Creating from Noise: Trace Generations Using Diffusion Model for Side-Channel Attack

Trevor Yap[1,2][0000−0001−8651−574X] and Dirmanto Jap[2][0000−0002−3149−9401]

[1] School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore
[2] Temasek Laboratories, Nanyang Technological University, Singapore
trevor.yap,djap@ntu.edu.sg

Abstract. In side-channel analysis (SCA), the success of an attack is largely dependent on the dataset sizes and the number of instances in each class. The generation of synthetic traces can help to improve attacks like profiling attacks. However, manually creating synthetic traces from actual traces is arduous. Therefore, automating this process of creating artificial traces is much needed. Recently, diffusion models have gained much recognition after beating another generative model known as Generative Adversarial Networks (GANs) in creating realistic images. We explore the usage of diffusion models in the domain of SCA. We proposed frameworks for a known mask setting and unknown mask setting in which the diffusion models could be applied. Under a known mask setting, we show that the traces generated under the proposed framework preserved the original leakage. Next, we demonstrated that the artificially created profiling data in the unknown mask setting can reduce the required attack traces for a profiling attack. This suggests that the artificially created profiling data from the trained diffusion model contains useful leakages to be exploited.

Keywords: Side-channel · Neural Network · Deep Learning · Profiling attack · Generative Models · Diffusion Model.

1 Introduction

Side-channel Attacks (SCA) are one of those crucial threats that are required to be evaluated. Information on the secret data could be leaked in physical properties such as power consumption [13], and electromagnetic emanation [1]. Many of such physical properties come in a form known as traces. SCA analyzes these traces to recover the secret data in various ways. Profiling attacks and non-profiling attacks are common forms of SCA. Very often, the success of these attacks relies heavily on the number of traces. However, in a practical setting, there might be a limitation on the number of traces that can be collected by the adversary, presumably due to some factors; for example, the device itself is protected, which only allows limited access to the device. Due to this limitation, the performance of the SCA could be affected, for example, making it hard to generalize the leakages. As such, there is a need for more data or traces to analyze. However, manually creating artificial traces can be quite complicated and tedious as it needs to capture the leakage information and its characteristics properly.

In recent years, there has been a rise in using machine learning techniques to tackle the automation of creating such artificial traces. One common approach is to use Generative Adversarial Networks (GAN) [7], a popular technique commonly used in the image processing domain for generating synthetic images. Recently, a few of the previous works [23] and [16] have investigated creating artificial traces using GAN. However, in the image domain community, another generative model known as the Denoising Diffusion Probabilistic Model (DDPM) has risen in popularity recently due to its performance in producing more realistic images exceeding that of GAN. As such, in this work, we aim to explore and investigate how to adopt DDPM into the SCA domain.

Our Contributions. In this work, our contributions are stated as follows:

1. We investigate the applicability of the DDPM approach for traces generation in the context of SCA. We proposed two different frameworks in which a diffusion model can be used: Known mask setting and unknown mask setting.
2. In the known mask setting, we highlight that the generated traces can exhibit the original leakages as observed in the original traces by evaluating the traces with Correlation Power Analysis (CPA).
3. On the other hand, we show the effectiveness of the diffusion model in generating artificial data in the unknown mask setting. By increasing the down-sampled profiling traces with the newly generated data from the diffusion model, we show that the number of attack traces needed for a profiling attack decreases.

In this work, we target synchronized and desynchronized traces. We validate our approach on traces up to the first-order masking for real traces. We leave higher-order masking of real traces to future works. The results can be publicly accessed on the following weblinks [3].

Paper Organization. The paper is organized as follows. In Section 2, we give an overview of related works over the recent years. Section 3 will provide the necessary background on side-channel analysis and DDPM. In Section 4, we present the datasets and the building blocks of the neural network being used. Section 5 provides a visualization of the leakages that an diffusion model could provide. Subsequently, we present the results of profiling attack when using artificial data created by the trained diffusion model for profiling in Section 6. Lastly, in Section 7, we conclude the paper and outline some future works.

2 Related Works

One of the common approaches adopted in the machine learning domain is the data augmentation approach. A form of data augmentation is the Synthetic

[3] https://github.com/yap231995/Diffusion-SCA

Minority Over-sampling Technique (SMOTE), which was explored in [19]. They applied data augmentation to deal with data imbalance due to the Hamming Weight (HW) leakage model. As such, after balancing the training data, it could improve the attack performance. Another work on different data augmentation was reported in [14], where the authors investigate data augmentation techniques against masked AES with hiding countermeasures. It reported that the data augmentation could help in decreasing the effectiveness of hiding countermeasures, albeit requiring specific configuration when dealing with different Deep Neural Network (DNN) architectures. Another work by Cagli *et al.* [4] proposed using data augmentation for Deep-Learning (DL)-based SCA. They proposed a data augmentation method by manually adding jitters into the original traces to increase the number of traces for profiling.

Recently, more works have performed more in-depth investigations on the applicability of data augmentation through the automatic generation of synthetic traces through the use of DNN. [23] introduced a new approach to generating new traces through the usage of Conditional Generative Adversarial Network (CGAN). They show that CGAN can generate new traces that learn the leakage from both unprotected and protected implementations. However, the leakage model considered in their work is the Hamming Weight (HW) leakage model, resulting in fewer classes. Furthermore, the correlation of the traces evaluated did not consider any comparison with other keys. In [16], the authors proposed another approach when generating traces automatically based on CGAN and Siamese networks. They used the proposed approach to generate datasets for both symmetric and public-key cryptographic implementations. Compared to previous work, they also investigate and analyze the effect of the GAN network on data generation. However, they only considered the dataset with fixed key profiling and attack traces called ASCADf and an ECC dataset. In [11], the authors proposed another CGAN-based approach. In their approach, the generator receives real traces as input and is not conditioned with label class, which allows it to extract the features from the unlabeled set. Therefore, their approach did not create new artificial data but as a form of feature extraction and dimensionality reduction.

In all recent works, the idea is to use data augmentation to generate artificial data, which can also capture the characteristics of the leakage as well as the countermeasures, such as hiding or masking leakage. Most of the works have been utilizing GAN as the main approach for data generation and work under the unknown mask setting. In this work, we investigate an alternative approach using diffusion model for data generation for both known mask and unknown mask settings and investigate how the approach could learn the leakage characteristics.

3 Background

In this section, we provide basic backgrounds on the topics that we will use throughout the whole paper.

3.1 Correlation Power Analysis (CPA)

One of the most commonly used attacks is the CPA [3]. The general approach is to use Pearson correlation to establish the relation between different intermediate values from different secret hypotheses and the actual leakage values. The attacker will use the intermediate values computed as function f of known inputs p and (hypothetical) secret $k \in K$. In this case, the attacker will compute $H = f(p, k)$. These intermediate values will then be compared with the actual leakage traces T obtained while processing actual secret k^*. The secret k with the highest (absolute) correlation can then be estimated as the secret value.

The Pearson correlation between x and y can be computed as follow:

$$r(x,y) = \frac{\sum_{i=1}^{N}((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^{N}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{N}(y_i - \bar{y})^2}}. \tag{1}$$

For the intermediate values, they have to be mapped to the leakage. In general, a leakage model is used to approximate the behavior of the measured traces. For the software implementation, the leakage is usually assumed to follow the HW model. In contrast, for hardware implementation, it is the Hamming distance (HD) model. In addition to the mentioned leakage values, an identity mapping (ID) can also be used as an alternative leakage model. In this work, we will mainly focus on the ID leakage model.

3.2 Profiling Attacks

Profiling attacks assume a worst-case scenario where the adversary has access to a clone device and a target device. These two devices are similar to each other. In this setting, the adversary can manipulate or know the device's key of the clone device while the key for the target device is unknown to him. Furthermore, the adversary has the ability to collect multiple traces from a known set of random plaintexts (or ciphertexts) from both devices. The adversary will obtain the profiling traces from the clone device while acquiring the attack traces from the target device. The goal of the adversary is to recover the unknown key from the target device.

Profiling attacks can be divided into the profiling phase and the attack phase. In the profiling phase, a distinguisher $\mathcal{F}$ is built from the set of profiling traces. This distinguisher will return a conditional probability mass function $\Pr(\boldsymbol{T}|Z = z)$. During the attack phase, the distinguisher returns a probability score for each hypothetical sensitive value. In other words, we obtain $\boldsymbol{y}_i = \mathcal{F}(\boldsymbol{t}_i)$ where $\boldsymbol{t}_i$ represents an attack trace. We compute the log-likelihood score for every key $k \in \mathcal{K}$ as follows:

$$s_{N_a}(k) = \sum_{i=1}^{N_a} \log(\boldsymbol{y}_i[z_{i,k}]),$$

where N_a as the number of attack traces used and $z_{i,k} = C(p_i, k)$ are the hypothetical sensitive values based on the key k with p_i being the corresponding public variable to the trace $\boldsymbol{t}_i$ and C is the cryptographic primitive. Next, we rank the key of the log-likelihood score in decreasing order and classify them into a guess vector $\boldsymbol{G} = [G_0, G_1, \ldots, G_{|\mathcal{K}|-1}]$ with the score G_0 corresponds to the score of the most likely key candidate, and the score $G_{|\mathcal{K}|-1}$ to be the score for is the least likely key candidate. The rank of the key shall be denoted as the index of guess vector $\boldsymbol{G}$. The guessing entropy GE is defined as the average rank of the correct key k^* over multiple experiments [22]. If $GE = 0$, when using N_a attack traces, the attack is considered successful. We denote $NTGE$ to be the least number of traces required to attain $GE = 0$.

The most known profiling attack is Template Attacks (TA). The distinguisher is built using the Bayes' Theorem with the assumption that the conditional probability $Pr(\boldsymbol{T}|Z = z)$ follows the multivariate Gaussian distribution [5]. Overall, it outputs the following as the posterior probability,

$$\Pr(Z = z_k|\boldsymbol{T} = \boldsymbol{t}) = -\frac{D}{2}\log(2\pi) - \frac{1}{2}\log(det(\Sigma_k)) - \frac{1}{2}(\boldsymbol{t} - \overline{\boldsymbol{t}_k})^T \Sigma_k (\boldsymbol{t} - \overline{\boldsymbol{t}_k}),$$

where $\overline{\boldsymbol{t}_k}$ is the sample mean of class z_k and Σ_k is the covariance matrix of class z_k with determinant $det(\Sigma_k)$.

3.3 Denoising Diffusion Probabilistic Models (DDPM)

DDPM was first created by [9] in 2020 to generate images and was extensively improved [17]. In fact, recently [6] shows that with enough tuning, DDPM could attain better performance compared to GAN. DDPM is a type of Markovian Hierarchical Variational Autoencoder (H-VAE), which can be viewed as stacking multiple Variational Autoencoder together (VAE). Figure 1 illustrates how DDPM works visually. Given a data distribution $x_0 \sim q(x_0)$, we define the for-

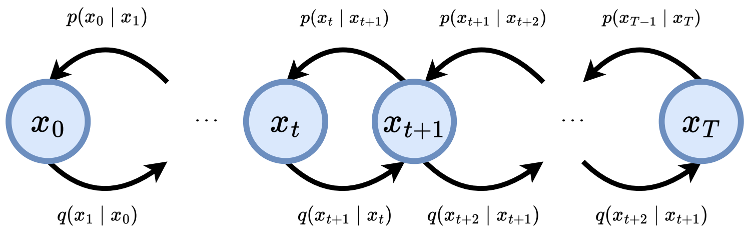


Fig. 1: Visualization representation of DDPM. x_0 represents the original data while x_T denote the pure Gaussian noise. The intermediate x_t portrays the noisy version of x_0 at time step t.

ward noising process q which iteratively adding Gaussian noise at each time t with a variance $\beta_t \in (0, 1)$ to x_0 to obtain x_1 to x_T as

$$q(x_1, \ldots, x_T|x_0) = \prod_{t=1}^{T} q(x_t|x_{t-1}),$$

$$q(x_t|x_{t-1}) = \mathcal{N}(x_t; \sqrt{1-\beta_t}x_{t-1}, \beta_t \boldsymbol{I}).$$

Suppose T is sufficiently large and the β_t follows a schedule, for example linear or cosine schedule, then the latent x_T is almost isotropic Gaussian distribution (i.e., $x_T \sim \mathcal{N}(0, \boldsymbol{I})$). This means that we can sample $x_T \sim \mathcal{N}(0, \boldsymbol{I})$ and reverse the process to obtain data from $q(x_0)$. Throughout this work, we set β_t to follow the cosine schedule as proposed in [17].

We estimate the reverse process by using a neural network by defining the reverse process as

$$p(x_{t-1}|x_t) = \mathcal{N}(x_{t-1}; \mu_\theta(x_t, t), \Sigma_\theta(x_t, t)).$$

As noted in [9], because noising process q is modeled from a Gaussian distribution, one can show that it is allowed to sample x_t for any t directly from the input data x_0,

$$x_t = \sqrt{\bar{\alpha}_t}x_0 + \sqrt{1-\bar{\alpha}_t}\epsilon$$

$$q(x_t|x_0) = \mathcal{N}(x_t; \sqrt{\bar{\alpha}_t}x_0, 1-\bar{\alpha}_t \boldsymbol{I})$$

where $\alpha_t = 1 - \beta_t$, $\alpha_t = \prod_{s=0}^{t} \alpha_s$ and $\epsilon \sim \mathcal{N}(0, \boldsymbol{I})$.

By Bayes' Theorem, we can reformulate $q(x_{t-1} \mid x_t, x_0)$ in terms of $\bar{\beta}_t$ and $\mu_q(x_t, x_0)$:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(x_t; \mu_q(x_t, x_0), \bar{\beta}_t \boldsymbol{I})$$

where

$$\bar{\beta}_t = \frac{1-\bar{\alpha}_{t-1}}{1-\bar{\alpha}_t}\beta_t, \text{ and } \mu_q(x_t, x_0) = \frac{\sqrt{\bar{\alpha}_{t-1}}\beta}{1-\bar{\alpha}_t}x_0 + \frac{\sqrt{\bar{\alpha}_t}(1-\bar{\alpha}_{t-1})}{1-\bar{\alpha}_t}x_t.$$

There are various way to approximate $\mu_\theta(x_t, t)$ to $\mu_q(x_t, x_0)$. One can rewrite $\mu_\theta(x_t, t)$ as $\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}}(x_t - \frac{\beta_t}{\sqrt{1-\bar{\alpha}_t}}\epsilon_\theta(x_t, t))$. We simply train a neural network ϵ_θ to minimize $\|\epsilon - \epsilon_\theta(x_t, t)\|^2$. We call this neural network to be the diffusion model.

Conditional Free Guidance Very often, one would want to produce data based on their label. In other words, we are also interested in modeling $p(x \mid y)$ where y is the label. Especially in SCA, we would like to create a diffusion model that could obtain traces based on their leakage model. [10] first introduces the concept of Conditional Free Guidance (CFG) by ditching the idea of using a separate classifier to predict newly generated data and train two diffusion models. CFG trains both the unconditional diffusion model and conditional diffusion simultaneously (in practice, this is just one model). The idea is to let the unconditional diffusion model guide the conditional model for more exploration, which allows for more diversity. An equivalent goal when training a diffusion model is known as the score-based formulation, where the objective is to maximize the score $\nabla_{x_t} \log p(x_t \mid y)$. One can formulate this score as

$$\nabla_{x_t} \log p(x_t \mid y) = \gamma \nabla_{x_t} \log p(x_t \mid y) + (1-\gamma) \nabla_{x_t} \log p(x_t)$$

where the term $\nabla_{x_t} \log p(x_t \mid y)$ corresponds to the score of the conditional diffusion model while $\nabla_{x_t} \log p(x_t)$ is the score of the unconditional diffusion model, with γ being the hyperparameter that controls how much the conditional diffusion model cares about the label. Throughout this paper, we consider γ to be 0.7. We refer readers to [15] for a more holistic understanding of the diffusion model and CFG.

4 Datasets and Neural Network Used for Experiment

In this section, we present the datasets that will be used for any of the experiments and the neural network used as the diffusion model.

4.1 Datasets

Simulated traces We generate simulated traces of 24 sample points. These traces are leaking with the value of the Advance Encryption Standard (AES) substitution box. Suppose each trace is defined as an array, $trace[0, \ldots, 23]$, and $d \in \{0, 1, 2, 3\}$ to be the masking order of the simulated dataset. The sample points that consist of the values of the masks are presented in Table 1. Here, we denote $Z = Sbox(pt \oplus k^*)$ where pt is the plaintext byte and k^* is the correct key. Here, we fix $k^* = 0x03$. Furthermore, m_i are randomly generated bytes for secret sharing. We randomly generate random byte values for the remaining points. Then, we add a small noise sampled from the normal distribution to every sample point with zero mean and variance of 0.01. We generated 14,000 traces for training the diffusion models.

Masking Order d	**Leakage Point**
0	$trace[5, \ldots, 10] = Z$
1	$trace[10, \ldots, 15] = Z \oplus m_1$ $trace[0, \ldots, 5] = m_1$
2	$trace[0, \ldots, 4] = Z \oplus m_1 \oplus m_2$ $trace[9, \ldots, 14] = m_1$, $trace[18, \ldots, 22] = m_2$
3	$trace[0, \ldots, 3] = Z \oplus m_1 \oplus m_2 \oplus m_3$ $trace[7, \ldots, 9] = m_1$, $trace[13, \ldots, 17] = m_2$, $trace[18, \ldots, 22] = m_3$

Table 1: Leakage points of the traces generated in simulated data based on the masking order.

ChipWhisperer (CW) The CW dataset provides a standard comparison base for the evaluation of different algorithms [18]. The dataset we consider runs the unprotected AES-128 implementation on the CW308 Target. We will refer to this dataset as CW throughout this paper. This dataset targets the first byte in the first round of the AES Sbox, $Sbox(pt \oplus k^*)$, with a fixed key k^*. The dataset consists of $10,000$ traces.

ASCADf and ASCADr The ASCAD dataset consists of a first-order masked AES implementation on an 8-bit AVR microcontroller (ATMega8515) [2]. We target the third byte of the first round AES Sbox. This is a first-order masked key byte. Two versions known as ASCADf and ASCADr are part of the ASCAD dataset. ASCADf contains traces corresponding to the same fixed key for both profiling and attack. ASCADr contains profiling traces generated from a random key setting, while the attack traces are obtained from the fixed key target device. The dataset consists of $50,000$ profiling traces and $10,000$ attack traces for both datasets. The traces in ASCADf are composed of 700 sample points, while the traces in ASCADr consist of 1400 sample points.

4.2 Neural Network Used: UNet

We train UNet [21] as the diffusion model. A simple illustration of an UNet is shown in Figure 2. We consider the UNet to consist of 1-dimensional convolutional layers, where each convolution layer is followed by a group normalization [24] and the activation, SiLU [8]. In the UNet, we applied multiple skipped connections (see Figure 2). Furthermore, the UNet also contains attention mechanisms to improve its performance. We refer to the weblink for the full architecture used. We train the UNet together with an Exponential Moving Averages (EMA) [12] to help to enhance the stability of the training. In order for the neural network to understand which timestamp t the noise is from, it is first applied as a word embedding and followed by a shallow perceptron with GeLU as the activation function. Similarly, in order for the UNet to learn the information of the label y, we feed y into another shallow perceptron with GeLU [8] as the activation function. The output of these two embeddings are concatenated and fed to every convolutional layer of the UNet.

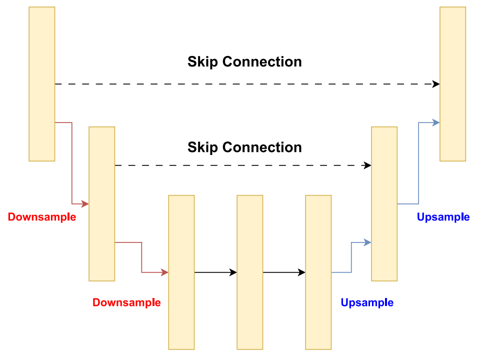


Fig. 2: UNet architecture.

5 Known Mask Setting: Evaluation of the Generated Traces Using CPA

In this section, we consider the scenario where we train a diffusion model under the known mask setting. Here, we can consider this trained diffusion model (together with the autoencoder) as a "portable oscilloscope" of the target device. If an adversary obtains this trained "portable oscilloscope" without access to any traces, they could essentially recover the secret key with the artificially generated data from this trained diffusion model.

In this part, we evaluate the quality of the generated traces from such diffusion model. To perform an evaluation of the quality, we conducted CPA on both original traces and generated traces. The idea is to observe if the generated traces can preserve the crucial leakages from the original traces. As such, while we are performing CPA in key recovery mode, we assume that for higher-order masking, the mask is known, and we assess if the mask leakage is also captured in the generated traces.

5.1 Evaluation Framework

Side-channel traces could go up to thousands of sample points. The time required to sample/generate new traces increases with the number of dimensionality. We propose to use the framework to be applied with CPA. We note that this framework can be used also in any non-profiling setting. This framework was first proposed in [20] to help speed up the sampling process in generating high-resolution images. The framework is as follows:

1. **(Autoencoder phase)** Train an autoencoder to encode and decode the traces into a latent space with reduced dimension.
2. **(Training phase)** We first encode every attack trace into the latent space by applying the encoder and train the diffusion model based on these attack traces.
3. **(Generative phase)** Next, for a given label class, we used the trained diffusion model to generate new latent embeddings by denoising randomly initialized embedding.

This framework introduces an autoencoder to help decrease the dimension of the traces into a latent space. This will help reduce the time required to sample new traces as the diffusion model generates a new latent embedding with smaller dimensional. We can then obtain the new traces by decoding the new latent embeddings.

Autoencoder phase To train the autoencoder, we have to first find the architecture of the model. To find the hyperparameters, we perform a random search (see Appendix A.1). In order to identify the best performing parameters, we use the trace correlation metric. In short, we compute the average correlation between the actual and reconstructed traces from the autoencoder and

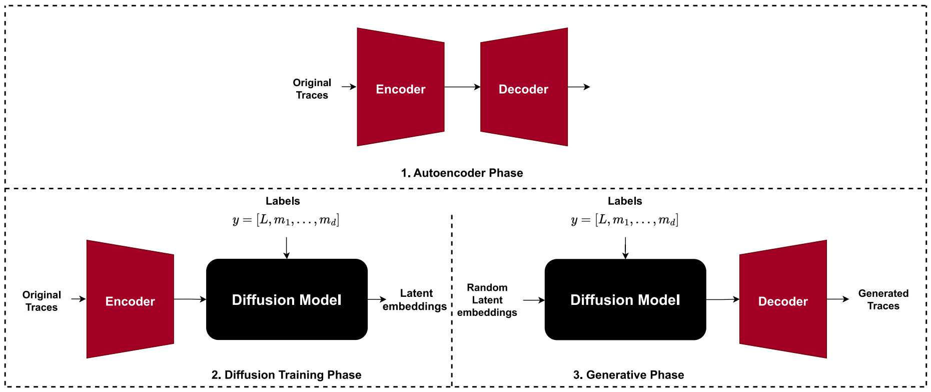


Fig. 3: Framework for using diffusion model in CPA. We define Z to be the corresponding mask data (e.g., $L = Z \oplus m_1 \oplus \cdots \oplus m_d$).

keep the parameter that resulted in the highest correlation. Given traces T and reconstructed T', we compute $r(T, T')$ as described in Equation 1.

After the autoencoder has been trained with the best performing parameters, we proceed with the trace generation using the diffusion model. We trained the diffusion model with the latent embedding instead of the actual traces. Once the model is trained with their proper labels, we can generate the new latent embedding with their corresponding labels. In the case of higher-order masking, for each trace, we generated corresponding mask share(s), and using this information, with knowledge of the secret, one can compute back the corresponding plaintext (or in the profiling setting, we can directly use the corresponding label for training). After the latent embeddings are generated, we use the autoencoder to decode these embeddings back to get the generated traces. If necessary, one can also normalize the generated traces as post-processing.

Training phase We train the diffusion model conditionally with a generalized CFG. In our setting, we consider a known mask setting where one would have the mask value and the mask data value. We set the label given to the diffusion to be both the mask and the mask data, namely $y = (Z \oplus m_1 \oplus \cdots \oplus m_{d-1}$, m_1,. . . , $m_d)$ where d is the masking order of the dataset and Z is the hypothetical sensitive values. We train the diffusion model according to Section 3.3. Note that $y, m_1, \dots, m_d$ are in real values when used to train the diffusion model.

Generative phase In this phase, one can choose the number of artificial data/latent embeddings one would want to generate.

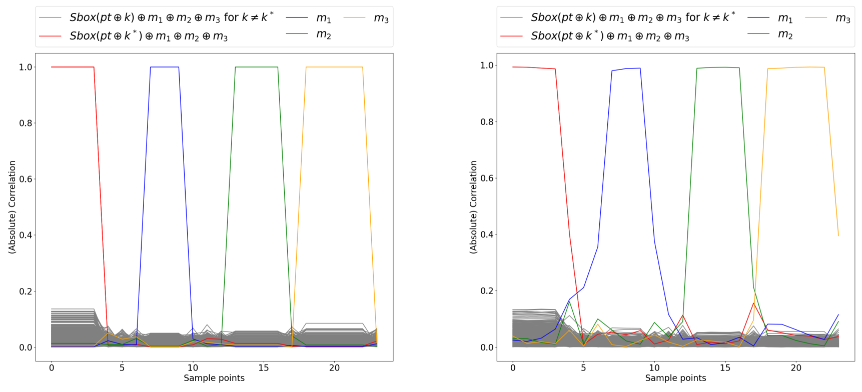
1. Firstly, we generate n_g latent embeddings for each value of L.
2. Then, we randomly generate values of each mask values $m_1, \dots, m_d$ and obtain the label $y = (L, m_1, \dots, m_d)$.

3. Feed the diffusion model with randomly generated latent embeddings and the label y to obtain denoised latent embeddings. Then, we decode these denoised latent embeddings to obtain the artificial traces.

5.2 Experiment Results

We apply our framework on simulated traces from higher-order masking and three datasets with real traces: CW, ASCADf and ASCADr.

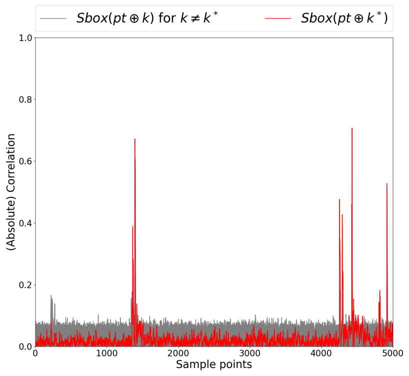
Simulated Traces We performed the assessment for masking order $0, 1, 2$ and 3 of the simulated traces. Since the results are similar for all the masking orders tested, we shall only present the results on masking order 3. Considering that the sizes of the simulated traces are small. We train the diffusion model without the use of an autoencoder. We train the diffusion model using a batch size of 512, a learning rate of 0.0005, and 50 epochs. We sample $n_g = 10$ artificial data for each value of $L = Sbox(pt \oplus k^*) \oplus m_1 \oplus m_2 \oplus m_3$ with randomly generated m_1, m_2 and m_3. We get a total of $256 * 10 = 2560$ artificial latent embeddings and apply the decoder to obtain the newly generated traces. We apply the CPA on these newly generated traces. The result of the CPA for each mask is shown in Figure 4. The correlation of the original traces is illustrated in Figure 4a while the correlation of the generated traces is depicted in Figure 4b. We see that the generated traces from the diffusion model obtain similar correlations on every share in comparison with the corresponding correlations within the original traces. In fact, we see that the correlation of mask m_1 is amplified within around the sample point 11. This shows that the diffusion model could potentially generate traces that amplify the leakages.



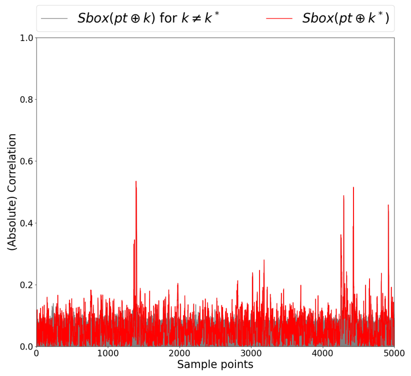
(a) Original Traces. (b) Generated Traces.

Fig. 4: CPA on original and generated simulated data of order 3 (with known shares).

CW We then test the approach on real datasets. Our first target is the CW dataset. Using random search, we found an autoencoder that mapped the traces into a latent embedding of size 992. Afterward, we generate new traces using the diffusion model. Here, we are using the ID leakage model, resulting in 256 classes. We then generate 2,000 traces in total using the diffusion model. We perform CPA on the original traces as well as on the generated traces, both using 2,000 traces in total. In Figure 5a, we show the CPA on original CW traces, and in Figure 5b, we show the CPA on generated CW traces. Overall, we could see that the generated leakage could preserve the important leakages from different sample points. Similarly, we also see the correlation with respect to $Sbox(pt \oplus k^*)$ is amplified in areas that are not correlated previously, showing that diffusion models could possibly increase the intensity of the leakages in areas that are previously not correlated.



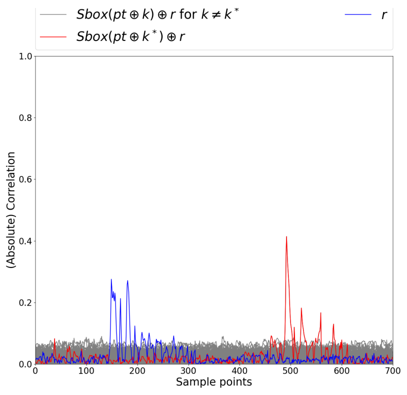
(a) Original Traces.



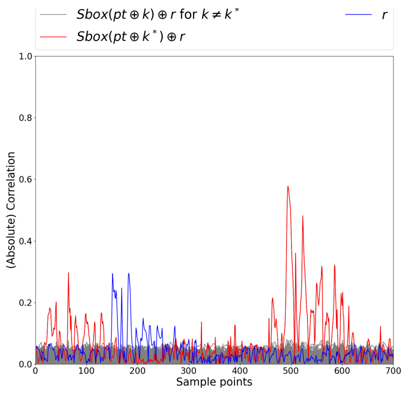
(b) Generated Traces.

Fig. 5: CPA on original and generated CW data.

ASCADf Next, we test the approach on the ASCADf dataset. Similar to the previous experiment, we employ random search on the hyperparameters to find an autoencoder. We then constructed an autoencoder that mapped the traces into a latent embedding of size 192. Afterward, we generate new traces using the diffusion model. Again, we are using the ID leakage model with 256 classes. We then generate for each class 10 traces, so we have 2,560 traces in total. We perform CPA on the original traces as well as on the generated traces, both using 2,560 traces in total. In Figure 6a, we showed the CPA on original ASCADf traces, and in Figure 6b, we showed the CPA on generated ASCADf traces. Here, we can observe that the leakage of the intermediate value, as well as the mask, can be preserved in the generated leakage. In observe that there is increase in correlation with respect to $Sbox(pt \oplus k^*) \oplus r$. Especially between sample points



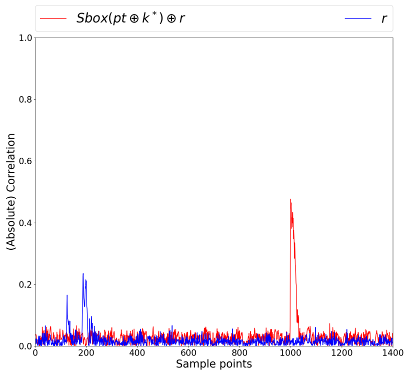
(a) Original Traces.



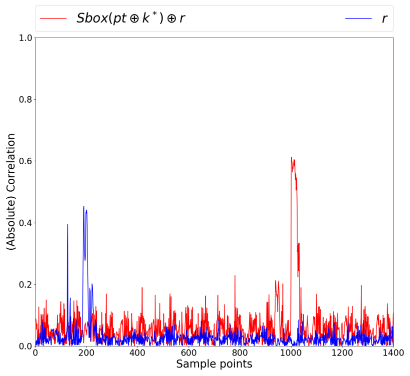
(b) Generated Traces.

Fig. 6: CPA on original and generated ASCADf data of order 1 (with known shares).

0 to 200, this could be that the diffusion model learns the leakages there and amplified it in those areas. We leave the study of explainability of diffusion models to future works.



(a) Original Traces.



(b) Generated Traces.

Fig. 7: CPA on original and generated ASCADr data of order 1 (with known shares).

ASCADr Lastly, we test the approach on ASCADr dataset. Since the key is not fixed, we cannot perform key recovery, instead, we plot the correlation on the intermediate value and the mask for the original (Figure 7a) and the generated traces (Figure 7b). Here, we can observe that similar to the ASCADf case, the generated traces can preserve the leakages from the original traces.

In general, from the experiments conducted on these three datasets, we can clearly observe that the diffusion model can generate new traces that can preserve and even amplified the important leakages from the original dataset in the known mask setting.

6 Unknown Mask Setting: Profiling Attack

We will first provide the framework when using DDPM in a profiling attack, followed by the experimental results. As mentioned earlier, we will be using the ID leakage model throughout this section.

6.1 Framework for Profiling Setting

In this section, we explore the effectiveness of using a diffusion model in the profiling attack setting. Figure 8 depicts the overall framework of using a diffusion model in a typical profiling setting. The framework for profiling can be described as follows:

1. Use a dimensionality reduction on the profiling traces to obtain its corresponding latent embeddings.
2. Train the diffusion models with the profiling latent embeddings.
3. Generate new latent embeddings with the trained diffusion model. If necessary, we can also normalize the generated traces as post-processing.
4. Use these new latent embeddings with the original latent embeddings as the new profiling set for the profiling attack.

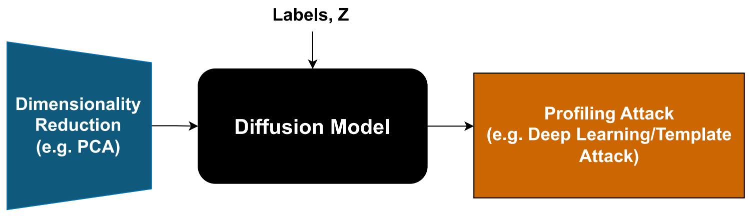

Labels, Z
Dimensionality Reduction (e.g. PCA)
Diffusion Model
Profiling Attack (e.g. Deep Learning/Template Attack)

Fig. 8: Framework for using diffusion model in Profiling attack. Z is defined to be the hypothesis sensitive variable (e.g., $Z = Sbox(pt \oplus k^*)$)

We choose Principle Component Analysis (PCA) as the dimensionality reduction technique. For each of the datasets, we pick the dimension that managed to recover the key with the least $NTGE$ required. Therefore, for each dimension ranging from 8 to 48 with an interval of 8, we apply PCA and subsequently perform TA.

We note here that the labels used to guide the diffusion model are the same as the ones used in the profiling attack. This means that this framework is in an unknown mask setting, unlike the previous framework in Section 5.1. The label y is the hypothetical sensitive variable Z.

In order to test the efficacy of the diffusion model for profiling attacks, we apply TAs in various settings. We use TA over the deep learning-based attack, as the deep learning-based attack has too many factors that could affect the $NTGE$ attained. For example, even when using the same architecture, the weights are randomly initialized, which could result in different performances in two different training. Therefore, TA is a much better baseline for exploring diffusion models' effectiveness than using deep learning-based profiling attacks.

Various Setting Tested We apply TA in three different settings. Firstly, we perform TA on the original dataset. We denote this as Original. Next, we decrease the dataset to balance every class to the minimum number of traces within a class. For example, in ASCADf, the class 213 obtains the least number of traces with 139 traces. Therefore, we downsample every class to 139 and obtain traces of size $139 \times 256 = 35584$ for profiling. This is to simulate when there is a lack of profiling traces. Then, we run TA on this downsampled dataset. We call this setting Downsampled. As the next step, we want to determine if using the diffusion model to generate new latent embedding would help in TA. We simply double the traces of each class within the downsampled dataset. We denote this scenario as Downsampled+Generated Latent. Subsequently, we train all our diffusion models with a batch size of 200, a learning rate of 0.001, and 2000 epochs for each dataset experimented.

6.2 Experiment Results for Profiling Attacks

ASCADf We consider the use of PCA to compress the dimension of the traces to 24 as it successfully breaks the dataset and obtains the least number of $NTGE$ with TA. We presented the results in Table 2. Here, we observed that Downsampled+Generated Latent slight improvement over the Downsampled scenario. This shows that when the adversary has the diffusion model and with limited traces, it could improve the attack.

	Original	**Downsampled**	**Downsampled+Generated Latent**
NTGE	$1,194$	$1,552$	$1,531$
Dataset Size	$45,000$	$35,584$	$71,168$

Table 2: $NTGE$ for ASCADf when applying TA in the various setting.

ASCADr For ASCADr, we compress the dimension to 16. We show the performance in Table 3. Similarly to the above, we see an improvement in $NTGE$ when adding newly generated latent from the diffusion model into the downsampled dataset. Since there is a reduction of approximately 200 traces in the $NTGE$. This shows that the generated latent/traces with the diffusion model could help to slightly improve the performance when the number of traces is limited to build the template.

	Original	**Downsampled**	**Downsampled+Generated Latent**
NTGE	$3,953$	$4,742$	$4,598$
Dataset Size	$45,000$	$34,816$	$69,632$

Table 3: $NTGE$ for ASCADr when applying TA in the various setting.

ASCADf_desync50: Here, we consider desynchronization within the ASCADf dataset, denoted as ASCADf_desync50. The dataset is created by considering random desynchronization up to 50 sample points in each trace within the raw traces before extracting the 700 sample points. For ASCADf_desync50, we reduce the dimension of the traces to a size of 48. Table 4 shows the performance results of the TA for ASCADf_desync50. Surprisingly, we observe that Downsampled+Generated Latent obtained a significant decrease in $NTGE$. The $NTGE$ decreases by around 4000 traces, which is almost half of the $NTGE$ attained when training with the original dataset. When adding the new latent created by the diffusion model into the downsampled dataset, it even attained the best $NTGE$ out of all the three settings. This shows that the diffusion model is effective even in desynchronized datasets.

	Original	**Downsampled**	**Downsampled+Generated Latent**
NTGE	$9,606$	$9,017$	$5,730$
Dataset Size	$45,000$	$35,584$	$71,168$

Table 4: $NTGE$ for ASCADf_desync50 when applying TA in the various setting.

7 Conclusion and Future Works

In this work, we have investigated and explored the applicability of using the DDPM approach for artificial trace generation in the context of SCA. We have conducted the study on two different frameworks, namely the known and unknown mask settings. We then performed the experiments on several datasets to create a new set of traces using the diffusion model. Our experimental results have shown that the generated traces can preserve the original leakages in the known mask setting. We have also demonstrated that in the unknown mask setting, the diffusion model can create artificial data that can help to improve the profiling attack, suggesting that leakages are learned within the generated data. In the future, we will investigate more on improving the performance of the proposed DDPM approach; for example, one direction is to optimize or speed up the sampling process. We would like to investigate if this can also be adapted for portability scenarios on custom traces from different setups.

Acknowledgment

This research is supported by the National Research Foundation, Singapore, and Cyber Security Agency of Singapore under its National Cybersecurity Research & Development Programme (Cyber-Hardware Forensic & Assurance Evaluation R&D Programme ¡NRF2018NCRNCR009-0001). Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the view of National Research Foundation, Singapore and Cyber Security Agency of Singapore

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM Side—Channel(s). In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002. pp. 29–45. Springer Berlin Heidelberg, Berlin, Heidelberg (2003)
2. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptogr. Eng. **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8, https://doi.org/10.1007/s13389-019-00220-8
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Cryptographic Hardware and Embedded Systems-CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings 6. pp. 16–29. Springer (2004)
4. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures: Profiling attacks without preprocessing. In: Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings. pp. 45–68. Springer (2017)
5. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers 12. pp. 253–270. Springer (2014)
6. Dhariwal, P., Nichol, A.: Diffusion models beat gans on image synthesis (2021)

7. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., Bengio, Y.: Generative Adversarial Networks (2014)
8. Hendrycks, D., Gimpel, K.: Gaussian error linear units (gelus) (2023)
9. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models (2020)
10. Ho, J., Salimans, T.: Classifier-free diffusion guidance (2022)
11. Karayalcin, S., Krcek, M., Wu, L., Picek, S., Perin, G.: It's a kind of magic: A novel conditional gan framework for efficient profiling side-channel analysis. Cryptology ePrint Archive (2023)
12. Klinker, F.: Exponential moving average versus moving exponential average. Mathematische Semesterberichte **58**(1), 97–107 (Dec 2010). https://doi.org/10.1007/s00591-010-0080-8, http://dx.doi.org/10.1007/s00591-010-0080-8
13. Kocher, P.C., Jaffe, J., Jun, B.: Differential Power Analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. p. 388–397. CRYPTO '99, Springer-Verlag, Berlin, Heidelberg (1999)
14. Li, H., Perin, G.: A systematic study of data augmentation for protected aes implementations. Cryptology ePrint Archive (2023)
15. Luo, C.: Understanding diffusion models: A unified perspective (2022)
16. Mukhtar, N., Batina, L., Picek, S., Kong, Y.: Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. In: Cryptographers' Track at the RSA Conference. pp. 297–321. Springer (2022)
17. Nichol, A., Dhariwal, P.: Improved denoising diffusion probabilistic models (2021)
18. O'Flynn, C., Chen, Z.D.: ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. In: International Workshop on Constructive Side-Channel Analysis and Secure Design (2014)
19. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems pp. 209–237 (2019)
20. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models (2022)
21. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation (2015)
22. Standaert, F.X., Malkin, T.G., Yung, M.: A Unified Framework for the Analysis of Side-Channel Key Recovery Attacks. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009. pp. 443–461. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
23. Wang, P., Chen, P., Luo, Z., Dong, G., Zheng, M., Yu, N., Hu, H.: Enhancing the Performance of Practical Profiling Side-Channel Attacks Using Conditional Generative Adversarial Networks (2020)
24. Wu, Y., He, K.: Group normalization (2018)

A Appendix

A.1 Random Search Hyperparameters for Autoencoder

We conducted random search to find hyperparameters for the autoencoder. In the following table, we listed down the range of values used for each parameters.

Parameters	Start	Max	Step
Number of Layers	2	6	1
Number of Batch Size	64	2048	32
Embedding Size	128	Trace length	32
Epoch Size	40	100	1
Learning Rate (10^x)	-5	-2	1
Node Size per Layer	32	2048	32

Table 5: Range for hyperparameter random search

A.2 Hyperparameters Used on Autoencoder

In the following tables, we reported the hyperparameters found through random search.

Parameters	CW	ASCADf	ASCADr
Number of Layers	3	3	3
Number of Batch Size	896	800	1536
Embedding Size	992	192	256
Epoch Size	86	97	97
Learning Rate (10^x)	-4	-3	-3
Size of Nodes	$[928, 448, 992]$	$[704, 992, 192]$	$[704, 992, 256]$
Correlation	0.87	0.97	0.85

Table 6: Hyperparameters used for each dataset, found through random search