Enhancing the Performance of Practical Profiling Side-Channel Attacks Using Conditional Generative Adversarial Networks

Ping Wang, Ping Chen, Zhimin Luo, Gaofeng Dong, Mengce Zheng, Nenghai Yu, Honggang Hu

Key Laboratory of Electromagnetic Space Information, CAS
University of Science and Technology of China, Hefei, China
{wangpin,mczheng}@mail.ustc.edu.cn,hghu2005@ustc.edu.cn

Abstract. Recently, many profiling side-channel attacks based on Machine Learning and Deep Learning have been proposed. Most of them focus on reducing the number of traces required for successful attacks by optimizing the modeling algorithms. In previous work, relatively sufficient traces need to be used for training a model. However, in the practical profiling phase, it is difficult or impossible to collect sufficient traces due to the constraint of various resources. In this case, the performance of profiling attacks is inefficient even if proper modeling algorithms are used.

In this paper, the main problem we consider is how to conduct more efficient profiling attacks when sufficient profiling traces cannot be obtained. To deal with this problem, we first introduce the Conditional Generative Adversarial Network (CGAN) in the context of side-channel attacks. We show that CGAN can generate new traces to enlarge the size of the profiling set, which improves the performance of profiling attacks. For both unprotected and protected cryptographic algorithms, we find that CGAN can effectively learn the leakage of traces collected in their implementations. We also apply it to different modeling algorithms. In our experiments, the model constructed with the augmented profiling set can reduce the required attack traces by more than half, which means the generated traces can provide useful information as the real traces.

Keywords: Profiling side-channel attacks · CGAN · Generated traces · Leakage learning · Insufficient profiling traces

1 Introduction

Side-Channel Attacks (SCA) was first introduced by Paul Kocher [Koc96] in 1996, conducting a timing attack on the cryptographic device to recover the secret information. This provides a new direction for the security evaluation of the cryptographic algorithm when considering physical implementation. In addition to timing, the leakages available for SCA also include power consumption and electromagnetic emanations, etc. SCA mainly includes two categories: *profiling* attacks and *non-profiling* attacks. Non-profiling attacks mean that we recover secret information directly by analyzing the leakages of an unknown device and during the process, we need to guess the secret key. Typical non-profiling attacks include Differential Power Analysis (DPA) [KJJ99], Correlation Power Analysis (CPA) [BCO04], and Mutual Information Analysis (MIA) [GBTP08], etc. Profiling attacks (e.g. Template Attacks (TA) [CRR02], and Stochastic Attacks (SA) [SLP05]) happen in the circumstance where the same fully-controlled profiling device exists before recovering the secret information used in a targeted device. Then we characterize the leakages of the controlled device by modeling algorithms and recover the secret information through the constructed model.

With the development of SCA, there appeared profiling attacks based on *Machine Learning* (ML) [HGM+11, BL12, LMBM13, LBM14], in which two commonly used models are SVM [CV95, WW98] and RF [Bre01]. Recently, *Deep Learning* (DL) based profiling attacks raises the concern of SCA community [MPP16, CDP17, PSB+18, KPH+19, PHJ+19]. Compared to classical TA, DL-based profiling attack is no need for the Points of Interest (POI) or the traces alignment, and it can also handle high-dimensional data. However, we find that these works focus on how to propose better modeling algorithms, or to improve the current modeling algorithms, in order to reduce the number of traces required for successful attacks. In fact, the traces they use in the profiling phase are relatively sufficient, which seems rational. Because the premise of profiling attacks is that the attackers have a fully-controlled profiling device and are able to obtain sufficient traces to construct the model.

However, if the profiling attack is applied to a specific application, due to various constraints like time, resources, and countermeasures, we cannot collect enough traces. For example, an encryption operation to be attacked may only be a subroutine in the entire application, but the encryption operation has many pre-operations, so in the process of collecting traces, it is necessary to wait for the completion of the entire pre-operation. Even the position of the encryption operation in the entire application is not fixed. In this case, the trace of the entire process executed by the application needs to be collected, so each trace collection requires a huge time cost. Besides, in some application scenarios, the key has a life cycle. For example, the session key used when communicating between two devices, then the number of traces collected for each session key is limited, depending on how long the session time is. In such a scenario, the attack performance of the model constructed with insufficient traces is inefficient, and even an effective attack model cannot be constructed. Consequently, the attack performance of the model constructed by various approaches based on ML and DL mentioned above will be greatly decreased when the traces are lacking during the profiling phase.

Therefore, we consider whether it works by using the existing insufficient profiling traces to generate new traces with more useful information. If it is possible, we can use the generated traces to expand the original profiling set, to build a more efficient model. We find that Generative Adversarial Networks (GAN) [GPM+14] proposed in 2014 has the ability to generate new data with additional information from raw data. GAN has a very rapid development in recent years and derives hundreds of different networks. Using GAN to generate data is mainly utilized in the field of image [ASE17, FKA+18, BCG+18]. For example, Bowles *et al.* used GAN to generate medical images [1], which obtained satisfactory results [BCG+18]. Thus, we try to use GAN to generate traces, to deal with the problem that sufficient profiling traces cannot be collected in the practical profiling attack. It should be noted that during the profiling phase, each trace has a corresponding label, thus we use Conditional Generative Adversarial Networks (CGAN) [MO14] which can specify the labels of the generated traces. To the best of our knowledge, the method of using CGAN to generate traces has not yet been studied in the context of SCA. In our experiments, we first explore how to generate effective traces, and propose methods to evaluate the quality of generated traces. Our results show that CGAN can effectively learn the leakages of traces and generate traces with additional useful information, thereby improving the attack performance in scenarios where profiling traces are insufficient.

[1] Because the annotation of medical images is expensive and time-consuming, it is impossible to get a lot training samples.

1.1 Related Work

The method of generating new traces then added to the original profiling set is also called Data Augmentation (DA) [SSP03]. In recent years, the DA technique has been applied to profiling attacks. In 2017, Cagli *et al.* mitigated overfitting and obtained better attack performance by randomly shifting traces, adding clock jitter, and then adding deformed traces to the original profiling set [CDP17]. In 2019, Picek *et al.* used the Synthetic Minority Oversampling Technique (SMOTE) to process imbalanced data caused by using Hamming weight (HW) (or the Hamming distance (HD)) [PHJ+19]. At the same time, we notice that Picek *et al.* proposed a restricted attacker framework where they restrict the ability of an attacker to obtain measurements in the profiling phase [PHG19]. Although we both study similar issues, the methods used are completely different. In fact, the DA techniques previously mentioned improving attack performance are used in the scenario of relatively sufficient profiling traces. It should be noted that we concentrate on the problem of how to build a more efficient model when the collected traces are insufficient during the profiling phase. We use the limited profiling set to generate new traces with additional useful information to effectively improve the attack performance.

1.2 Our Contributions

Our contributions mainly include the following aspects:

1. We give a new perspective against the problem that how to build a more efficient model with insufficient traces in the practical profiling attacks. Then we first introduce CGAN (GAN) in the SCA context, and use it to generate traces to solve the problem of insufficient profiling traces, which improves the potential to apply profiling attacks in real scenarios.
2. We present a detailed discussion on the label used to generated traces and evaluating the quality of generated traces. Our results show that the number of traces required for successful attacks can be reduced by more than half in the scenario of insufficient profiling set.
3. For various types of AES implementations, such as unprotected, first-order masked protected, and random delay protected ones, we find that CGAN can effectively learn the leakages contained in traces.
4. We have tried different models based on ML and DL, and find that our method is suitable for different modeling algorithms.

1.3 Outline

The paper is organized as follows. In Section 2 we give an overview of profiling attacks, evaluation metrics, DA and CGAN (GAN). In Section 3, we describe the basic steps of using CGAN to generate traces and improve the attack performance and then verify the feasibility of the method through simulation experiments. In Section 4, through the experiments with real datasets, we verify that our method can improve the performance of profiling attacks in the case of insufficient profiling traces. The applicability of this method to different AES implementations and different modeling algorithms is also studied. In Section 5, we give a discussion. Finally, in Section 6 we have a summary of the proposed method and the future outlook.

2 Background

2.1 Profiling Side-Channel Attacks

We first briefly introduce the TA which was proposed in 2002 by Chari *et al.* [CRR02] and is also the earliest profiling attack. TA consists of two phases: the profiling phase and the attack phase. During the profiling phase, it assumes that before the secret key of an unknown device is recovered, an identical device already exists and is fully controlled. The attackers calculate the intermediate values based on the plaintexts/ciphertexts and the fixed possible key and then divide the collected traces (or profiling set) according to the intermediate values. Finally, templates are established by using a Multivariate Gaussian distribution to characterize the relationship between the intermediate values and the leakages. In the attack phase, the attackers collect the traces of the unknown device and use the templates to recover the secret key used [2]. Subsequent profiling attacks based on ML and DL are basically the same as TA except that the modeling algorithms used are different.

Formally, during the profiling phase, the attackers get a profiling set $T_{profiling} = \{\vec{l_i} | i \leq N_p\}$ with known plaintexts/ciphertexts $P_{profiling} = \{p_i | i \leq N_p\}$ and the fixed key k_p^*. Function φ computes the intermediate value z_i between p_i and k_p^*. Then, the model is constructed by the modeling algorithm F:

$$M_z = F(\vec{l_i}) \ where \ z = \varphi(p_i, k_p^*). \tag{1}$$

M_z is a model constructed with all the traces corresponding to the same intermediate value z, consequently we obtain models for each different intermediate value.

Similarly, we assume the attackers get attack set $T_{attack} = \{\vec{l_j} | j \leq N_a\}$ with known plaintexts/ciphertexts $P_{attack} = \{p_j | j \leq N_a\}$ during attack phase. Then we use *Maximum Likelihood* strategy to calculate the score S for each guessed key k_{guess}:

$$S_{k_{guess}} = \prod_{j=1}^{N_a} M_{\varphi(p_j, k_{guess})}(\vec{l_j}). \tag{2}$$

The k_{guess} that maximizes S is the key that the model thinks correct:

$$k_{real} = \underset{k_{guess}}{\arg\max}\, S_{k_{guess}}. \tag{3}$$

Note that, ML and DL-based profiling attacks usually build only one model, but will output scores (or probability) corresponding to all intermediate values in the attack phase, which is equivalent to a model including all models M_z.

2.2 Evaluation Metric

In ML and DL, accuracy is often used as an evaluation metric. Although we use the ML and DL based methods in profiling attacks, some researchers have found that using accuracy to evaluate SCA is not suitable [PHJ+19]. Hence, we will use Guessing Entropy (GE) [SMY09], one of the common SCA metrics. In simple terms, GE is to calculate the averaged rank of the correct key over several experiments, and the term rank refers to the score ranking of all key hypotheses given by the model. What we usually do is to calculate the average number of traces required when the rank converges to zero. In our experiments, after CGAN learns the distribution of the original data, the generated traces each time are different and the improvement of the attack result also changes slightly. Therefore, in each experiment, we will generate new traces and the original training set and testing set are randomly selected. Then we average the attack results of 10 experiments as GE.

[2] We usually use a divide-and-conquer method to recover the subkey first.

2.3 Data Augmentation

Data Augmentation [SSP03] is a regularization technique used to deal with the overfitting problem, in order to build a more robust Model. Naturally, this method is often used in the field of image. For example, Krizhevsky *et al.* used two forms of DA techniques to effectively reduce overfitting, one of which is image translations and horizontal reflections, and the other is altering the intensities of the RGB channel [KSH12]. Since GAN was introduced in 2014, some GAN-based Data Augmentation has been proposed and used to acquire good results [ASE17, FKA$^+$18, BCG$^+$18]. In the field of SCA, there have also been some researches [CDP17, PYW$^+$17, PHJ$^+$19] on the use of DA technique recently. Besides, it should be emphasized that our main concern is how to build a more efficient model when sufficient traces cannot be collected during the profiling phase, instead of reducing overfitting [3].

2.4 Conditional Generative Adversarial Networks

In this section, we mainly introduce the basic principles of Generative Adversarial Networks (GAN) and Conditional Generative Adversarial Networks (CGAN). GAN [GPM$^+$14] was proposed in 2014 and mainly used as a generative model to learn the distribution of data. GAN includes two parts, generator G and discriminator D. Assuming that the real data is x and its distribution satisfies p_{data}, the role of the generator is to learn the distribution of real data p_{data} through a non-linear function $G(z;\theta_g)$ which maps a prior noise distribution $p_z(z)$ to p_{data}. Assuming that the distribution of the learned data through the generator is p_g, all we have to do is to make p_g as close to p_{data} as possible. The discriminator is also a non-linear function $D(x,\theta_d)$, giving the probability that a sample comes from p_{data} rather than p_g. Then we alternately train the discriminator and generator, and adjust the parameters θ_d and θ_g to maximize $\log D(X)$ and minimize $\log(1-D(G(Z)))$. This training process can be regarded as a mini-max game between the generator G and the discriminator D. Using an objective function to express it is:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1-D(G(z))\right)\right]. \tag{4}$$

GAN is actually a kind of unsupervised learning, and the generated data has no label. However, the profiling SCA based on ML and DL is supervised learning which means every trace has a corresponding label obtained through the intermediate value and energy model. Therefore, each generated trace must also have a corresponding label so that the original profiling set can be expanded. Consequently, we use a conditioned version of GAN, namely CGAN [MO14], to generate traces. The difference between CGAN and GAN is that in CGAN when training G and D, we add additional information (or auxiliary information) y, which can be the label of the data or the data under a certain distribution. Then, the target function becomes:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}\left[\log D(x|y)\right] + \mathbb{E}_{z\sim p_z(z)}\left[\log\left(1-D(G(z|y))\right)\right]. \tag{5}$$

Figure 1 simply shows the architecture of CGAN. After using CGAN, we can specify label and produce the corresponding traces.

3 CGAN on SCA

In this section, we outline the overall process of how to use CGAN to enhance the performance of profiling attacks when the profiling traces are insufficient and verify the

[3] In fact, when we use CGAN to generate traces and expand the original profiling set, the overfitting problem will also be alleviated, but this is not in conflict with the main problem we are aiming at.

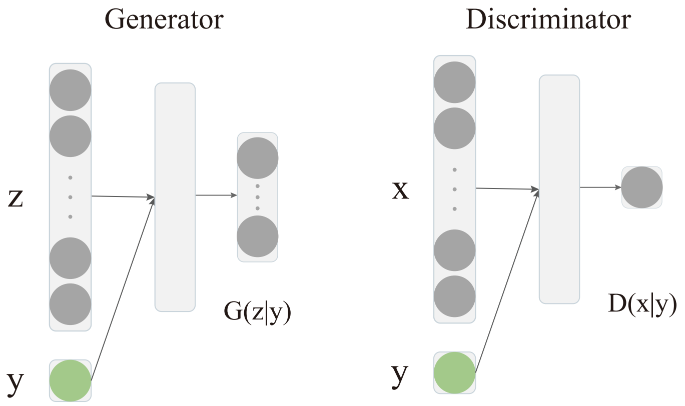

Figure 1: Conditional Generative Adversarial Network.

feasibility of this method by simulation experiments. Then, we discuss some experimental parameters that need attention when traces are generated.

3.1 The Basic Steps

3.1.1 Choosing an Insufficient Training Set

First, we divide the traces into two parts, a set of training traces and a set of testing traces, respectively corresponding to the profiling phase and the attack phase. Afterward, we reduce the size of training set as much as possible to simulate the scenario of insufficient profiling traces. In our experiments, we use the training sets of different sizes to build models in advance and compare the attack performance on the same testing set. Then we select a training set with general attack performance whose set size is relatively small [4]. In fact, when adequate traces cannot be collected during the profiling phase, all traces would be utilized maximally to construct the model.

3.1.2 The CGAN Structure

For the generator and discriminator, it is mainly composed of fully connected layers, and no more complex layer is used. However, in the process of CGAN training, instability is probable to occur. Here we use some techniques: (1) Use Batch Normalization (BN) in both Generator and Discriminator [RMC16], (2) The prior noise is sampled from the Gaussian distribution rather than Uniform distribution [Whi16], (3) Use the leaky rectified activation (LeakyReLU) [MHN13] as the activation function for the hidden layer, (4) Use the Adam optimizer [KB15] to train the model, (5) Use the Dropout layer in generator during both training and testing phases [IZZE17]. In addition, tuning CGAN hyperparameters is similar to tuning classifiers. The difference is that the optimal parameters should be selected according to the performance of the model constructed with the augmented training set. At the same time, it should be noted that we should use the evaluation metrics related to side-channel attacks. The specific model structure of CGAN is given in Appendix A.1.

3.1.3 Evaluating the Generated Traces

After traces are generated, there arises a question of how we evaluate the quality of the generated traces. When using the technology related to GAN to generate images, there

[4] We apply the same strategy in subsequent experiments, i.e., to choose the size of the training set beforehand and use a relatively small training set to simulate the lack of traces during the profiling phase.

are many common evaluation metrics, such as Accuracy, IS [SGZ+16], FID [HRU+17]. However, these metrics are not necessarily applicable for SCA. For example, paper [PHJ+19] verified that ML evaluation metrics are not precise for sure in SCA evaluations, such as accuracy. Even if the generated traces are similar to the real traces, we cannot decide whether the generated traces can contribute to the model. Therefore, in the final analysis, we need to add the generated traces to the original profiling set and see if the attack results are improved, in order that we can determine whether the generated traces are effective. That is to say, the quality of the generated traces comes down to the extent of improvement they bring to the attack performance.

Additionally, there are other intuitive methods to estimate the quality of generated traces. One of them is to visually observe the shapes of generated traces. If the generated traces seem largely different from the original traces, such generated traces are not expected to yield better improvement. This is because the purpose of GAN is exactly to learn the distribution of the original data and the generated objects should be similar to the original objects. Another method is to use Correlation Power Analysis (CPA) [BCO04] to separately calculate the correlation of original traces and generated traces, then compare the location of the leaks and the intensity of the correlation to verify whether the leakages have been learned. The paper [Tim19] also mentioned that the leakage points make the greatest contribution to modeling, thus observing the correlation of the generated traces at the leakage points is the simplest and most effective way to assess the quality of the generated traces. Based on multiple experiments, we discover that if the CPA results between the generated traces and the original traces are closer, the enhancement of the attack performance after data expansion is more obvious.

We also tried to use DPA [KJJ99] to evaluate whether the generated traces have learned effective leakage, and the experimental results show that it is indeed feasible. See Appendix A.2 for the detailed results of using DPA to evaluate the generated traces.

3.1.4 The Way to Improve the Attack Performance

We must emphasize that we are talking about the same model, that is, the ways are on the basis of the premise that the overall structure of the model is unchanged [5]. Without considering various pre-processing methods, then the size of the profiling set used to construct the model determines the performance of the attack.

The idea of the method is simple, we can utilize the existing insufficient profiling traces to generate new traces and expand original profiling set to obtain better attack performance. Let T_{raw} be the original profiling set, M_{raw} be the model built on T_{raw}, $P(M_{raw})$ be the attack performance of the model M_{raw}. Let $T_{generated}$ be the traces generated by T_{raw}. Then we can build model $M_{raw+generated}$ with T_{raw} and $T_{generated}$. Finally, if the generated trace is effective, we can get $P(T_{raw} + T_{generated}) > P(T_{raw})$.

3.2 Simulation Experiments

Firstly, we generate $N = 10000$ simulated traces [6], each of which has 50 time samples, and is related to the fixed key k^*. The plaintext p_i related to each trace is randomly generated between $0 \sim 255$ and the amplitude of each time sample is also a randomly value between $0 \sim 255$. The leakage point is located at $t_{leakage} = 25$, and the amplitude of the leakage point is $Sbox(k^* \oplus p_i) + \mathcal{N}(0, 1)$.

In the experiment, two disjoint sets are randomly divided out from the first 5000 simulated traces, the training set with 500 traces and the validation set with 2000 traces.

[5] Our goal is not to find the optimal model, but to figure out whether using generated traces to expand original training set in a scenario where profiling traces are insufficient can improve the attack performance. Therefore, we ought to assume that the model structure remains the same.

[6] The method of simulated traces generation is referred to in the paper [Tim19].

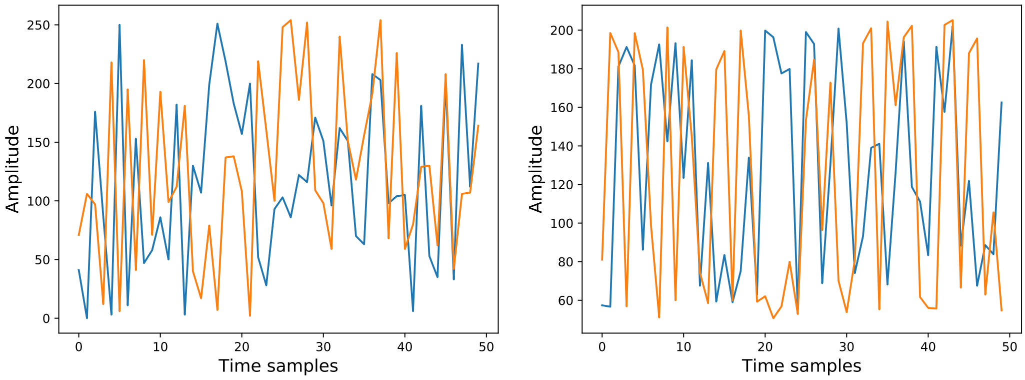

Figure 2: Left: original traces. Right: generated traces.

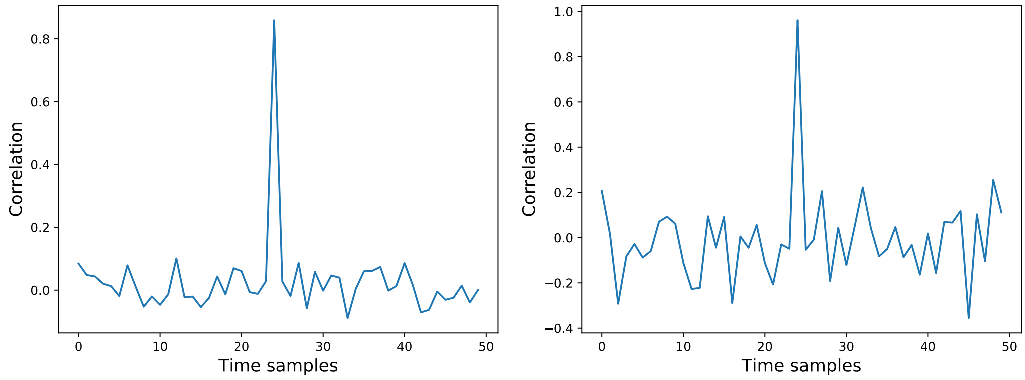

Figure 3: Left: CPA of 500 original traces. Right: CPA of 400 generated traces.

Besides, 2000 traces are divided out from the last 5000 simulated traces as the testing set.

Secondly, we build an MLP model with two hidden layers and the numbers of neurons in the two layers are separately 70, 50 and the label is Least Significant Bit (LSB) of $Sbox(k^* \oplus p_i)$. Following the steps in Subsubsection 3.1.1, we choose the size of the training set to be 500 to simulate the scenario where the profiling traces are insufficient.

Thirdly, we use the 500 traces of the original training set to generate 400 traces, among which 200 traces are with label 0 and another 200 traces are with label 1 [7]. We randomly choose two traces from the original training set and the generated traces separately and they are shown in Figure 2.

Then, we continue to analyze the correlation of the 500 original training traces and the 400 generated traces, and compare the CPA results in these two cases and determine whether the leakages have been learned through the currently generated traces. The CPA results are illustrated in Figure 3.

It indicates that in the CPA using the generated traces, the location of the leakage point and the amplitude of leakage peak are consistent with the CPA using the original training set. Meanwhile, we also see that the correlation of other time samples is higher in the CPA using the generated traces, and we call it *leakage noise*. The reason for the

[7] In the later experiments, we will specify that the number of generated traces with labels 0 and 1 are equal to each other when we use LSB as the label.

leakage noise may be that when CGAN learns the distribution of the original traces, the generator would think that it is easier to fool the discriminator when the leakages appear at more locations. In the later experiments, we will prove that the generated traces with high leakage noise are unable to improve the performance of the modeling attack, or even reduce the performance.

Fourthly, as we point out in Subsubsection 3.1.3, although the CPA results of the generated traces are close to that of the original traces, applying such generated traces for trace augmentation does not mean that the attack performance will definitely be improved. As a consequence, we need to decide whether our method is feasible by the practical attack results. For comparison, we randomly select 400 traces from the original 500 traces and add the 400 traces to the original training set to confirm that the addition of generated traces does improve the attack performance. The specific steps of the experiment are consistent with Subsection 4.1, which we will not repeat here. The attack results are presented in Figure 4.

It is apparently seen that using the augmented training set to build the model can achieve better attack performance, where the number of traces required for GE to converge to 0 is reduced by about 400. The results reveal that the generated traces are capable of providing additional useful information, which is equivalent to the circumstance where we collect more traces. Of course, the generated traces would make less contribution due to leakage noise in most cases.

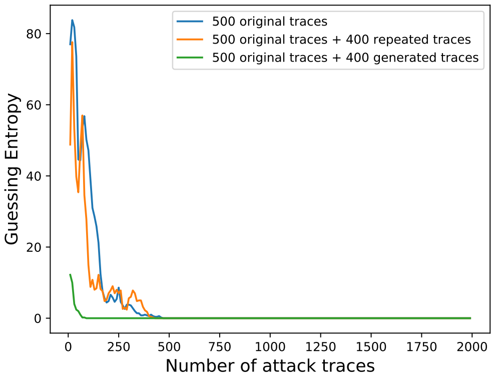

Figure 4: The attack results of models built with {500 original traces, 500 original traces + 400 repeated traces, 500 original traces + 400 generated traces}.

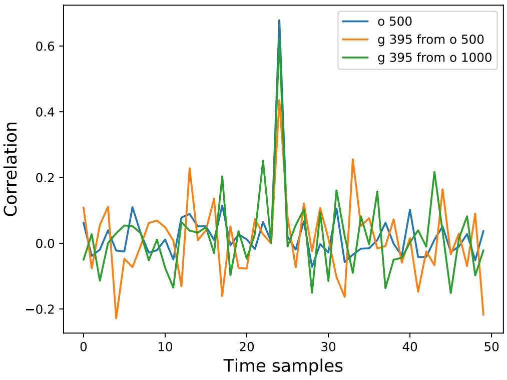

Figure 5: CPA of {500 original traces, 395 generated traces from 500 original traces, 395 generated traces from 1000 original traces} when using HW as label.

3.3 Labels

We continue to discuss how to select the label in the process of generating traces. In Subsection 3.2 we have been using the label LSB, next we will apply the same experimental configuration, except that the label is replaced with HW. Note that for the label HW, there are 9 categories in total and they satisfy the binomial distribution. Thus, we generate 395 traces according to the distribution, among which the amount of each category conforms to the binomial distribution. Furthermore, for better comparison, based on the original training set with 500 traces, we add another 500 traces and then use the 1000 traces to generate traces. Figure 5 shows the comparison of CPA results.

From Figure 5 we can see that if only 500 traces are used to generate traces, the correlation coefficient at the leakage points calculated by the generated traces is reduced by about 0.2, and the noise of other non-leakage points is more obvious as well. Then we add three sets of generated traces to the original training set of 500 traces to construct the models. The attack results are presented in Figure 6.

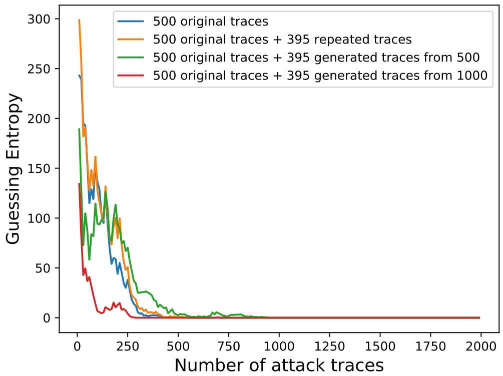

Figure 6: The attack results of models built with {500 original traces, 500 original traces + 395 repeated traces, 500 original traces + 395 generated traces from 500 original traces, 500 original traces + 395 generated traces from 1000 original traces} when using HW as label.

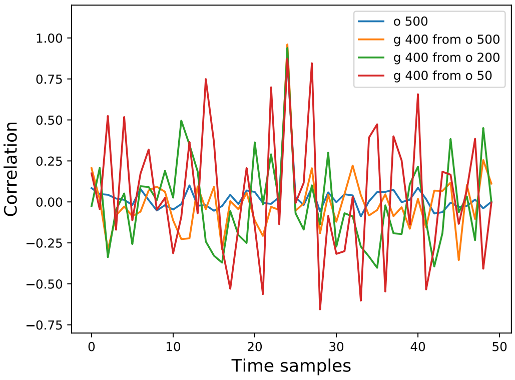

Figure 7: CPA of {500 original traces, 400 generated traces from 500 original traces, 400 generated traces from 200 original traces, 400 generated traces from 50 original traces}.

We find that the generated traces by using the 500 original traces do not bring about better attack performance, and even cause the attack result to become worse. We note that when the label HW is used, there exists a problem called imbalanced data [PHJ$^+$19]. Similarly, when using the label HW for trace generation, we may also be confronted with this problem. Imagine that using a training set with 500 traces to generate traces, the amount of the traces with HW = 0, 8 maybe only 1 or 2, which makes it difficult for us to learn these two categories of traces. Therefore, the addition of these two categories of corresponding generation traces will not contribute to the model but aggravate the problem of data imbalance. While using 1000 traces for trace expansion produces slight enhancement, which indicates that using the label HW to generate traces is relatively difficult in the situation where the amount of existing original traces is small.

3.4 Sizes of Training Set and Generated Trace Set

In this section, we use training sets of different sizes to generate traces and study how the amount of traces in the original training set affects the quality of generated traces. We respectively use the original training sets of size 500, 200, and 50 to generate 400 traces with label LSB. Figure 7 demonstrates the CPA results of generated traces using distinct training sets.

The CPA results show that though the correlation at the leakage points is similar, the noise at the non-leakage points increases as the size of the original training set is reduced. Especially, when only 50 traces are used for trace generation, the noise is strongly obvious, which tells us that if the original training traces are quite fewer, the generated traces would possess poor quality. Then, we add the generated traces above to the same original training set with 500 traces for modeling and observe whether the attack performance has been improved. The attack results are shown in Figure 9.

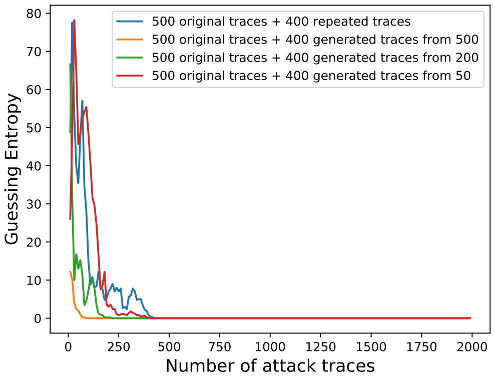

Figure 8: The attack results of models built with {500 original traces + 400 repeated traces, 500 original traces + 400 generated traces from 500 original traces, 500 original traces + 400 generated traces from 200 original traces, 500 original traces + 400 generated traces from 50 original traces}.

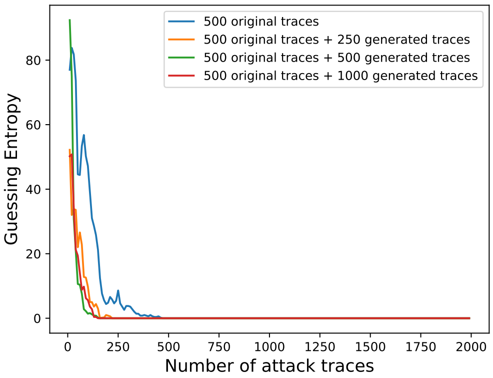

Figure 9: The attack results of models built with {500 original traces + 250 generated traces, 500 original traces + 500 generated traces, 500 original traces + 1000 generated traces}.

It implies that as the size of the original training set is decreasing, the improvement of the attack performance brought by the generated traces is constantly weakening. For example, when we only use 50 traces for trace generation and expansion, it is almost impossible to produce any amelioration. In the light of the above results and analysis, for effective generated traces, not only high correlation at the leakage points is required, but also low correlation at the non-leakage points is needed. In other words, we want the CPA results of the generated traces and the original traces to be as close as possible.

In addition to observing the effect of the size of original training set on the quality of the generated traces, we also have studied how the ratio between the size of added generated trace set and the size of original training set influence the attack performance. On the basis of previous experimental configuration, we use the 500 original traces to respectively generate 250, 500, and 1000 traces as three different generated trace sets, then add them to the 500 original traces in turn for modeling and compare the influence on attacks. The results are displayed in the following Figure **??**.

The comparison of the attack results indicates that the ratio between the size of added trace set and the size of original training set does not actually affect the attack performance greatly. Hence, we are not going to discuss this issue in detail subsequently when we conduct experiments with real traces. Undoubtedly, if adding too few traces, it is hard to make any improvements. On the other hand, adding too many generated traces would make the model learn more from the generated traces, which will also reduce the attack performance.

4 Experiments

In this section, we first discuss the application of using CGAN to generate traces under three AES implementations: unprotected, first-order masked protected and random delay protected. Then we test the universality of the proposed method to different modeling algorithms. Recently, profiling attacks based on DL are becoming very popular, thus we mainly test common modeling algorithms used in DL.

4.1 Experiment Setup

In Subsection 4.2, Subsection 4.3 and Subsection 4.4, we will use the public model MLP_{best} in [PSB+18]. The reason why we use it is that our goal is to verify whether adding the generated traces to the profiling set can enhance the attack performance, rather than to find out the optimal model. Hence, we do not change the structure of the model, which has four hidden layers with 200 neurons in each layer. For different implementations of AES, we just modify the learning rate, epoch and batch size. Besides, we think the experimental results are more convincing by using public models.

Specifically, we first divide the data set into two parts D_1 and D_2. Then we randomly divide out the training set $D_{training}$ and the validation set $D_{validation}$ from D_1 in each experiment [8], and divide out the testing set $D_{testing}$ from D_2. Because of the overall structure of the model is fixed, we only need to fine-tune the learning rate, epoch, and batch size through $D_{validation}$ to get the best model $M_{original}$ [9] under the original training set $D_{training}$. Secondly, we use CGAN to generate trace set $D_{generated}$ through $D_{training}$ and combine $D_{generated}$ and $D_{training}$ as the new training set $D_{new\_training}$. We then use $D_{validation}$ to fine-tune the above parameters to get the best model M_{new} under $D_{new\_training}$. Finally, we attack testing set $D_{testing}$ by using $M_{original}$ and M_{new} respectively and get the score ranking of the correct key. We repeat the above experiment 10 times, with the average attack results as the final GE. Moreover, it must be emphasized that in order to make the final results more credible the number of traces in D_1 is greater than the sum of the number of traces in $D_{training}$ and $D_{validation}$, and the number of traces in D_2 is also greater than the number of traces in $D_{testing}$ in each experiment.

To better verify the effectiveness of the traces generated by CGAN, we added three sets of comparative experiments for each data set. (1) add the repeated traces to the training set, (2) use the method of adding noise to the traces to obtain new traces and add them to the training set [KPH+19], (3) add real traces to the training set. Another point that needs to be added is the comparison of the shape of the generated trace and the original trace, which we have placed in Appendix A.3.

4.2 Unprotected AES

In this section, we use two trace sets collected in two implementations of unprotected AES, one of which is DPAcontest v4 dataset, and the other is power trace set in unprotected AES-128 implementation running on an Atmel XMEGA128 chip, measured by ChipWhisperer-Lite (CW) platform [CW]. In the following two subsections we verify that using generated traces can improve the effectiveness of profiling attacks with insufficient profiling traces. At the same time, we have found an interesting phenomenon that we can successfully perform profiling attacks only using generated traces.

4.2.1 DPAcontest v4

DPAcontest v4 dataset provides measurements in software implementation of AES-256 with Rotating Sbox Masking (RSM) protection [NSGD12]. We can directly compute the masked S-box output through the known masks, thus turning it into an unprotected AES implementation. Here we only use 10,000 traces in the DPAcontest v4 dataset. We let the first 6,000 traces as D_1 and the remaining 4,000 traces as D_2. Then we randomly divide out 1,000 traces from D_1 as the training set and 2,000 traces as the validation set. We randomly choose 2,000 traces from D_2 as the testing set. We perform a CPA to intercept

[8] $D_{training}$ and $D_{validation}$ are not intersected.

[9] In fact, the learning rate, epoch and batch size corresponding to $M_{original}$ in each experiment will be basically unchanged. We do this because we consider that the size of the original training set would change after adding the generated traces, and the optimal value of the three parameters might also change, then we add a cross-validation process.

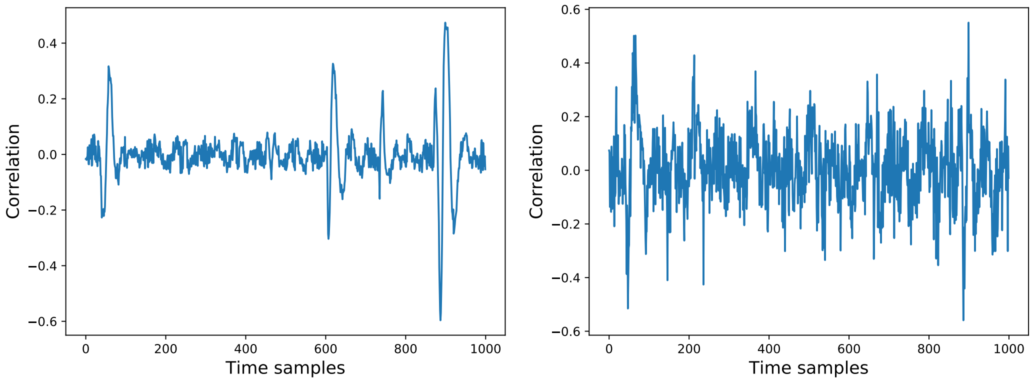

Figure 10: Left: CPA of 1000 original traces. Right: CPA of 1000 generated traces.

Table 1: The attack results on DPAcontest v4.

Training set	**Guessing entropy**
1000 original traces	550
1000 original traces + 1000 repeated traces	561
1000 original traces + 1000 noisy traces	504
1000 original traces + 1000 generated traces	207
2000 original traces	172

1,000 consecutive time samples related to the masked S-box output in the first round. We use the LSB of the masked S-box output as the training label, and then use 1,000 traces of the training set to generate 1,000 new traces.

Then we use CPA to determine whether the generated traces have learned the leakages. We perform CPA with 1,000 traces of the original training set and 1,000 traces of the generated trace set respectively, the results are presented in Figure 10.

We can see that there are four main leakage points in the CPA results of original traces and the generated traces mainly learn the leakages of the far left and far right points, while the other two leakage points in the middle are not learned obviously. At the same time, the leakage noise at the non-leakage points is very high, thus learning the leakages of real traces is more difficult than the simulated traces. We then verify the effectiveness of the generated traces through the attack results in Table 1.

The attack results have improved significantly after using the generated traces to augment the original training set in Table 1 and the number of traces required for GE to converge to 0 is about 300 fewer. The results confirm that the traces we generate can contribute to the model like the real traces.

4.2.2 CW

We use CW to collect 8000 traces to further prove the effectiveness of the generated traces. We choose the first 4000 traces as D_1, and the remaining 4000 traces as D_2. Then we randomly divide out 100 traces from D_1 as the training set, 2000 traces as the validation set each time, and randomly choose 2000 traces from D_2 as the testing set. The target of the attack is the S-box output in the first round. We intercept 500 consecutive time samples related to the first S-box output leakage points from the total 3,000 time samples in the original traces according to CPA. To simulate the profiling phase where few traces are available, we only use 100 traces as the training set. Next, we use the 100 original traces to generate 400 traces and give the comparison of CPA results in Figure 11.

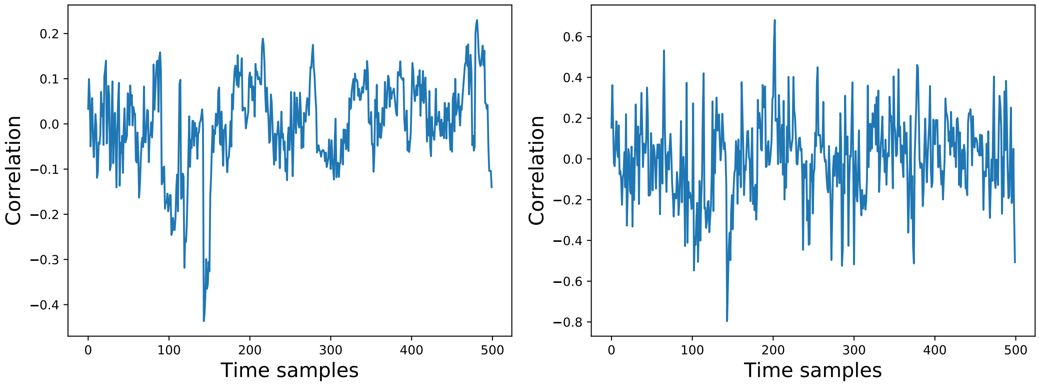

Figure 11: Left: CPA of 100 original traces. Right: CPA of 400 generated traces.

Table 2: The attack results on CW unprotected implementation.

Training set	**Guessing entropy**
100 original traces	> 2000
100 original traces + 400 repeated traces	> 2000
100 original traces + 400 noisy traces	1859
100 original traces + 400 generated traces	853
500 original traces	548

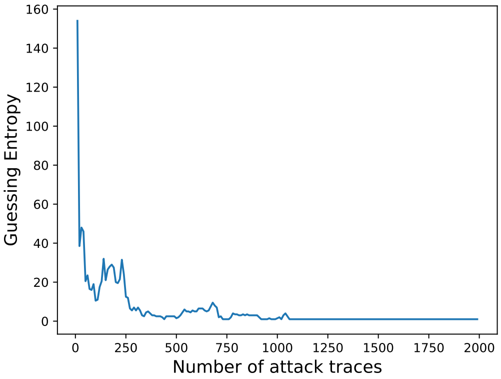

Figure 12: The attack results of model built with 400 generated traces only.

Although the leakage noise of the generated trace is very high, the main leakage at the time samples of 100 to 200 is obvious, and we find that the overall shape of the curves in the two CPA results is similar. Then we give the attack results in Table 2.

From the attack results, GE can not converge to 0 by using only 100 traces as the training set when using 2000 attack traces. When we use the augmented training set to model, GE can converge to 0 by using 853 traces. Therefore, the improvement of the attack results again proves the validity of the traces generated by CGAN.

In order to further verify that the generated traces contain useful information for modeling, we try to only use the generated traces to build a model and conduct a successful attack. Here we only use 400 generated traces from the previous 100 traces to build a model and the attack results are shown in Figure 12.

We can clearly see that using only 400 generated traces can attack successfully and yield a better attack performance than the original 100 traces. Therefore, the traces generated

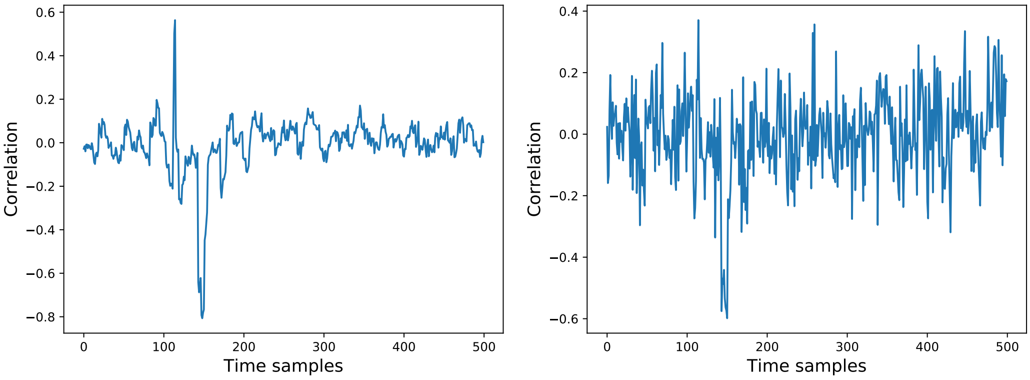

Figure 13: Left: CPA of 400 original traces. Right: CPA of 395 generated traces.

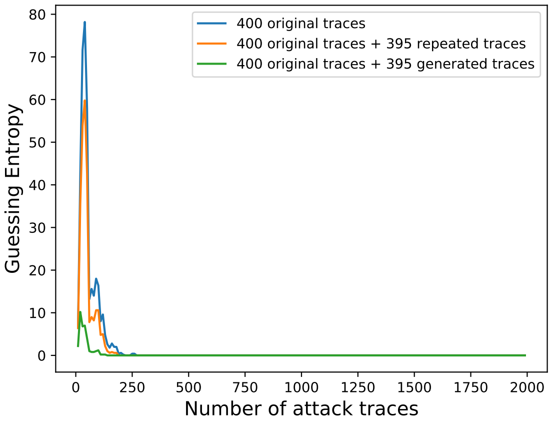

Figure 14: The attack results of models built with {400 original traces, 400 original traces + 395 repeated traces, 400 original traces + 395 generated traces} when using HW as label.

by CGAN can contribute to modeling just as real traces do. Besides, we can also evaluate the effectiveness of the generated traces by only using generated traces to build a model.

In Subsection 3.3, we have mentioned that using HW as a label to generate traces is difficult, especially when sufficient traces cannot be collected in the profiling phase. Here, we will test the effectiveness of generating traces using HW label in real experiments. We randomly select 400 traces from D_1 as the training set, 2000 traces as the validation set, and randomly choose 2000 traces from D_2 as the testing set. It need to be emphasized that minority class may not appear when only 100 traces are used to generate. Then we use 400 traces in the training set to generate 395 traces which still obey the binomial distribution. First, we perform CPA on the training set and the generated traces respectively. The CPA results are presented in Figure 13.

The correlation of the generated traces at the main leakage points has decreased, which was also mentioned in previous simulation experiments. Next, we respectively use the original training set and the training set augmented with the generated traces to model. The attack results are presented in Figure 14.

It can be seen that in real experiments, it is feasible to use the HW label to generate traces, but the improvement of the attack performance is not obvious. This also reflects

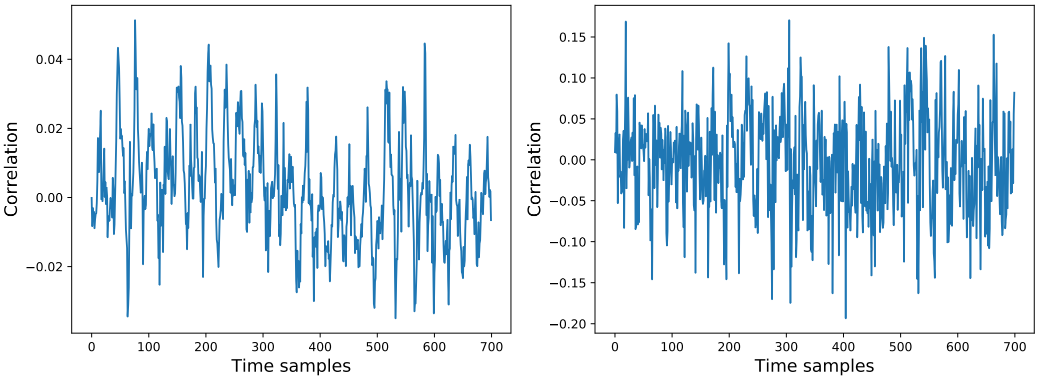

Figure 15: Left: CPA of 100 original traces. Right: CPA of 395 generated traces.

Table 3: The attack results on ASCAD.h5.

Training set	**Guessing entropy**
2000 original traces	878
2000 original traces + 2000 repeated traces	852
2000 original traces + 2000 noisy traces	753
2000 original traces + 2000 generated traces	182
4000 original traces	91

that it is not appropriate to use HW as a label to generate traces when profiling traces are insufficient.

4.3 First-Order Masked AES Implementation

In this section, we try to expand the application scope of using CGAN to generate traces and test whether it can still improve the performance of profiling attacks on first-order masked AES. Here we use dataset ASCAD.h5 in the public database ASCAD. One purpose of the paper [PSB+18] is to investigate whether a DL model can break masking scheme protection. Therefore, it does not calculate the output of the masked S-box, but directly takes the output of the third S-box as the label. In fact, there is no first-order leakage. Hence, when we generate traces using the label directly related to the S-box output, we can not see obvious peaks in the CPA results of generated traces. Although it is not possible to judge whether the generated traces have learned leakages through CPA, we can use the attack results to verify the effectiveness of the generated traces.

We let the training set of 50000 traces be D_1, and the testing set of 10000 traces be D_2. In order to simulate a scenario where sufficient traces cannot be collected in practical profiling phase, we randomly select 2000 traces from D_1 as the training set, 2000 traces as the validation set, and randomly select 2000 traces from D_2 as the testing set. The label is $LSB(Sbox(p[3] \oplus r[3]))$. Then we use the training set to generate 2000 traces.

First, we still compare the CPA results of the original training set and generated traces. The results are presented in Figure 15. As mentioned above, the S-box output of the first-order masked AES has no first-order leakage, and no leakages can be seen in the CPA results of the original training set or the generated traces. Then we judge the effectiveness of the generated traces by whether it can improve the attack performance of the model. The comparison of attack results is shown in Table 3.

Compared with the original training set, the training set expanded with the generated traces can build a better model, and the traces required for GE to converge to 0 is reduced by almost four times. In order to further prove that CGAN can learn higher-order leakage,

Table 4: Based on the known masks, we construct four labels for the training set of CGAN.

$LSB(mask)$	$LSB(masked)$	Gan_label	$LSB(Sbox\_output)$
0	0	0	0
1	0	1	1
0	1	2	1
1	1	3	0

we conduct a more detailed trace generation using known masks. As shown in the following table, we divide the generated traces into four categories in Table 4.

Four labels in the Table 4 have the following relationships:

$$Gan\_label = 2 \cdot LSB(masked) + LSB(mask). \tag{6}$$

$$LSB(Sbox\_output) = LSB(mask) \oplus LSB(masked). \tag{7}$$

Based on Eq. (6), we can get $LSB(mask)$ and $LSB(masked)$ from Gan_label[10], to perform a CPA on the generated traces targeting the mask and the masked S-box output. On the other hand, we can calculate the $LSB(Sbox\_output)$ from $LSB(mask)$ and $LSB(masked)$ through Eq. (7), as the label of generated traces when expanding the original training set. First, we randomly divide out 2000 traces from D_1 as a training set. Then we generate 2,000 traces using the original training set, with 500 traces in each class. Finally, the results of CPA targeting mask and masked S-box output which is performed on the original training set and generated traces respectively are displayed in Figure 16:

From above CPA results, the generated traces can learn major leakage related to both mask and masked S-box output, which indicates that CGAN can learn high-order leakages. At the same time, this experiment is also a supplement to previous trace generation only using LSB of the S-box output. It confirms that the generated traces in the previous experiment have learned effective leakage. Therefore, even if the mask cannot be obtained in a real situation, we can directly use the label related to the output of the S-box to effectively generate traces.

4.4 Desynchronized Traces

In this section, we continue to study the application of using CGAN to generate traces on the desynchronized traces. We first perform a simulation experiment to verify whether CGAN can generate effective desynchronized traces. The configuration of the simulation experiment is the same as that in section 3.2, except that we add a random jitter $s \sim (-2, 2)$ to the original leakage point. The location of the leakage point has changed to $t_{leakage} = 25 + s$, thus we obtain a desynchronized dataset.

In each repeated experiment, the 500 traces of the original training set are still used to generate 400 traces. The CPA results of the generated traces and the original traces are shown in Figure 17.

As a result of desynchronization, the correlation peak at the leakage point of the original training set not only drops a lot but also becomes wider. While in the CPA results of generated traces, we find that the correlation at the leakage point is twice as high, but the leakage noise is also more obvious, which indicates that it is more difficult to generate desynchronized traces. Next, we give a comparison of the attack results in Figure 18.

The attack performance of the trained model based on desynchronized traces is much worse than the synchronized traces in section 3.2. Nevertheless, using the generated traces to augment the original trace set can still effectively improve the attack performance.

Next, we will continue to verify the effectiveness of CGAN on desynchronized traces by using AES_RD dataset which is collected in the AES implementation with random

[10] Gan_label refers to the label used during CGAN training.

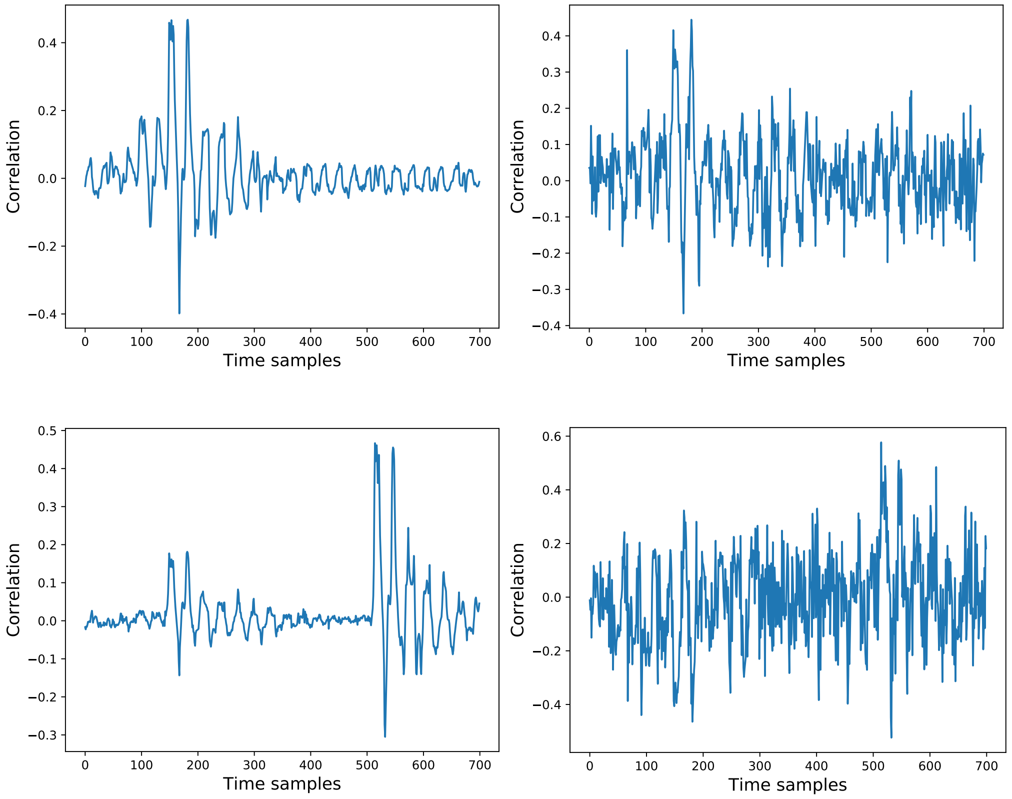

Figure 16: Top-left: mask CPA of original traces. Top-right: mask CPA of generated traces. Bottom-left: masked CPA of original traces. Bottom-right: masked CPA of generated traces.

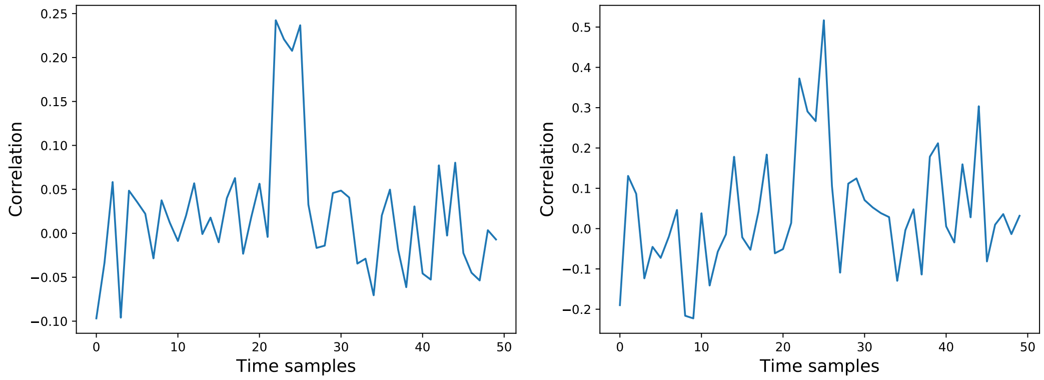

Figure 17: Left: CPA of 500 original traces. Right: CPA of 400 generated traces.

delay countermeasure [CK09]. The original AES_RD dataset contains 50,000 traces. We choose the first 30,000 traces as D_1, the last 20,000 traces as D_2. Then we randomly divide out 10,000 traces from D_1 as the training set and 10,000 traces as the validation set. We randomly divide out 10,000 traces from D_2 as the testing set. The label we use is

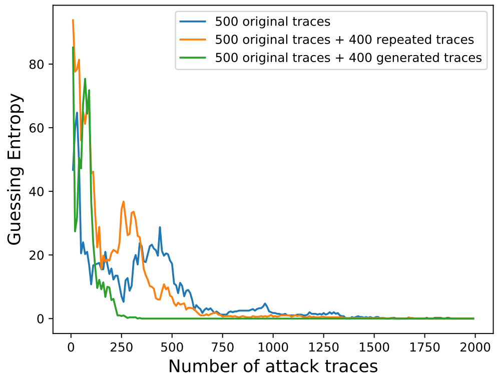

Figure 18: The attack results of models built with {500 original traces, 500 original traces + 400 repeated traces, 500 original traces + 400 generated traces} on the simulated desynchronization traces.

Table 5: The attack results on AES_RD.

Training set	Guessing entropy
10000 original traces	4125
10000 original traces + 10000 repeated traces	3998
10000 original traces + 10000 noisy traces	3365
10000 original traces + 10000 generated traces	1369
20000 original traces	890

also LSB. Finally, we use the training set to generate 10,000 traces. The attack results are presented in Table 5.

The number of traces required for a successful attack has been reduced by about 2700 when we use the original training set augmented with generated traces to build a model. And it also shows that CGAN can generate effective traces even when the traces are not synchronized.

4.5 Application to Other Modeling Algorithms

In previous experiments, the model we have always used is MLP. In this section, we will verify the universality of our method to different modeling algorithms. We test three models, SVM, RF, and CNN. The first two are machine learning models, and the last one is a DL model which is often applied to profiling attacks in recent years.

Here we will test the datasets used in sections 4.2.2 and 4.3 and the experimental configuration is the same as before, except that the modeling algorithm used is different. The specific parameters used in the model are given in Appendix A.4. Then, under three other modeling algorithms, we compare the attack results before and after using the generated traces to expand the original training set.

From the attack results in Table 6, we can see that even if different modeling algorithms are used, in scenarios where the profiling traces are insufficient, the traces generated by CGAN can improve the attack performance. This also confirms the universality of our method to different modeling algorithms.

Table 6: The attack results of model SVM, RF, CNN on dataset CW, ASCAD.

Dataset	Training set	SVM	RF	CNN
CW	100 original traces	> 2000	250	> 2000
	100 original traces + 400 repeated traces	1305	259	> 2000
	100 original traces + 400 noisy traces	1591	204	1760
	100 original traces + 400 generated traces	316	76	171
	500 original traces	194	53	86
ASCAD	2000 original traces	1502	625	850
	2000 original traces + 2000 repeated traces	1493	874	702
	2000 original traces + 2000 noisy traces	941	533	597
	2000 original traces + 2000 generated traces	313	380	294
	4000 original traces	244	257	232

5 Discussions

1. **Compared with the previous profiling attack work, our attack effect improvement seems to be smaller.** We want to emphasize that our work is in a scene with insufficient profiling traces, so it is very difficult to improve the attack effect. At the same time, when we compare some other data augmentation techniques, such as adding Gaussian noise, the improvement effect is very weak, which also shows that the generation of the adversarial network is powerful and can produce traces that bring additional useful information. Of course, we are manually simulating the lack of traces scenarios to confirm the effectiveness of CGAN and provide a method to solve the problem of how to better improve profiling attacks in this scenario.

2. **In the scenario of insufficient profiling traces, the attack effect of the constructed model is not good, does it mean that the model is not optimal?** One thing to note is that the optimal model we use here is relatively in a scene with sufficient trace, just like the public model we used in this paper. From the perspective of information theory, in a scenario where the profiling traces are lacking, the current model cannot extract more information, and a better model may exist. Our work is not to directly optimize this model, but to generate new traces so that existing profiling methods can get better profiling.

6 Conclusion

In this paper, we first introduce CGAN(GAN) in the context of side-channel attacks. When sufficient traces cannot be collected during the profiling phase, CGAN can generate traces for data augmentation, which effectively improves the performance of profiling attacks. In the process of using CGAN to generate traces, we summarize some methods and techniques: By CPA or DPA, we can determine in advance whether the generated traces have learned effective leakages. It is easy to use the single-bit label to generate effective traces. We must add a suitable amount of generated traces when augmenting the original profiling set. We find that CGAN has the ability to learn not only first-order leakage, but also high-order leakage, and it can also effectively generate desynchronized traces. Besides, we also test different modeling algorithms, including SVM, RF, MLP, and CNN, which demonstrates the universality of our method to different modeling algorithms. In future work, we plan to generate traces collected from other types of cryptographic implementations by CGAN and improve the quality of generated traces.

References

[ASE17] Antreas Antoniou, Amos J. Storkey, and Harrison Edwards. Data augmentation generative adversarial networks. *CoRR*, abs/1711.04340, 2017.

[BCG+18] Christopher Bowles, Liang Chen, Ricardo Guerrero, Paul Bentley, Roger N. Gunn, Alexander Hammers, David Alexander Dickie, Maria del C. Valdés Hernández, Joanna M. Wardlaw, and Daniel Rueckert. GAN augmentation: Augmenting training data using generative adversarial networks. *CoRR*, abs/1810.10863, 2018.

[BCO04] Eric Brier, Christophe Clavier, and Francis Olivier. 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*, pages 16–29, 2004.

[BL12] Timo Bartkewitz and Kerstin Lemke-Rust. Efficient template attacks based on probabilistic multi-class support vector machines. In *Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers*, pages 263–276, 2012.

[Bre01] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures - profiling attacks without pre-processing. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, pages 45–68, 2017.

[CK09] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In *Cryptographic Hardware and Embedded Systems - CHES 2009, 11th International Workshop, Lausanne, Switzerland, September 6-9, 2009, Proceedings*, pages 156–170, 2009.

[CRR02] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, pages 13–28, 2002.

[CV95] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20(3):273–297, 1995.

[CW] ChipWhisperer Website. https://newae.com/tools/chipwhisperer/.

[FKA+18] Maayan Frid-Adar, Eyal Klang, Michal Amitai, Jacob Goldberger, and Hayit Greenspan. Synthetic data augmentation using GAN for improved liver lesion classification. In *15th IEEE International Symposium on Biomedical Imaging, ISBI 2018, Washington, DC, USA, April 4-7, 2018*, pages 289–293, 2018.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, pages 426–442, 2008.

[GPM$^+$14] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C. Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 2672–2680, 2014.

[HGM$^+$11] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptographic Engineering*, 1(4):293–302, 2011.

[HRU$^+$17] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6626–6637, 2017.

[IZZE17] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks. In *2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017*, pages 5967–5976, 2017.

[KB15] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *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 *Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings*, pages 388–397, 1999.

[Koc96] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 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 Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(3):148–179, 2019.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States*, pages 1106–1114, 2012.

[LBM14] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Power analysis attack: an approach based on machine learning. *IJACT*, 3(2):97–115, 2014.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A machine learning approach against a masked AES. In *Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, pages 61–75, 2013.

[MHN13] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[MO14] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets. *CoRR*, abs/1411.1784, 2014.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *Security, Privacy, and Applied Cryptography Engineering - 6th International Conference, SPACE 2016, Hyderabad, India, December 14-18, 2016, Proceedings*, pages 3–26, 2016.

[NSGD12] Maxime Nassar, Youssef Souissi, Sylvain Guilley, and Jean-Luc Danger. RSM: A small and fast countermeasure for aes, secure against 1st and 2nd-order zero-offset scas. In *2012 Design, Automation & Test in Europe Conference & Exhibition, DATE 2012, Dresden, Germany, March 12-16, 2012*, pages 1173–1178, 2012.

[PHG19] Stjepan Picek, Annelie Heuser, and Sylvain Guilley. Profiling side-channel analysis in the restricted attacker framework. *IACR Cryptology ePrint Archive*, 2019:168, 2019.

[PHJ+19] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019(1):209–237, 2019.

[PSB+18] Emmanuel Prouff, Rémi Strullu, Ryad Benadjila, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. *IACR Cryptology ePrint Archive*, 2018:53, 2018.

[PYW+17] Sihang Pu, Yu Yu, Weijia Wang, Zheng Guo, Junrong Liu, Dawu Gu, Lingyun Wang, and Jie Gan. Trace augmentation: What can be done even before preprocessing in a profiled sca? In *Smart Card Research and Advanced Applications - 16th International Conference, CARDIS 2017, Lugano, Switzerland, November 13-15, 2017, Revised Selected Papers*, pages 232–247, 2017.

[RMC16] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. In *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[SGZ+16] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2226–2234, 2016.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop, Edinburgh, UK, August 29 - September 1, 2005, Proceedings*, pages 30–46, 2005.

[SMY09] François-Xavier Standaert, Tal Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, pages 443–461, 2009.

[SSP03] Patrice Y. Simard, David Steinkraus, and John C. Platt. Best practices for convolutional neural networks applied to visual document analysis. In *7th International Conference on Document Analysis and Recognition (ICDAR 2003), 2-Volume Set, 3-6 August 2003, Edinburgh, Scotland, UK*, pages 958–962, 2003.

[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.

[Whi16] Tom White. Sampling generative networks: Notes on a few effective techniques. *CoRR*, abs/1609.04468, 2016.

[WW98] Jason Weston and Chris Watkins. Multi-class support vector machines. Technical report, Citeseer, 1998.

A

A.1 CGAN

	Generator	**Discriminator**
Input layer	(noise with $laten\_dim = 100$,embedding(label))	(traces, embedding(label))
Hidden layer	(Dense,LeakyReLU,BN) * N	(Dense, LeakyReLU) * N
Output layer	Dense with tanh	Dense with sigmoid

A.2 DPA Analysis

From top to bottom, Figure 19, Figure 20 and Figure 21 correspond to the traces used for CPA analysis in Subsection 3.2, Subsubsection 4.2.1 and Subsubsection 4.2.2 respectively.

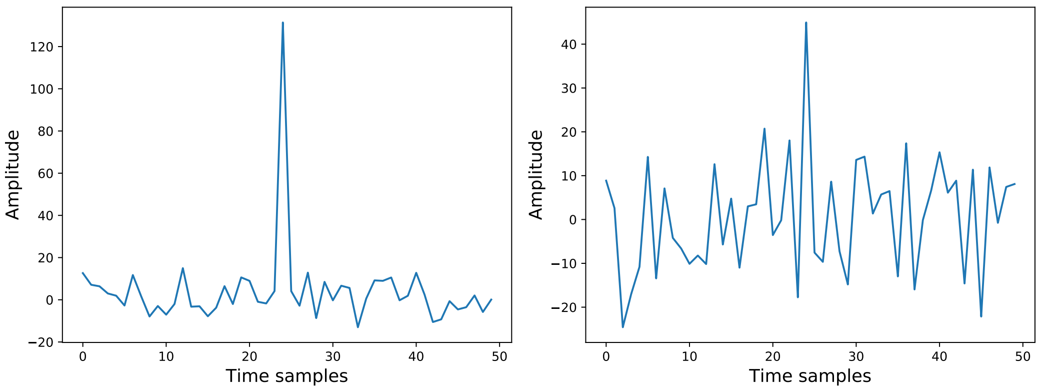

Figure 19: Left: DPA of 500 original traces. Right: DPA of 400 generated traces.

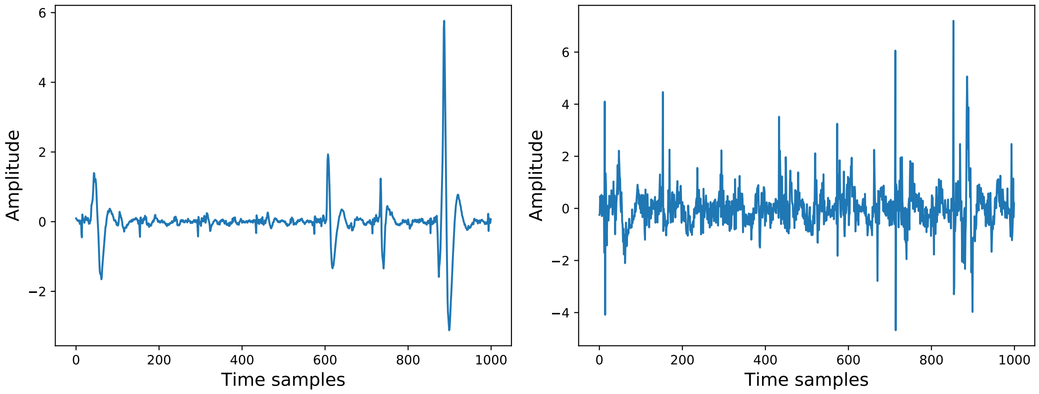

Figure 20: Left: DPA of 1000 original traces. Right: DPA of 1000 generated traces.

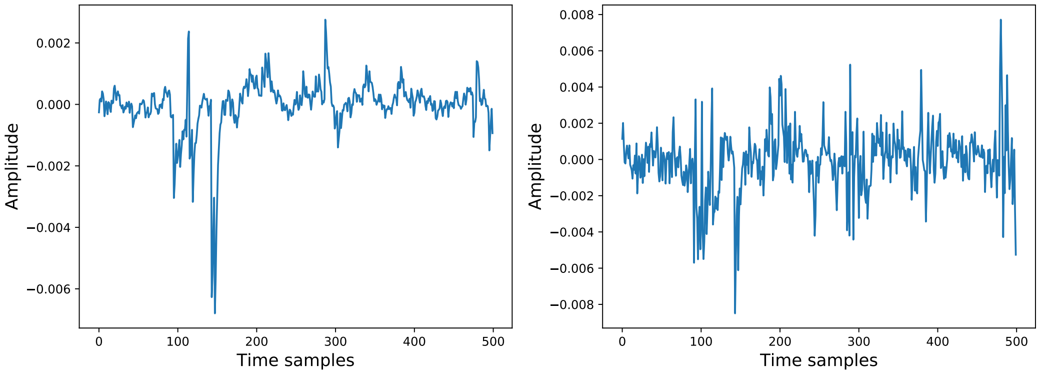

Figure 21: Left: DPA of 100 original traces. Right: DPA of 400 generated traces.

A.3 The Shape of Generated Traces

The shape of the generated trace is similar to the original one. The difference is that there are slight amplitude jitters at the peaks of the generated trace, and these slight jitters contain useful information we need for modeling. The comparison results are shown in Figure 22, Figure 23 and Figure 24.

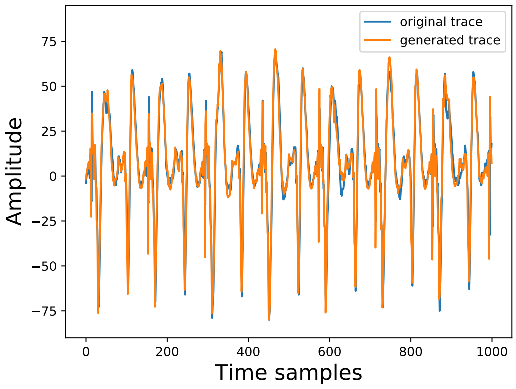

Figure 22: The shape of original trace and generated trace (DPAv4).

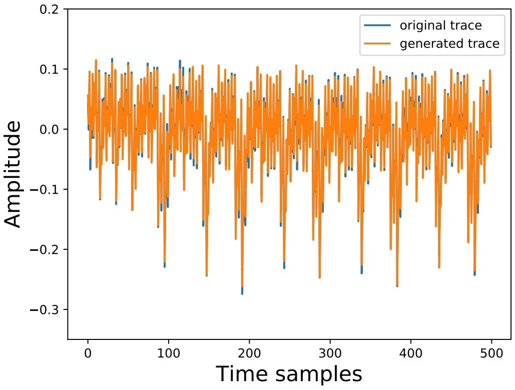

Figure 23: The shape of original trace and generated trace (CW).

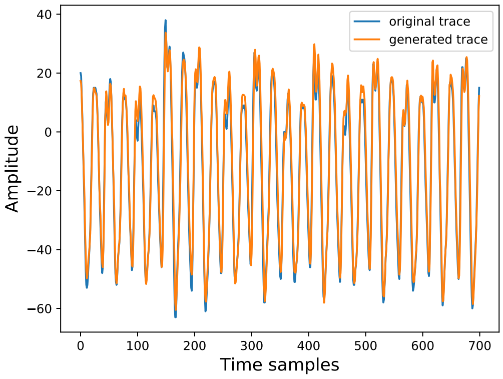

Figure 24: The shape of original trace and generated trace (ASCAD).

A.4 Classifiers

	CW	**ASCAD**
SVM	Margin: $C = 100$ kernel parameter: $\gamma = 0.001$	Margin: $C = 500$ kernel parameter: $\gamma = 0.001$
RF	The number of trees: $n\_estimators = 100$	The number of trees: $n\_estimators = 1000$

CNN(filters, size, stride, padding, activation)	(32, 11, 1, ‘same’, relu)
	(64, 11, 1, ‘same’, relu)
	Dense output with softmax
		(64, 11, 1, ‘same’, relu)
		AveragePooling(size=2, stride=2)
		(128, 11, 1, ‘same’, relu)
		AveragePooling(size=2, stride=2)
		(256, 11, 1, ‘same’, relu)
		Dense(2048, relu)
		Dense(2048, relu)
		Dense output with softmax