transforms. In Stefan Mangard, editor, *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, volume 7771 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2012.

[PCBP21] Guilherme Perin, Lukasz Chmielewski, Lejla Batina, and Stjepan Picek. Keep it unsupervised: Horizontal attacks meet deep learning. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(1):343–372, 2021.

[PCP20] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):337–364, Aug. 2020.

[PS15] Santos Merino Del Pozo and François-Xavier Standaert. Blind source separation from single measurements using singular spectrum analysis. In Tim Güneysu and Helena Handschuh, editors, *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, volume 9293 of *Lecture Notes in Computer Science*, pages 42–59. Springer, 2015.

[PWP22] Guilherme Perin, Lichao Wu, and Stjepan Picek. Exploring feature selection scenarios for deep learning-based side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2022(4):828–861, Aug. 2022.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In Nassir Navab, Joachim Hornegger, William M. Wells III, and Alejandro F. Frangi, editors, *Medical Image Computing and Computer-Assisted Intervention - MICCAI 2015 - 18th International Conference Munich, Germany, October 5 - 9, 2015, Proceedings, Part III*, volume 9351 of *Lecture Notes in Computer Science*, pages 234–241. Springer, 2015.

[SKP+24] Ioana Savu, Marina Krček, Guilherme Perin, Lichao Wu, and Stjepan Picek. The need for more: Unsupervised side-channel analysis with single network training and multi-output regression. In Romain Wacquez and Naofumi Homma, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 113–132, Cham, 2024. Springer Nature Switzerland.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, volume 3659 of *Lecture Notes in Computer Science*, pages 30–46. Springer, 2005.

[Tim19] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(2):107–131, 2019.

[vdBSB+23] Danny van den Berg, Tom Slooff, Marco Brohet, Kostas Papagiannopoulos, and Francesco Regazzoni. Data under siege: The quest for the optimal convolutional autoencoder in side-channel attacks. In *International Joint Conference on Neural Networks, IJCNN 2023, Gold Coast, Australia, June 18-23, 2023*, pages 1–9. IEEE, 2023.

[WCL+20] Ping Wang, Ping Chen, Zhimin Luo, Gaofeng Dong, Mengce Zheng, Nenghai Yu, and Honggang Hu. Enhancing the performance of practical profiling side-channel attacks using conditional generative adversarial networks. *CoRR*, abs/2007.05285, 2020.

[WP20] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(4):389–415, 2020.

[WPP22a] Lichao Wu, Guilherme Perin, and Stjepan Picek. The best of two worlds: Deep learning-assisted template attack. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2022(3):413–437, 2022.

[WPP22b] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. *IEEE Transactions on Emerging Topics in Computing*, pages 1–12, 2022.

[WPP24] Lichao Wu, Guilherme Perin, and Stjepan Picek. Not so difficult in the end: Breaking the lookup table-based affine masking scheme. In Claude Carlet, Kalikinkar Mandal, and Vincent Rijmen, editors, *Selected Areas in Cryptography – SAC 2023*, pages 82–96, Cham, 2024. Springer Nature Switzerland.

[YJ24] Trevor Yap and Dirmanto Jap. Creating from noise: Trace generations using diffusion model for side-channel attack. Cryptology ePrint Archive, Paper 2024/167, 2024. https://eprint.iacr.org/2024/167.

[YLMZ19] Guang Yang, Huizhong Li, Jingdian Ming, and Yongbin Zhou. CDAE: towards empowering denoising in side-channel analysis. In Jianying Zhou, Xiapu Luo, Qingni Shen, and Zhen Xu, editors, *Information and Communications Security - 21st International Conference, ICICS 2019, Beijing, China, December 15-17, 2019, Revised Selected Papers*, volume 11999 of *Lecture Notes in Computer Science*, pages 269–286. Springer, 2019.

[YZS+24] Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications. *ACM Comput. Surv.*, 56(4):105:1–105:39, 2024.

[ZBC+23] Gabriel Zaid, Lilian Bossuet, Mathieu Carbone, Amaury Habrard, and Alexandre Venelli. Conditional variational autoencoder based on stochastic attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(2):310–357, 2023.

[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.