It's a Kind of Magic: A Novel Conditional GAN Framework for Efficient Profiling Side-channel Analysis

Sengim Karayalcin[1], Marina Krcek[3], Lichao Wu[2], Stjepan Picek[2] and Guilherme Perin[1]

[1] Leiden University, The Netherlands
[2] Radboud University, The Netherlands
[3] Delft University of Technology, The Netherlands

Abstract. Profiling side-channel analysis is an essential technique to assess the security of protected cryptographic implementations by subjecting them to the worst-case security analysis. This approach assumes the presence of a highly capable adversary with knowledge of countermeasures and randomness employed by the target device. However, black-box profiling attacks are commonly employed when aiming to emulate real-world scenarios. These attacks leverage deep learning as a prominent alternative since deep neural networks can automatically select points of interest, eliminating the need for secret mask knowledge. Nevertheless, black-box profiling attacks often result in non-worst-case security evaluations, leading to suboptimal profiling models. In this study, we propose modifying the conventional black-box threat model by incorporating a new assumption: the adversary possesses a similar implementation that can be used as a white-box reference design. We create an adversarial dataset by extracting features or points of interest from this reference design. These features are then utilized for training a novel conditional generative adversarial network (CGAN) framework, enabling a generative model to extract features from high-order leakages in protected implementation without any assumptions about the masking scheme or secret masks. Our framework empowers attackers to perform efficient black-box profiling attack that achieves (and even surpasses) the performance of the worst-case security assessments.

Keywords: Side-channel Analysis · Deep learning · Generative AI

1 Introduction

Commonly deployed cryptographic algorithms are widely regarded as mathematically secure, as simply knowing the input and output data along with the details of the algorithm is insufficient to recover the key within a reasonable computation time. However, when cryptographic implementations run on electronic devices, unintentional and unavoidable information leakages occur, primarily through power consumption, electromagnetic emission, execution time, temperature, and acoustics. These side-channel information leaks can be exploited by attackers who measure and analyze them during encryption or decryption operations. This enables them to extract the key through statistical analysis of the side-channel measurements.

The research on the side-channel analysis (SCA) domain has mainly concentrated on two main approaches. On one side, there are statistical methods, distinguishers, and leakage assessment techniques for direct side-channel attacks. Examples of such attacks include Differential Power Analysis [KJJ99], Correlation Power Analysis [BCO04], and

Mutual Information Analysis [GBTP08]. On the other hand, research on profiling SCA (all of them derived from the classic template attack [CRR02]) has largely aimed at enabling worst-case security assessments [BDMS22]. In the first category, attack methods are implicitly treated as real-world security threats, mainly because a side-channel attack can be directly mounted on the victim's device, in which the access to the target, together with the ability to query encryption or decryption executions, are the only requirements to deploy the attack itself. Profiling SCA, although also realistic, involves more assumptions within a well-defined threat model. Its primary research objective is to assess the security of a cryptographic implementation when an adversary possesses an identical and open (i.e., programmable) copy of the victim's device. A conventional threat model for worst-case security assumes the attacker has sufficient information about the countermeasures, including the random values generated during cryptographic executions [PPM+22].

Profiling SCA follows two main phases. In the first (or profiling) phase, a set of side-channel measurements (with varying keys and varying input data), usually very large, is collected from the open device to *learn* a statistical function that can describe the probability distribution of measurements. In the second (or attack) phase, a separate set of side-channel measurements (with a fixed key and varying input data) is collected from the target device and *matched* on the computed statistical function. A worst-case security assessment is not always possible in security evaluations, especially following certification schemes. If the profiling phase has some limitations concerning access to secret random values (which are normally written in protected registers and not accessible from outside), the implementation of feature selection or dimensionality reduction becomes highly limited, and the profiling phase has to learn from large side-channel measurement intervals, with massive amounts of noisy data points. This means a security evaluation must follow non-worst case assumptions, which is assumed to be hard as building an optimal profiling model requires more computation efforts [PWP22].

Nevertheless, the target device is usually similar or shares several implementation details with other targets. For instance, if the security assessment is performed on a first-order masked AES implementation, the evaluator may have access to similar AES designs running on different devices from other manufacturers. Therefore, we propose the following scenario:

> *How much would the security of a cryptographic implementation be compromised when an adversary possesses an open and fully-controlled similar implementation?*

This work introduces another ingredient to the classic (non-worst case) profiling attack threat model. Specifically, we assume that the adversary possesses access to a comparable (i.e., different) and openly available implementation where the randomness elements (such as masks) are known and the implementation of feature selection is straightforward. We implement a novel conditional generative adversarial network (CGAN)-based structure that *learns to efficiently extract features from a target implementation with unknown masks* by learning the same process from a reference implementation with known masks. In the CGAN training, the adversarial dataset is given by the features selected from the reference dataset. The only requirement for our framework is that both implementations are similar (e.g., two software AES implementations with first-order Boolean masking countermeasures). Our results indicate that our CGAN-based framework can convert a black-box profiling attack into a profiling attack with similar (and eventually better) attack performance than white-box profiling attacks.

The proposed framework incorporates side-channel measurements from two distinct devices or measurement setups, but it does not specifically address portability issues [BCH+20], which are typically addressed through the utilization of transfer learning techniques in deep learning-based solutions [CZLG21, CZG+22]. Instead, it focuses on efficient feature

extraction from a target device with unknown masks using a black-box profiling approach, leveraging measurements from an open and different device with known masks. This approach improves the interpretability of the attack on the target dataset compared to conventional black-box deep learning-based profiling attacks.

In summary, our main contributions are:

- We propose a novel conditional generative adversarial network-based SCA framework that allows an adversary to leverage the knowledge from a reference implementation to extract features from a target implementation. This modified threat model and corresponding CGAN-based framework demonstrate the potential risks that arise when an adversary has full control over an implementation similar to the target one.
- The proposed framework allows an adversary to convert a black-box profiling attack towards a white-box profiling attack capability, which drastically improves the black-box profiling attack performance. Our results demonstrate that applying our framework reduces the difficulties of finding an optimal profiling model in a non-worst-case security evaluation significantly (we refer to the Section 6 of [MCLS23] for a discussion about difficulties in finding deep learning-based profiling models in worst and non-worst case security evaluations). For some of the tested scenarios, after applying our proposed CGAN-SCA framework, the chances to find a successful profiling attack increase from 8.92% to 90.47% and from 12.14% to 99.55% in a hyperparameter search process. We skip any feature selection process for datasets considered target devices, commonly done in the related deep learning-based SCA (DLSCA) literature [PPM+22].
- The proposed CGAN-SCA framework can precisely extract features from high-order leakages, such as first-order masking schemes. We provide a detailed analysis to demonstrate how the generator in a CGAN architecture precisely mimics the features selected from a reference implementation.

2 Background

2.1 Datasets

To demonstrate the performance of our proposed framework, we consider five publicly available datasets, all containing side-channel measurements from first-order masked AES software implementations. Four of these datasets, namely ASCADr, ASCADf, DPAv4.2, and CHES CTF 2018 are the same as adopted for the NOPOI scenario in [PWP22] (see Section 2.3 for specific details and Table 2 for information about the selected intervals from raw measurements of DPAv4.2 and CHES CTF 2018). The raw side-channel measurements from ASCADr, ASCADf, DPAv4.2, and CHES CTF 2018 contain large traces with 100 000, 250 000, 150 000, and 150 000 sample points per trace, respectively. Working with such large intervals is computationally intensive, and in this paper, we also consider window resampling with a window of 20 and step of 10. The resampled datasets result in preprocessed side-channel measurements with 25 000, 10 000, 15 000, and 15 000 samples per trace, and we consider 200 000, 50 000, 70 000 and 30 000 measurements as profiling sets for ASCADr, ASCADf, DPAv4.2, and CHES CTF 2018, respectively. For all datasets, we consider 5 000 measurements as validation sets and another 5 000 as attack sets.

The fifth dataset is ESHARD-AES128, and it consists of side-channel measurements collected from a software-masked AES-128 implementation running on an ARM Cortex-M4 device. The AES implementation is protected with a first-order Boolean masking scheme and shuffling of the `S-box` operations. In this work, we consider a trimmed version of the dataset that is publicly available [1] and includes the processing of the masks and all

[1] https://gitlab.com/eshard/nucleo_sw_aes_masked_shuffled

`S-box` operations in the first encryption round without shuffling. This dataset contains 100 000 measurements, which are split into groups of 90 000, 5 000, and 5 000 for profiling, validation, and attack sets, respectively.

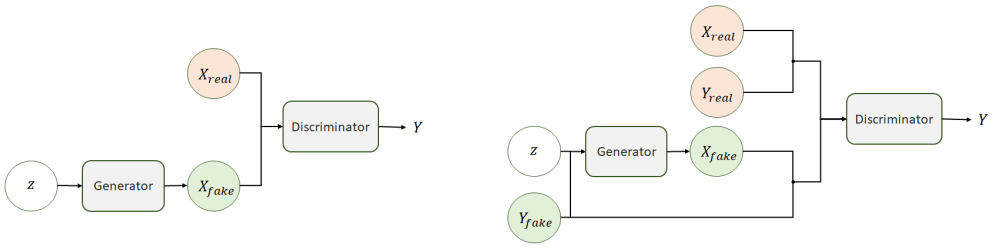
2.2 GANs and CGANs

Generative models are machine learning models that learn the underlying probability distribution of a given dataset. The primary objective of generative models is to generate new samples that resemble the training data in terms of statistical properties and structure. While discriminative models focus on learning the decision boundary between different classes or categories of data, generative models aim to understand and capture the characteristics and patterns of the entire dataset.

Generative adversarial networks proposed a novel way to train generative models [GPM+14]. The structure consists of two adversarial models competing against each other: a generator $\mathcal{G}$, with parameters θ_g, and a discriminator $\mathcal{D}$, with parameters θ_d. The main goal of the generator is to take input noise distribution $p(z)$ and to produce synthetic or fake output data $\mathcal{G}(z, \theta_g)$ that follows a data distribution present in real data. The discriminator is trained to provide the probability $\mathcal{D}(x, \theta_d)$ that an input data x comes from a real training set or from the generator. Both generator and discriminator are trained simultaneously in a way that θ_g is trained to minimize $\log(1 - \mathcal{D}(\mathcal{G}(z)))$ and θ_d is trained to minimize $\log \mathcal{D}(x)$, as following a min-max game with value function:

$$\min_{\mathcal{G}} \max_{\mathcal{D}} V(\mathcal{G}, \mathcal{D}) = \mathbb{E}_{x \sim p(x)}[\log \mathcal{D}(x)] + \mathbb{E}_{z \sim p(z)}[\log(1 - \mathcal{D}(\mathcal{G}(z)))]. \tag{1}$$

Conditional Generative Adversarial Networks (CGANs) [MO14] are a variant of the traditional GAN architecture incorporating additional information to guide the generation process. In CGANs, the generator and discriminator receive extra input in the form of conditional variables, which can be class labels, attribute vectors, or any other auxiliary information. This conditioning allows for the generation of more targeted and controlled outputs. By providing specific conditions, such as a particular class label, a CGAN can produce images or data samples that conform to desired characteristics. This capability makes such models particularly useful in various applications, such as image synthesis, style transfer, text-to-image generation, and data augmentation. By feeding the generator and discriminator networks with conditional variables, these models enable more precise control over the generated outputs. Figure 1 shows the structure of GAN and CGAN structures.



(a) GAN **(b)** CGAN

Figure 1: GAN and CGAN structures.

3 Related Works

The usage of generative models in side-channel analysis has been limited to a few applications. In [WCL+20], the authors considered generative adversarial networks for data augmentation. Later, a more elaborated analysis with conditional generative adversarial networks also considered data augmentation [MBPK22]. Both analyses were applied to protected AES implementations. In [ZBC+23], the authors considered Variational AutoEncoders (VAE) to generate reconstructed and synthetic traces that model the true conditional probability distribution of real side-channel traces. In [CLM23], the authors proposed the EVIL-machine, a framework using a GAN-like structure to find a suitable leakage model for the target device, replacing the need for prior knowledge of the leakage characteristics. The structure is extended to mount non-profiled attacks that exploit the learned leakage model. In [CZG+22], the authors presented an approach using a GAN-based structure to mitigate the issues related to the portability of profiling models. Similar to our framework, the authors extracted an intermediate representation of the leakages from a profiling device and then trained a generator to extract a similar representation from unlabeled attack traces measured on another device of the same model. They consider only unprotected implementations running on the same device model.

The CGAN-SCA framework that we propose in the next section acts as a feature extractor from raw datasets, and therefore our work can also be seen as a preprocessing method. Regarding the application of deep neural networks for preprocessing side-channel traces specifically, we refer to Section 4.2 from [PPM+22]. We emphasize that none of the related works consider a generative adversarial architecture to efficiently extract features from a target dataset by learning the probability distribution from an adversarial dataset, as detailed next.

4 CGAN-SCA Framework

This section proposes a novel profiling attack framework based on conditional generative adversarial networks for side-channel analysis (CGAN-SCA). To this end, we propose an extended profiling SCA threat model, as described next.

4.1 Extended Threat Model

The classic threat model commonly adopted in profiling SCA assumes that an adversary possesses an open (i.e., programmable) and identical version of the target device. Thus, the adversary collects a set of profiling measurements from the open device by setting the input (i.e., plaintexts or ciphertexts) and keys. In [PPM+22], this threat model is defined as a portability threat model, while the case when the same device is used for both profiling and attack phases is defined as a classic threat model.

We modify both classic and portability threat models for profiling attacks by adding one more assumption: the adversary has a second and similar implementation running on another and different device (we denote it as *reference implementation or dataset*) that runs an implementation similar to the target one (we denote it as *target implementation or dataset*). In our extended threat model, the adversary can implement a feature or points-of-interest selection from the reference implementation. We assume that all secret randomnesses are known concerning this reference implementation (i.e., a conventional white-box approach). For the target implementation, the adversary has no access to secret randomness and needs to perform a classical black-box profiling attack.

This threat model is realistic, as even public datasets could be used as a reference set. Since the public datasets have known secret information, if the target is similar, e.g., running AES with Boolean masking, we could utilize the public information with the

proposed CGAN framework. On the other hand, if information about the target device is known, such as countermeasures or low-level implementation details (e.g., information from documentation, masking scheme based on public literature, etc.), the adversary could potentially produce a customized reference implementation.

Throughout this paper, we refer to two categories of trace sets. When the trace set is placed as a reference dataset, we refer to it as X_{ref}. The trace set X_{target} is always referred to as the target dataset. A feature selection process over X_{ref} results in an adversarial dataset[2] for the proposed CGAN, also referred to as reference features f_{ref}. The features extracted with the proposed CGAN-based architecture, f_{target}, from X_{target} are referred to as target or generated features. N_f, is the number of features in each element of f_{ref} or f_{target} sets.

4.2 A Novel Conditional GAN Framework

The proposed Conditional GAN-based framework, referred to as CGAN-SCA, is illustrated in Figure 2. The structure consists of three main blocks:

1. Feature selection: this block receives at its inputs the set of reference side-channel measurements X_{ref} and the masks (randomness) associated with this dataset. This block outputs the features f_{ref} (i.e., the adversarial dataset), which should contain the most leaky samples from X_{ref}, similar to the points-of-interest selection.

2. Generator $\mathcal{G}$: this block receives the side-channel measurements from the target implementation at its inputs, X_{target}. The generator's output is the set of extracted features, f_{target}, also a latent representation of X_{target}.

3. Discriminator $\mathcal{D}$: this block receives at its input the set of features (f_{ref} or f_{target}) and the corresponding set of labels (Y_{ref} or Y_{target}). The output of the discriminator is a value representation of the loss function.

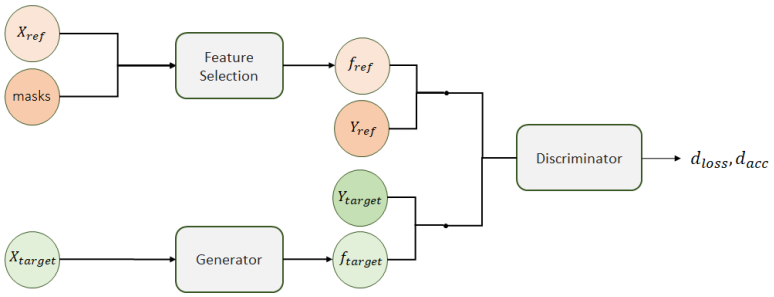


Figure 2: Proposed CGAN-SCA framework.

The main goal of training the proposed CGAN model is to generate f_{target} outputs with the same dimension as given by f_{ref} and with most of its features containing main side-channel leakages from X_{target}. Thus, the generator acts as a feature extraction or dimensionality reduction mechanism. Different from a classic CGAN structure where the generator receives at its inputs a random source and a label, *our generator receives only the original traces X_{target} from the target implementation.* We emphasize that this architecture is a new concept introduced in this paper.

The main goal in training the generator $\mathcal{G}$ is to learn parameters $\theta_{\mathcal{G}}$ such that new samples f_{target} are statistically indistinguishable from samples from reference f_{ref}. In other

[2]Note that the term adversarial is not connected with the domain of security of AI, e.g., adversarial examples but with the fact that it is a dataset used by an adversary and produced by GAN.

words, we train $\theta_{\mathcal{G}}$ to transform input target traces X_{target} to the probability distribution of f_{ref}. Determining the distance between two distributions is a two-sample hypothesis test problem, which is difficult for complicated distributions with high dimensions. Therefore, we will also judge the quality of the generator by computing the SNR between f_{target} (i.e., the output of the generator) and the high-order secret shares (i.e., mask shares and masked *S-box* output bytes) from the target device. In our threat model, an attacker does not know these high-order secret shares from X_{target}, and these values will not interfere with the training of generator and discriminator models. Here, we consider them only to visually confirm that the generator extracts meaningful representations from X_{target}. It is important to note that, during the CGAN training, X_{target} should contain only profiling measurements, as the structure requires the knowledge of its labels Y_{target}. In this paper, the labels Y_{ref} and Y_{target} are always the value of the `S-box` output byte from the first AES encryption round, without assuming the knowledge of any mask value. Only when the ESHARD-AES128 dataset is involved, labels Y_{ref} and Y_{target} are the Hamming weight values of this `S-box` output byte as this dataset leaks more on Hamming weight.

After training the CGAN structure, the generator is isolated and predicted with profiling, validation, and attack sets from X_{target}, as shown in Figure 3. Note how this feature extraction process does not involve any label. Next, a profiling attack is built from extracted features from these sets. In the attack phase, we obtain the probability $P(k)$ for each key candidate k, which allows us to derive the guessing entropy or success rate [SMY09] of the correct key.

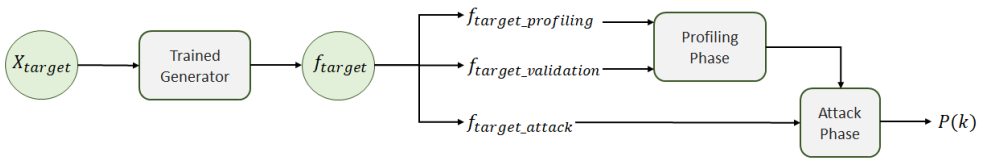


Figure 3: Profiling and attack phases.

4.3 Model Architectures in the Proposed CGAN Structure

To implement the CGAN-SCA framework, it is necessary to construct both the generator and discriminator neural networks. Thus, to define an efficient group of hyperparameters for generator and discriminator architectures, a new hyperparameter search process is necessary for every combination of reference and target datasets. In this paper, for the evaluated datasets described in Section 2.1, we performed a hyperparameter search with details given in this section, and we verified that finding efficient hyperparameters requires relatively low effort. More precisely, running up to 100 search attempts for both generator and discriminator always ends up in a highly efficient CGAN structure. However, we acknowledge that different ciphers or AES implementations with distinct countermeasures would likely require a more complex hyperparameter search process.

According to [SC22], tuning generative adversarial networks is challenging due to its inherent complexity. It is important to note that a well-performing generator does not necessarily imply a strong discriminator, and vice versa. The primary difficulty lies in establishing a reliable objective metric for hyperparameter tuning. One straightforward objective is to achieve a discriminator accuracy of 0.5 when classifying both real and fake data. Alternatively, a context-oriented approach can be employed, where the generated data is evaluated in a secondary task. In our specific case, this secondary approach follows the approach illustrated in Figure 3, which involves splitting X_{target} into profiling and validation traces. Therefore, a profiling attack metric, such as guessing entropy, success

rate, or perceived information [BHM+19], could be adopted as the reference metric to assess the quality of the CGAN architecture.

During the training of the CGAN model, the objective of the discriminator is to distinguish between f_{ref} and f_{target}. Conversely, the objective of the generator is to generate f_{target} that is similar to f_{ref}. While these objectives will result in realistic looking f_{target}, the generator is not forced to extract the side-channel leakages from X_{target} in any way as it is not conditioned. While a conventional CGAN model, where labels Y_{target} are provided to both the generator and the discriminator, seems like a straightforward solution to alleviate this problem, the labels are unavailable during the attack phase. In other words, the generator needs to convert X_{target} into f_{target} when X_{target} is the profiling, the validation, or the attack set. As labels are only known when X_{target} is either the profiling or the validation set, the generator cannot receive labels when X_{target} is the attack set, as these labels are unknown. As such, we provide labels only to the discriminator, which only received f_{target} that is generated when X_{target} is the profiling set, and therefore, labels are known. This choice allows the discriminator to check whether the provided leakages in f_{target} correspond to the label Y_{target}. This will then force the generator to use the side-channel leakages in X_{target} in its generated f_{target} as otherwise, the discriminator can easily classify the features as fake.

4.3.1 Discriminator Architecture

We first look at how to construct the discriminator model as a poorly configured discriminator will always result in the CGAN model failing to generate useful f_{target}. Our main goal in constructing the discriminator is to ensure it uses the leakages in f_{ref} and f_{target} and does not 'memorize' the correct f_{ref}. As several works have shown the capability of MLPs to learn to classify first-order protected datasets from relatively small intervals containing leaky samples [BPS+20] or even raw traces [PWP22], it should be relatively easy for an MLP-based discriminator to learn to combine leakages when its inputs contain only leaky samples. Developing architectures for other schemes should also be relatively straightforward, as full access to secret shares of the reference implementation is available. Pre-training (part of) the discriminator in a classification task, as is done in [CZG+22], can also be an option. Learning higher-order schemes can then be accomplished using knowledge of secret shares during training [MS21, DNG22].

The discriminator serves two primary purposes: (1) classifying the input, which comprises a combination of labels and features, into two classes (0 or 'fake' and 1 or 'real'), and (2) comprehending the relationship between labels and features. In the second case, we expect the discriminator to recognize an input combination of labels and features as 'real' if the features represent the corresponding label class. Suppose the discriminator cannot classify whether a given combination of features and labels is real or fake. In that case, we assume that the generated features, denoted as f_{target}, are as realistic as the reference features f_{ref}. The discriminator model is set with a binary cross-entropy loss function.

The number of features in f_{ref} and f_{target} is limited to $N_f = 100$, as the evaluated datasets contain a limited number of leaky points-of-interest to what concerns the processing of high-order leakages (e.g., masks and masked `S-box` output bytes). In the first experiments from Section 5, we define $N_f = 100$ for ASCADr, ASCADf, and DPAv4.2. For CHES CTF 2018 and ESHARD-AES128, we consider $N_f = 20$, as these two datasets are less leaky than previous ones. A more detailed analysis of the impact of N_f on the whole framework performance is provided in Section 6.2.

Figure 4 illustrates the generic structure of the MLP-based discriminator architecture. The input label (due to the conditioned fashion of the CGAN structure) is concatenated with the input features that can be either f_{ref} or f_{target}. For this architecture, we use relatively large fully-connected layers after the embedding layer of the class label. Later, in

Section 5, we refer to the number of dense layers after the embedding layer as *dense layers embedding*, in which the number of neurons in these layers will be referred to as *neurons embedding*. After the concatenation layer, we consider dense layers, and each one of them is followed by a dropout layer. Similarly to the embedding layers, the number of dense layers after the concatenation, whose are always interleaved with a dropout layer, will be referred as *dense layers dropout*, each one with a number of neurons referred to as *neurons dropout*. The output layer of the discriminator always employs the *sigmoid* activation function. Dropout layers are included in the discriminator as a means of regularization. To determine the optimal number of dense layers, their activation functions, and the corresponding number of neurons, it is recommended to perform hyperparameter tuning. To reduce the search space, this model utilizes the Adam optimizer with a learning rate of 0.0025 and a β value of 0.5. These hyperparameters are commonly employed in MLP-based profiling attacks [BPS+20, PPM+22], and we assume they will also yield favorable results in this case. We emphasize that tuning is performed for the rest of the hyperparameters.

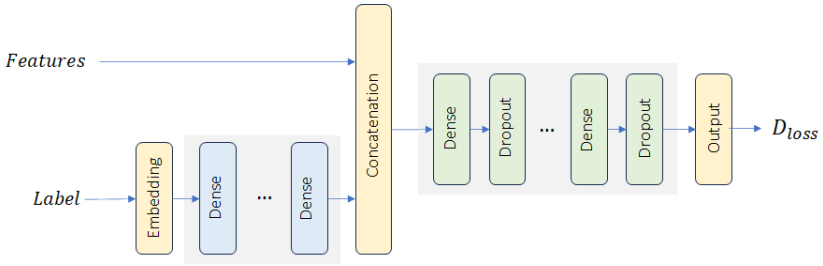


Figure 4: Generic architecture for the discriminator.

4.3.2 Generator Architecture

Different from the originally proposed CGAN structure [MO14] and its variants [ZPIE17, CDH+16], our generator receives at its input real data X_{target} rather than a noise distribution $p(z)$. The generator architecture is a simple MLP structure without any regularization mechanism. What is expected from the generator is to learn a mapping function $f(x, \theta_G) : X_{target} \rightarrow f_{target}$ representing a feature extraction process. When X_{target} is a set of side-channel traces collected from a first-order masked AES implementation, the generator is expected to transfer from the input to the output the features from X_{target} that contain the highest SNR values with respect to two secret shares (e.g., leaky points when the mask and a masked `S-box` output are processed).

While the task the generator needs to perform is conceptually fairly simple, in practice learning to extract leaky points-of-interest can be difficult. This is especially the case when attacks against the (resampled) full-length traces are considered. In Table 9 of [PWP22], we see that only between 0% and 5% of random models result in successful attacks against full-length traces, while when features are selected based on SNR values in the RPOI scenario, almost all of them can successfully recover the target key byte. As such, finding an architecture that is well-tuned to the task of extracting these features also requires hyperparameter tuning effort.

4.4 Assessing CGAN Efficiency

In the previous section, we discussed the primary objective of training the proposed CGAN architecture, which is to achieve automated feature extraction from the target dataset, X_{target}, using selected features, f_{ref} (manually selected from the reference dataset, X_{ref}), as adversarial examples. Our CGAN-based attack framework assumes that only the

reference device is fully controlled and that its secret shares are known. On the other hand, the secret shares of the target dataset, X_{target}, are unknown. This creates a challenging situation where accurately verifying the quality of the extracted features, denoted as f_{target}, from X_{target} becomes difficult. In simpler terms, we aim to measure the extent to which f_{target} represents the extracted high-order leakages from X_{target} when the target is an n-order masked implementation. To demonstrate the effectiveness of our CGAN solution, we utilize publicly available AES-128 datasets that also provide secret masks. Consequently, we calculate the Signal-to-Noise Ratio (SNR) of the secret shares derived from the extracted features, f_{target}, which comes from the generator's output. These SNR values are computed solely to confirm that the trained generator can automatically extract leakages from X_{target}. It is important to note that the CGAN model is neither trained nor validated using any information regarding the masks associated with the target dataset.

At the end of each CGAN training epoch, we predict the generator with the profiling set from the target dataset X_{target}, and we compute the SNR between extracted features f_{target} and the secret shares. This gives us two vectors with the same number of features from f_{target}. From these SNR vectors, we store the maximum SNR value. As results from Section 5 confirm, the generator can extract features from X_{target}, and the SNR values of secret masks from f_{target} are very high.

5 Experimental Results

This section presents the experimental results obtained using the publicly available datasets described in Section 2.1. The main goal is to run a hyperparameter search to find generator and discriminator architectures for different reference and target datasets combinations. In Section 8, we take the best CGAN architectures found in this section to conduct profiling attacks and compare them with the state-of-the-art. For each scenario using a different reference dataset, we apply the SNR-based approach described in Section 4.4 to assess the feature extraction quality of the corresponding best CGAN architecture against multiple target datasets.

5.1 CGAN Hyperparameter Search

A hyperparameter search is necessary to find an efficient CGAN architecture for each combination of reference and target datasets. Through preliminary experiments, we have already confirmed that identifying effective generator and discriminator architectures is a cost-effective process, as most of the hyperparameter combinations we have tested yield satisfactory results, but better groups of hyperparameters can be found. For this purpose, we employ a random search approach with predefined hyperparameter ranges, as outlined in Table 1. Dense layers may have different numbers of neurons for the generator, and the subsequent layer never has more neurons than the previous layer. This design choice reduces the search space. Due to the limited options available for each specific hyperparameter, the number of potential generator architectures is capped at 744, while the number of potential discriminator architectures is limited to 324. Consequently, there exists a total of 241 056 possible CGAN hyperparameter selections. In addition, the generator and discriminator employ the Adam optimizer with fixed learning rates. For the discriminator, we set the learning rate to 0.0025, while for the generator, the learning rate is set to 0.002. These values are selected arbitrarily, and after all the experiments, we confirmed that they could deliver good performances without necessarily being included as part of the search space. Similar to other neural network training procedures, the CGAN training process is conducted in batches, with a fixed batch size of 400 across all hyperparameter configurations and experiments from this paper.

The guessing entropy is computed from the validation set, which constitutes a fraction

Table 1: Hyperparameter search ranges for generator and discriminator architectures.

Generator		Discriminator	
Hyperparameter	Options	Hyperparameter	Options
Dense layers	1, 2, 3, 4	Dense layers Embedding	1, 2, 3
Neurons	100, 200, 300, 400, 500	Neurons Embedding	100, 200, 500
Activation Function	linear, relu, selu, elu, leakyrelu, tanh	Dense layers Dropout	1, 2, 3
		Neurons Dropout	100, 200, 500
		Dropout Rate	0.5, 0.6, 0.7, 0.8
		Activation Function	leakyrelu

of f_{target} and is not used for profiling purposes. In essence, for every hyperparameter search attempt, we perform a profiling attack on the target dataset to evaluate the effectiveness of the trained CGAN model. The CGAN model that generates an f_{target} output resulting in the most successful profiling attack is considered the optimal solution.

For each hyperparameter search attempt, which results in a trained CGAN architecture, we take the trained generator to convert X_{target} into f_{target} for profiling and validation sets from the target dataset. This procedure produces two preprocessed trace sets f_{target_prof} and f_{target_val} from profiling and validation sets, respectively. Next, we implement a profiling attack by training a 4-layer MLP (each layer with 100 neurons and elu activation function) for 100 epochs with f_{target_prof} and obtain attack results, which are guessing entropy and the corresponding number of validation traces f_{target_val} to reduce it to 1. We decided to compare to both ASCAD and DPAv4.2 datasets, which makes the CGAN-SCA framework inefficient, and it provided consistent key recovery results for the experiments conducted in this paper. The best CGAN architectures will be the ones that require the minimum number of validation traces f_{target_val} to recover the target key byte.

For labeling, results described in the next section always consider $\texttt{S-box}(d_{i,j} \oplus k_{i,j})$ where $d_{i,j}$ (resp. $k_{i,j}$) denote the j-th plaintext byte (resp. j-th key byte) from the i-th side-channel measurements. Only when ESHARD-AES128 is involved the datasets labeled according to the Hamming weight of `S-box` output bytes, i.e., $HW(\texttt{S-box}(d_{i,j} \oplus k_{i,j}))$, as this dataset leaks in this leakage model and no successful attack results were found otherwise. It is important to keep in mind that both reference and target datasets need to be labeled with same leakage model.

Next, we provide results for ASCADr, ASCADf, and DPAv4.2 as reference datasets. ESHARD-AES128 is not considered a reference dataset because it provides significantly fewer leakages in comparison to both ASCAD and DPAv4.2 datasets, which makes the CGAN-SCA framework not very efficient. We provide a more elaborate discussion in Section 8.

5.2 ASCADr as the Reference Dataset

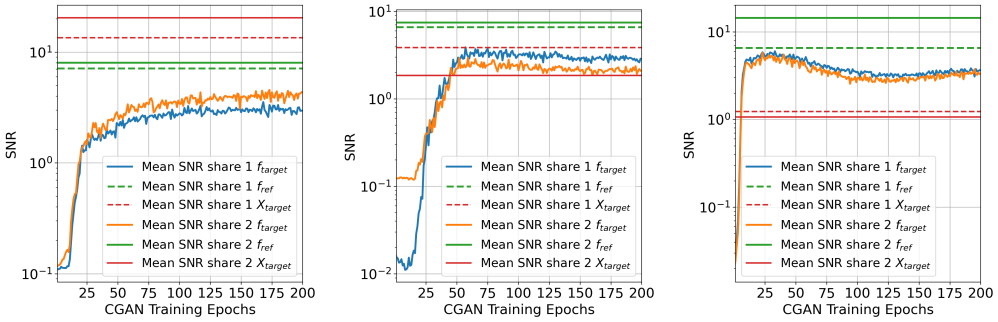
In our first analysis, ASCADr is considered as the reference dataset. We deploy a random hyperparameter search process for each target dataset with 100 search attempts. The CGAN is trained for 200 epochs for each of these search attempts. At the end of each training epoch, we compute SNR between generated features f_{target} and secret shares, specifically the masks and masked `S-box` output, available with the target dataset. It is important to note that, as mentioned in Section 4.4, these secret shares are assumed to be unknown to the attacker. However, in this context, we utilize their knowledge to provide evidence of our results.

Table 2 lists the best-found CGAN hyperparameters when ASCADf, DPAV4.2, ESHARD-AES128, and CHES CTF 2018 are considered as target datasets. Each profiling attack conducted after each hyperparameter search attempt is applied only to the target key byte. When the target dataset is ASCADf, the target key byte is 2, the first masked key byte in this dataset. For the DPAv4.2, ESHARD-AES128, and CHES CTF 2018 datasets, the target key byte during a random search is 0.

Table 2: Best CGAN hyperparameter for different target datasets when ASCADr as a reference dataset.

	Generator Network			
Hyperparameter	ASCADf	DPAv4.2	ESHARD-AES128	CHES CTF 2018
Dense layers	1	4	3	4
Neurons	300	200-200-200-100	500-500-500	100-100-100-100
Activation Function	linear	linear	leakyrelu	linear
	Discriminator Network			
Hyperparameter	ASCADf	DPAv4.2	ESHARD-AES128	CHES CTF 2018
Dense layers Embedding	2	1	1	2
Neurons Embedding	100	100	200	200
Dense layers Dropout	3	1	1	1
Neurons Dropout	200	200	200	200
Dropout Rate	0.7	0.8	0.7	0.5
Activation Function	leakyrelu	leakyrelu	leakyrelu	leakyrelu

After finding the best GGAN architecture for each target dataset, we repeat the CGAN training plus the profiling attack for the rest of the key bytes from the validation set. Figure 5 shows the evolution of the maximum SNR values for each secret share during CGAN training. This plot illustrates the results for all target key bytes, and the average SNR is illustrated in blue for share 1 and in orange for share 2. Results are provided for ASCADf, DPAv4.2, and ESHARD-AES128 as target datasets. We cannot produce SNR results for CHES CTF 2018 as this analysis requires the knowledge of secret masks for each separate trace. Note that these figures also show the maximum SNR values from the f_{ref} (in dashed green line) and X_{target} (dashed red line), which are averaged over SNR obtained from secret shares associated to each key byte. As we can see, for all target key bytes, the generator can extract features f_{target} from X_{target}, which results in high SNR values. This confirms that our proposed CGAN structure can efficiently extract features from high-order leaky points. In Section 7, a visualization analysis is applied to the generator to express in more detail what features are extracted from X_{target}.



(a) ASCADf. **(b)** DPAv4.2. **(c)** ESHARD-AES128

Figure 5: Performance of CGAN architecture against different target datasets, X_{target}, when ASCADr is the reference dataset.

5.3 ASCADf as the Reference Dataset

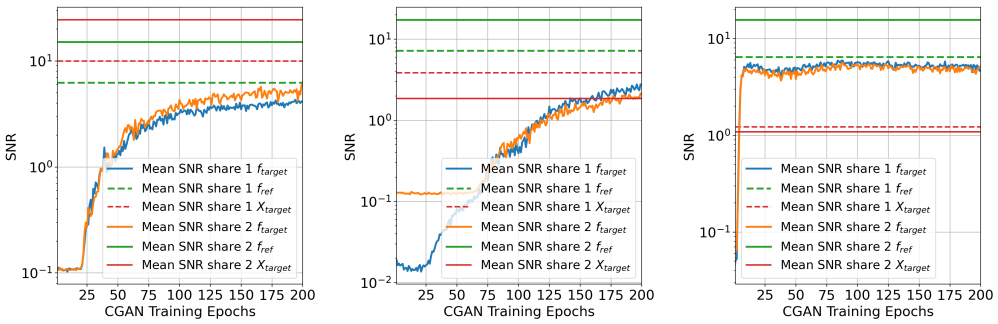
Table 3 lists the best-found CGAN architectures when ASCADf is set as the reference dataset and ASCADr, DPAv4.2, ESHARD-AES128, and CHES CTF 2018 are set as target datasets.

Similar to ASCADr, the ASCADf dataset provides side-channel measurements with high SNR levels with respect to high-order secret shares. Thus, in this case, the generator

Table 3: Best CGAN hyperparameter for different target datasets when ASCADf as a reference dataset.

	Generator Network			
Hyperparameter	ASCADr	DPAv4.2	ESHARD-AES128	CHES CTF 2018
Dense layers	3	2	2	4
Neurons	200-200-100	300-100	500-400	100-100-100-100
Activation Function	leakyrelu	linear	selu	linear
	Discriminator Network			
Hyperparameter	ASCADr	DPAv4.2	ESHARD-AES128	CHES CTF 2018
Dense layers Embedding	2	2	1	2
Neurons Embedding	200	200	500	200
Dense layers Dropout	1	2	1	1
Neurons Dropout	200	100	200	200
Dropout Rate	0.6	0.8	0.7	0.5
Activation Function	leakyrelu	leakyrelu	leakyrelu	leakyrelu

can also extract high-SNR features from target datasets, as shown in Figure 6. Again, the SNR values are always an average over the results obtained from all target key bytes. Note how the CGAN model converges relatively quickly during the CGAN training process when ESHARD-AES128 is set as the target dataset.



(a) ASCADr **(b)** DPAv4.2 **(c)** ESHARD-AES128

Figure 6: Performance of CGAN architecture against different target datasets, X_{target}, when ASCADf is the reference dataset.

The evolution of the SNR values from Figure 6 indicates that the CGAN architecture, when DPAv4.2 is the target dataset, needs considerably more epochs to start converging. One potential reason could be the relatively high dropout rate in the discriminator model, which aggressively regularizes this model. Another reason could be because ASCADf and DPAV4.2 contain a reduced number of profiling traces (50 000 and 70 000, respectively), which also limits the capacity of the generator. Still, a CGAN model that converges faster could be found with a more sophisticated hyperparameter search strategy.

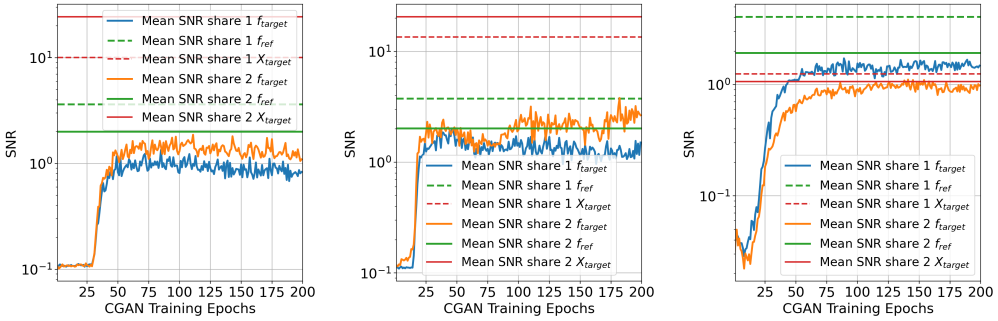
5.4 DPAv4.2 as the Reference Dataset

Next, the DPAV4.2 dataset is taken as the reference dataset for the CGAN-SCA framework. After running a random hyperparameter search for each target dataset, we select the best CGAN hyperparameters, as listed in Table 4.

Results provided in Figure 7 show that taking DPAv4.2 as the reference dataset also allows us to extract high SNR features from X_{target} when ASCADr, ASCADf, and ESHARD-AES128 are set as target datasets. It is also interesting to realize how the CGAN-based architecture converges relatively soon for all target datasets, which provides evidence that our proposed framework may not be very time-consuming.

Table 4: Best CGAN hyperparameter for different target datasets when DPAV4.2 as a reference dataset.

	Generator Network			
Hyperparameter	ASCADr	ASCADf	ESHARD-AES128	CHES CTF 2018
Dense layers	1	2	2	1
Neurons	100	500-100-100-100	400-300	100
Activation Function	elu	linear	selu	linear
	Discriminator Network			
Hyperparameter	ASCADr	ASCADf	ESHARD-AES128	CHES CTF 2018
Dense layers Embedding	1	1	1	1
Neurons Embedding	500	200	500	500
Dense layers Dropout	1	1	1	1
Neurons Dropout	100	100	500	500
Dropout Rate	0.6	0.7	0.6	0.8
Activation Function	leakyrelu	leakyrelu	leakyrelu	leakyrelu



(a) ASCADr. **(b)** ASCADf. **(c)** ESHARD-AES128

Figure 7: Performance of CGAN architecture against different target datasets, X_{target}, when DPAv4.2 is the reference dataset.

6 The Analysis of the Latent Space

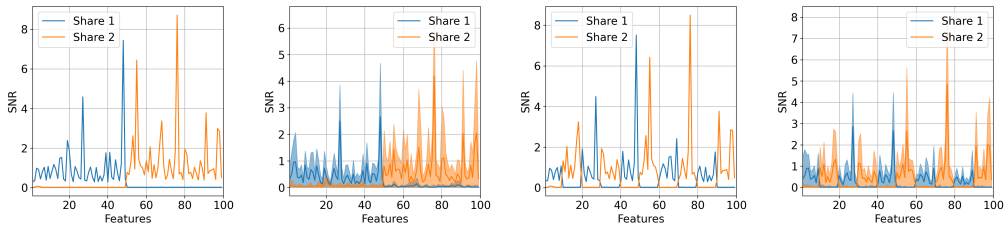
In this section, we analyze how variations in the construction of f_{ref} can impact how the network performs at extracting features. We first look at what the effect is of organizing leaky features in f_{ref} in various ways and whether the generator network can mimic these patterns accurately. Then, we look at varying the number of features, N_f.

6.1 Varying f_{ref} Leakage Pattern

Here, we analyze whether the generator network in the CGAN framework can mimic the leakage patterns present in the adversarial set f_{ref}. This analysis provides more insights into the relationship between the generator and discriminator. As explained before, the generator needs to extract main features from X_{target}, and it is important to confirm if these extracted features f_{target} follow the pattern from reference features f_{ref}. This analysis considers ASCADr as the reference dataset and ASCADf as the target dataset. This scenario was chosen as these datasets both have very high SNR peaks with respect to their secret shares (i.e., random masks and masked `S-box` output bytes) and are of the same implementation and device model, which simplifies the analysis without expensive hyperparameter tuning efforts. Note, however, that these datasets were acquired with distinct acquisition settings.

From the SNR-based feature selection process on ASCADr, we select 50 features for each secret share to have a total of $N_f = 100$. Thus, we organize these features in three different patterns, as shown in Figures 8a and 8c. During the training of the CGAN

architecture, at the end of each epoch, we compute the SNR levels for the secret shares on f_{target}, provided by the generator. Note in results given in Figures 8b and 8d how the generator learns to mimic precisely the leakage distributions from f_{ref}. These plots represent the range of minimum and maximum SNR values obtained during CGAN training. The solid lines represent the mean SNR values. These results confirm that our generator, even not being conditioned with labels, can generate feature extraction from the input target traces X_{target}. An essential insight derived from this analysis is the significant role played by the feature selection process in transforming X_{ref} into f_{ref} for the generator's feature extraction task. The number and distribution of leaky points of interest in f_{ref} directly impact the generator's performance on its task.



(a) f_{ref} (ASCADr). **(b)** f_{target} (ASCADf). **(c)** f_{ref} (ASCADr). **(d)** f_{target} (ASCADf).

Figure 8: SNRs of f_{ref} (left) with the corresponding f_{target} (right).

6.2 Varying the Number of Features N_f

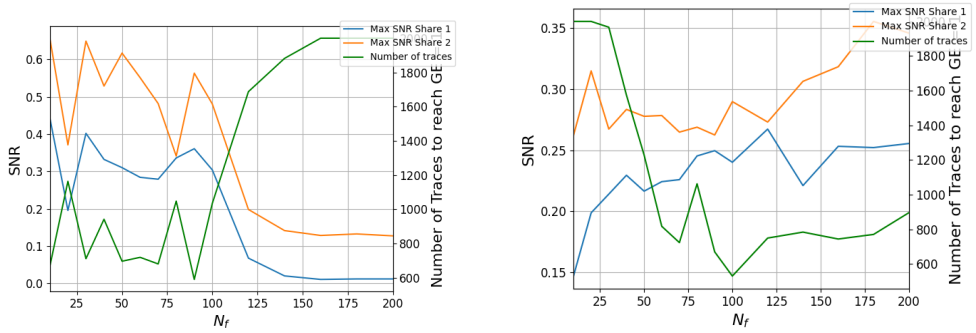
We analyze the performance of the CGAN-SCA framework for varying sizes of the feature space. We explore two scenarios: (1) ASCADr vs. DPAv42, in which the reference dataset (ASCADr) shows higher SNR peaks, with respect to the processing of two secret shares, than the target dataset (DPAv42) and (2) ESHARD-AES128 vs. ASCADf, in which the reference dataset (ESHARD-AES128) presents lower SNR peaks compared to the target dataset (ASCADf). The generator architecture we use in both scenarios is a 4-layer MLP with 100 neurons in the first 3 layers and N_f in its last layer with linear activations. For the discriminator, we set one embedding dense layer and one dropout dense layer, with 100 neurons. The dropout rate is 0.5. We average the results over 10 CGAN training runs.

In Figure 9a, we can see that reducing the size of the latent space from 100 does not significantly harm the attack performance when X_{ref} has higher SNR levels than X_{target}. When we consider lower N_f, the generator still seems to produce f_{ref} that contains the same amount of information as in cases with higher N_f. In cases where we consider N_f higher than 100 and lower SNR features are included in f_{ref}, the framework's performance is significantly degraded.

In Figure 9b, we can see that for much lower SNR f_{ref}, the performance can suffer for low N_f. We attribute this to the leakages included in f_{ref} being insufficient for effectively comparing leakages to Y_{ref} to classify whether the features are real. Another observation from Figure 9b is that successful attacks are possible here with N_f higher than 100. This is because ASCADf is a relatively easy dataset to attack, which mitigates the downsides of adding non-leaky features to f_{ref}.

These results indicate that when the SNR levels in f_{ref} are sufficient, the feature dimensions need to be set to a number that is not too large. Intuitively, we want to limit the number of lower SNR features included in f_{ref} and generate a 'cleaner' set of reference features. This way, the generator learns only to include leaky features and does not also need to mimic noisy features. Furthermore, reducing N_f too far can result in the discriminator being unable to utilize leakages in f_{ref}. This effect does not seem to

be present when the SNR levels of the f_{ref} are higher, but some careful consideration is required here.



(a) ASCADr vs. DPAv4.2 **(b)** ESHARD-AES128 vs. ASCADf

Figure 9: Training results for different N_f.

7 Visualizing Generator's Feature Extraction with LRP Attribution Method

In the previous section, we demonstrated that the generator effectively extracts features from X_{target} by mimicking the pattern observed in f_{ref}. Moreover, we verified how the size of the latent space, N_f, impacts the performance of the whole CGAN-SCA framework. This section applies the Layer-wise Relevance Propagation (LRP) [BBM+15] method to analyze the generator further. LRP is a cost-effective solution that provides interpretability and, for our case, confirms that the generator accurately captures leakage from actual leaky points-of-interest from X_{target}. The primary objective of this section is to present evidence that the generator, although not conditioned with labels, possesses the ability to extract features from the high-order leaky points-of-interests, rather than functioning solely as a preprocessing step that leads to dimensionality reduction.

In Figure 10, we provide two scenarios. The figure on the top-left shows the LRP values obtained from the trained generator when the reference dataset is ASCADr, and the target dataset is ASCADf. The generator's output produces f_{target} with $N_f = 100$ features per trace. For this case, the selected pattern for f_{ref} is exactly what is shown in Figure 8a. Thus, as the generated features f_{target} have the same shape as shown in Figure 8b, we compute LRP for the first 50 features for share 1 and the other 50 features for share 2. Comparing with the SNR values obtained from the same target key byte of ASCADf (plot on the bottom-left of Figure 10), we see that the generator extracts the correct features from X_{target}.

Furthermore, we present an example using the ESHARD-128 dataset. The generator is trained with ASCADf as the reference dataset in this case. Following the same process as in the previous example, we obtain the results depicted on the right side of Figure 10. It is noteworthy how the generator can extract features that align with the location of SNR peaks in relation to the processing of high-order leakages. This interpretability analysis confirms the generator's effectiveness in extracting high-order leakages from a target dataset when it is not even conditioned to any label class. Indeed, only conditioning the discriminator in our proposed CGAN structure is enough to implement efficient feature extraction from masked datasets. This discriminator receives labels from the X_{target} profiling set. However, as the CGAN structure never sees the labels from the target attack set and is still able to extract features from this attack set efficiently, we may

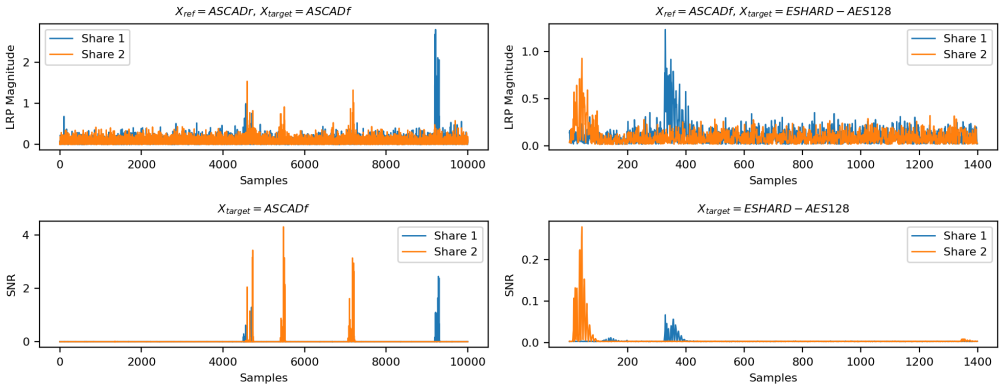


Figure 10: Comparison between LRP magnitude and SNR values from secret shares obtained for a single target key byte.

intuitively conclude that generator learns to extract input features from specific positions. The results in this section provide conditions to make the application of CGAN-SCA framework to black-box profiling attacks more interpretable. It points out the locations in the target dataset X_{target} where feature extraction can expose potential vulnerabilities in the implemented countermeasures.

8 Profiling Attacks and Comparison with State-of-the-Art

We employ state-of-the-art profiling attack methods as a benchmark to compare against our results. More precisely, we compare the number of attack traces that are necessary to achieve guessing entropy equal to 1 when the attack considers up to 2 000 traces. Also we compare the search success of an hyperparameters search process. The following analysis is conducted for each dataset:

- **CGAN-SCA with DL-based profiling attack (CGAN-SCA):** this attack is implemented with the CGAN-SCA framework presented in Section 4.2. The CGAN-SCA architecture is trained to achieve an efficient generator model that converts X_{target} traces into f_{target}. After obtaining f_{target} for both profiling and attack sets, we apply a DL-based profiling attack.

- **DL-based black-box profiling attack (BBDL):** in this case, we apply DL-based profiling SCA on datasets without feature selection. The attack is considered black-box as the profiling phase does not consider any knowledge about countermeasures or secret randomness.

- **DL-based white-box profiling attack (WBDL):** this profiling attack assumes that during profiling, an adversary can implement feature selection as countermeasures (i.e., the masking scheme) and secret randomness (i.e., secret masks) are known. Therefore, feature or points-of-interest selection can be applied to profiling and attack traces.

- **White-box Gaussian Template Attack (WBTA):** this process follows a white-box profiling attack in which points-of-interest are selected based on the set of highest SNR peaks obtained with the knowledge of secret masks. For all scenarios, we select 1 000 points-of-interest by targeting a second-order leakage function (500 points-of interest for each share), which are reduced with linear discriminant analysis (LDA) to 10 points-of-interest. Afterward, we build Gaussian templates with them.

- **White-box Gaussian Template Attack with WBTA-SASCA:** in this case, we apply Soft-Analytical SCA (SASCA)-based profiling attacks by following the white-box

approach proposed in [BCS21].

The first three profiling methods, which consist of deep learning-based profiling models, include a hyperparameter tuning process for a small MLP model. As the main idea here is to focus on the process that is efficient with less hyperparameter tuning efforts with respect to finding a good profiling model, we decided to limit the profiling model size to small MLP networks with up to four hidden layers. During this search, we vary the number of hidden layers, the number of neurons, activation function, learning rate, and training batch-size. For each of the 16 target key bytes from the full AES 128-bit key, we search for 100 random MLP architectures. Each of these MLP architectures is then trained, validated, and tested separately with:

1. f_{target_prof}, f_{target_attack}, and f_{target_val} sets, respectively, obtained by predicting the generator $\mathcal{G}$ with the profiling, validation and attack sets from the target device. This way, we implement the aforementioned **CGAN-SCA with DL-based profiling attack**;
2. original X_{target}, split into profiling, validation, and attack traces, to implement the aforementioned **DL-based black-box profiling attack**: BBDL.
3. SNR-based selected features from X_{target} profiling, validation, and attack traces, to implement the aforementioned **DL-based white-box profiling attack**: WBDL.

Table 5: The minimum number of attack traces to obtain guessing entropy equal to 1. The symbol x indicates that the target key byte is not recovered with 2 000 attack traces. The NA indicates that the attack is not applicable because the target key bytes are unprotected.

Method \ Target	k_0	k_1	k_2	k_3	k_4	k_5	k_6	k_7	k_8	k_9	k_{10}	k_{11}	k_{12}	k_{13}	k_{14}	k_{15}
	ASCADr															
CGAN-SCA (ref: ASCADf)	NA	NA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
CGAN-SCA (ref: DPAv4.2)	NA	NA	4	2	2	2	5	11	10	2	6	8	5	2	2	2
White-Box DL	NA	NA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
Black-Box DL	NA	NA	1	2	3	1	29	5	1	16	9	9	9	6	1	1
White-Box TA	NA	NA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
White-Box TA-SASCA	NA	NA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	ASCADf															
CGAN-SCA (ref: ASCADr)	NA	NA	1	1	1	1	1	1	4	1	5	1	15	5	1	5
CGAN-SCA (ref: DPAv4.2)	NA	NA	3	3	2	2	2	3	7	2	5	2	7	5	2	5
White-Box DL	NA	NA	1	1	1	1	1	1	4	1	2	1	3	3	1	1
Black-Box DL	NA	NA	9	6	9	8	18	x	27	6	20	6	x	34	17	7
White-Box TA	NA	NA	1	1	1	1	1	1	4	1	2	1	8	4	1	5
White-Box TA-SASCA	NA	NA	1	1	1	1	1	1	1	1	1	1	1	1	1	1
	DPAv4.2															
CGAN-SCA (ref: ASCADr)	1	3	2	5	3	2	4	2	7	2	3	3	2	2	5	2
CGAN-SCA (ref: ASCADf)	5	2	3	12	3	6	5	9	4	3	7	6	3	2	5	8
White-Box DL	1	1	1	2	3	1	2	1	3	1	2	2	1	1	2	1
Black-Box DL	x	315	140	x	x	x	145	x	x	x	x	x	x	x	x	x
White-Box TA	3	2	2	3	2	3	3	2	5	3	4	3	2	2	3	2
White-Box TA-SASCA	131	158	174	190	210	125	203	95	240	163	244	170	145	147	186	220
	CHES CTF 2018															
CGAN-SCA (ref: ASCADr)	36	24	22	20	51	19	21	36	34	18	25	23	24	22	22	19
CGAN-SCA (ref: ASCADf)	19	39	27	30	19	14	22	25	32	15	29	30	33	18	27	18
CGAN-SCA (ref: DPAv4.2)	91	47	36	115	159	200	73	138	858	56	136	50	557	78	124	52
Black-Box DL	x	471	367	77	668	132	304	121	136	957	83	662	x	459	413	380
	ESHARD-AES128															
CGAN-SCA (ref: ASCADr)	556	110	312	224	709	257	396	206	967	385	244	272	309	294	292	299
CGAN-SCA (ref: ASCADf)	491	124	528	357	153	532	641	493	622	552	373	513	732	406	454	572
CGAN-SCA (ref: DPAv4.2)	135	x	x	124	x	x	x	105	x	178	164	x	x	x	x	x
White-Box DL	640	154	875	727	x	774	799	667	x	846	487	745	116	649	818	1037
Black-Box DL	758	748	625	616	195	950	536	700	x	769	846	479	152	769	572	462
White-Box TA	67	81	97	89	110	75	123	107	152	100	100	89	127	87	111	111
White-Box TA-SASCA	119	120	107	144	234	131	142	135	267	121	116	125	164	132	105	112

Through this comparison, we emphasize the significantly reduced effort from the CGAN-SCA approach in finding an efficient profiling model that shows performance comparable to optimal profiling models, as is expected for WBDL, WBTA, and SASCA. Table 5 provides the performance of the five aforementioned profiling attack methods on datasets listed in

Section 2.1. For the case of CGAN-SCA methods, we provide results for different reference datasets. This table shows results with different colors to differentiate among profiling attack categories for better readability.

Table 6: Search success for MLP-based profiling attack with random hyperparameter search. The percentage indicates the number of successful MLP models out of 100, and it is averaged for all target key bytes.

Target \ Method	CGAN-SCA (ref: ASCADr)	CGAN-SCA (ref: ASCADf)	CGAN-SCA (ref: DPAv4.2)	White-Box DL	Black-Box DL
ASCADr	NA	72.80%	90.47%	99.88%	8.92%
ASCADf	64.22%	NA	68.25%	70.58%	9.24%
DPAv4.2	65.07%	62.16%	NA	63.68%	0.74%
CHES CTF 2018	61.10%	99.55%	33.15%	NA	12.14%
ESHARD-AES128	94.56%	54.48%	10.17%	63.68%	35.60%

The most remarkable results obtained with the CGAN-SCA framework are for the DPAv4.2 and CHES CTF 2018 datasets. Attack results obtained for CGAN-SCA are comparable to (and in some cases superior to) profiling attacks following worst-case assumptions, as is the case of WBDL, WBTA, and SASCA approaches. However, CGAN-SCA is a non-worst-case profiling attack, and results obtained for these two datasets indicate that our framework can deliver results that are close to white-box analyses. The results obtained with BBDL are inferior, especially for the DPAv4.2 dataset. As we explore various reference datasets for different target datasets, it becomes evident that certain reference datasets are more effective than others. This became apparent when comparing DPAv4.2 as a reference dataset with ESHARD-AES128 or CHES CTF 2018 as target datasets. One potential explanation is that DPAv4.2 does not exhibit the same level of leakage as the ASCAD datasets for high-order leakages. These findings serve as a cautionary reminder that the reference dataset should offer a minimum acceptable SNR level concerning the high-order intermediate variables of the target dataset.

Table 6 shows the search success from the hyperparameter search part of DL-based profiling attacks methods. The search success indicates the percentage of times a profiling model has reached the guessing entropy of 1 with less than 2 000 attack traces. The percentages are the average of all target key bytes. CGAN-SCA and WBDL present similar performances and are significantly superior to black-box DL. This is an impressive finding if we keep in mind that CGAN-SCA is a black-box (i.e., non-worst case) profiling approach. The results from Table 6 corroborate what was already shown in [PWP22]: spending significant effort on hyperparameter search process eventually results in a high-performing deep neural network against first-order masking in AES implementations. However, what matters in this table is the search success, which informs more about the chances of finding a good group of hyperparameters and training settings. Although black-box DL-based profiling attack result in successful attacks with (in some cases) very few required attack traces, the search success with CGAN-SCA framework and white-box DL approaches are significantly higher. For instance, when ASCADr is set as reference and DPAv4.2 is set as target, the search success for a Bback-box DL is 8.92% while for CGAN-SCA is 90.47%. For the case when ASCADf is the reference and CHES CTF 2018 is the target, the search success increases from 12.14% for a black-box approach to 99.55% with out CGAN-SCA framework. This justifies the need for feature selection (in the case of white-box) or feature extraction (in the case of CGAN-SCA framework) to speed up security evaluations. Since our proposed solution is also black-box, it becomes very attractive for efficiently assessing the security of masked implementations.

9 Discussion

Profiling attack results presented in this paper are aligned with the state-of-the-art for the evaluated datasets (see [PWP22] for the ASCAD, CHES CTF 2018, and DPAv4.2 datasets. To the authors' knowledge, there are no published ESHARD-AES128 dataset results for profiling attacks.). Such results were possible due to the following extra ingredients in a security assessment process:

1. **The usage of a (white-box) reference dataset.** The CGAN-SCA structure requires a reference device with similar implementation specifications to the target one. This paper shows that reference and target datasets can be gathered from different devices, cryptographic designs (with at least the same cryptographic algorithm with a similar masking scheme), varying source codes, and different acquisition setups. For some experimental examples, reference and target datasets also come from different side-channel types (e.g., power and electromagnetic analysis). Together with the availability of a reference implementation, it should also be possible to implement feature selection from this same implementation. This paper assumes that secret masks from the reference implementation are known to compute feature selection.
2. **The employment of a generative model for feature extraction from target side-channel measurements.** As specified in Section 4.2, the CGAN-SCA framework can implement feature extraction from a target dataset, and a reference dataset is used as an adversarial dataset. We are aware that this whole process increases the complexity of the analysis because a CGAN architecture (i.e., generator and discriminator neural networks) needs to be trained before applying a profiling attack on the extracted features from the target dataset. However, our experimental analysis demonstrated that when an efficient CGAN architecture is found, and the extracted features contain high SNR levels concerning the leakage of intermediate variable (e.g., masks and masked `S-Box` outputs), defining a profiling model becomes relatively easy. Therefore, in practice, the efforts to find an efficient profiling model (see [PWP22] where authors performed very costly hyperparameter tuning processes) are transferred to defining an efficient CGAN architecture.
3. **Hyperparameter tuning for generator and discriminator models.** An efficient CGAN architecture requires some carefully tuned generator and discriminator models. Overall, this is the only time-consuming part of the proposed CGAN-SCA framework. However, this whole process brings clear benefits, as a feature extraction process from raw side-channel measurements becomes possible without assuming any knowledge about low-level countermeasure details and secret randomness.

10 Conclusions and Future Work

This paper proposes a novel CGAN-based framework to automatically extract features from a target dataset when the adversarial dataset comes from a similar, open, and fully-controlled implementation. Our solution differs from conventional CGAN architectures from the literature: the generator receives real (target) traces instead of noise, and it is not conditioned with label class, allowing it to extract features from an unlabeled attack set. By applying our framework to five publicly available masked AES datasets, we obtain profiling attack results that significantly surpass the state-of-the-art in black-box security assessment and rival the performance of worst-case security evaluations. The proposed CGAN-SCA framework can precisely extract features from high-order leakages by mimicking the feature distribution present in a reference dataset. Our method makes hyperparameter tuning in a deep-learning-based profiling attack almost negligible, similar to white-box deep learning-based security evaluations.

For future work, we plan to investigate the effectiveness of CGAN architectures to

extract features from high-order masking schemes. Moreover, we plan to implement more complex generator and discriminator models, such as CNN-based architectures, which could extract features from desynchronized datasets. More complex CGAN structures could perhaps reduce some of our framework's limitations, such as using a reference dataset with a minimum acceptable SNR level regarding the n secret shares. A way to define a cost-efficient early stopping metric during CGAN training could also be an interesting research direction. Finally, we plan to explore whether the proposed structure can be adapted to non-profiling settings.

References

[BBM+15] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015.

[BCH+20] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020, San Diego, California, USA, February 23-26, 2020*. The Internet Society, 2020.

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

[BCS21] Olivier Bronchain, Gaëtan Cassiers, and François-Xavier Standaert. Give me 5 minutes: Attacking ASCAD with a single side-channel trace. *IACR Cryptol. ePrint Arch.*, page 817, 2021.

[BDMS22] Olivier Bronchain, François Durvaux, Loïc Masure, and François-Xavier Standaert. Efficient profiled side-channel analysis of masked implementations, extended. *IEEE Trans. Inf. Forensics Secur.*, 17:574–584, 2022.

[BHM+19] Olivier Bronchain, Julien M. Hendrickx, Clément Massart, Alex Olshevsky, and François-Xavier Standaert. Leakage certification revisited: Bounding model errors in side-channel security evaluations. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 713–737. Springer, 2019.

[BPS+20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *J. Cryptographic Engineering*, 10(2):163–188, 2020.

[CDH+16] Xi Chen, Yan Duan, Rein Houthooft, John Schulman, Ilya Sutskever, and Pieter Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In Daniel D. Lee, Masashi Sugiyama, Ulrike von Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2172–2180, 2016.

[CLM23] Valence Cristiani, Maxime Lecomte, and Philippe Maurine. The evil machine: Encode, visualize and interpret the leakage. In *Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing*, SAC '23, page 1566–1575, New York, NY, USA, 2023. Association for Computing Machinery.

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

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

[CZLG21] Pei Cao, Chi Zhang, Xiangjun Lu, and Dawu Gu. Cross-device profiled side-channel attack with unsupervised domain adaptation. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2021(4):27–56, 2021.

[DNG22] Elena Dubrova, Kalle Ngo, and Joel Gärtner. Breaking a fifth-order masked implementation of crystals-kyber by copy-paste. *IACR Cryptol. ePrint Arch.*, page 1713, 2022.

[GBTP08] Benedikt Gierlichs, Lejla Batina, Pim Tuyls, and Bart Preneel. Mutual information analysis. In Elisabeth Oswald and Pankaj Rohatgi, editors, *Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings*, volume 5154 of *Lecture Notes in Computer Science*, pages 426–442. Springer, 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 Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *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.

[KJJ99] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, pages 388–397, London, UK, UK, 1999. Springer-Verlag.

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

[MCLS23] Loïc Masure, Valence Cristiani, Maxime Lecomte, and François-Xavier Standaert. Don't learn what you already know scheme-aware modeling for profiling side-channel analysis against masking. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):32–59, 2023.

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

[MS21] Loïc Masure and Rémi Strullu. Side channel analysis against the anssi's protected AES implementation on ARM. *IACR Cryptol. ePrint Arch.*, page 592, 2021.

[PPM+22] Stjepan Picek, Guilherme Perin, Luca Mariot, Lichao Wu, and Lejla Batina. Sok: Deep learning-based physical side-channel analysis. *ACM Comput. Surv.*, oct 2022. Just Accepted.

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

[SC22] Divya Saxena and Jiannong Cao. Generative adversarial networks (gans): Challenges, solutions, and future directions. *ACM Comput. Surv.*, 54(3):63:1–63:42, 2022.

[SMY09] François-Xavier Standaert, Tal G. Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009*, pages 443–461, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

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

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

[ZPIE17] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*, pages 2242–2251. IEEE Computer Society, 2017.