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

Sengim Karayalcin[1] [0009-0000-1598-8400], Marina Krcek[2,3] [0000-0001-8475-1853], Lichao Wu[2,3] [0000-0002-7139-732X], Stjepan Picek[3] [0000-0001-7509-4337], and Guilherme Perin[1] [0000-0003-3799-7636]

[1] Leiden University, Leiden, Netherlands
{s.karayalcin, g.perin}@liacs.leidenuniv.nl
[2] Delft University of Technology, Delft, Netherlands
[3] Radboud University, Nijmegen, Netherlands
stjepan.picek@ru.nl

Abstract. Profiling side-channel analysis (SCA) is widely used to evaluate the security of cryptographic implementations under worst-case attack scenarios. This method assumes a strong adversary with a fully controlled device clone, known as a profiling device, with full access to the internal state of the target algorithm, including the mask shares. However, acquiring such a profiling device in the real world is challenging, as secure products enforce strong life cycle protection, particularly on devices that allow the user partial (e.g., debug mode) or full (e.g., test mode) control. This enforcement restricts access to profiling devices, significantly reducing the effectiveness of profiling SCA.

To address this limitation, this paper introduces a novel framework that allows an attacker to create and learn from their own white-box reference design without needing privileged access on the profiling device. Specifically, the attacker first implements the target algorithm on a different type of device with full control. Since this device is a white box to the attacker, they can access all internal states and mask shares. A novel conditional generative adversarial network (CGAN) framework is then introduced to mimic the feature extraction procedure from the reference device and transfer this experience to extract high-order leakages from the target device. These extracted features then serve as inputs for profiled SCA. Experiments show that our approach significantly enhances the efficacy of black-box profiling SCA, matching or potentially exceeding the results of worst-case security evaluations. Compared with conventional profiling SCA, which has strict requirements on the profiling device, our framework relaxes this threat model and, thus, can be better adapted to real-world attacks.

1 Introduction

Commonly used cryptographic algorithms, such as AES and 3DES, are 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, cryptographic implementations on hardware may introduce unintentional information leakages via, for instance, power consumption [26], electromagnetic emission (EM) [1], execution time [25], temperature [23], and acoustics [17]. These leakages could be exploited through side-channel analysis (SCA) and finally extract secret information.

The research community has known the threat of SCA for more than 25 years. Multiple attacks have been developed and can be generally categorized into two groups, depending on the availability of the profiling device. Non-profiling SCA leverages statistical methods, distinguishers, and leakage assessment techniques to launch direct side-channel attacks. Examples of such attacks include Differential Power Analysis [27], Correlation Power Analysis [5], and Mutual Information Analysis [18]. These attack methods are implicitly treated as real-world security threats, mainly because an attack is directly mounted on the victim's device; querying encryption or decryption executions are the only requirements to deploy the attack. On the other hand, research on profiling SCA, such as template attack [11] and deep learning-based SCA [44,33] has largely aimed at enabling worst-case security assessments [7] with an assumption of access to an identical copy of the target device. A profiling model is first built by mapping the relationship between the leakage measurements and the corresponding labels (key-related intermediate data) obtained from this copy. Then, an attacker collects leakage measurements from the target device and feeds them to the profiling model to obtain the label prediction. On top of this threat model, recent works further assume insight into values of mask shares generated during cryptographic executions [35], making the attack fully white-box. An attacker can profile each intermediate data or mask share and can finally combine the profiling results together to recover the secret. Although optimal attack performance can be reached via this threat model, we argue this threat model, including the access to an identical device copy and the mask-shares knowledge, is not realistic, as secure products often enforce robust life cycle protections, especially on devices that offer partial (e.g., debug mode) or full (e.g., test mode) user control. Even for the security evaluation labs that are supposed to have all design details [41], this threat model is overly strong, as the mask shares are commonly stored in protected registers and are not accessible even by a kernel user. Even in evaluation settings for software implementations, it is often not possible to access randomness as this would require modifications to the implementation (e.g., using a known seed or instrumenting source code), which results in the evaluation targeting a characterization of the implementation as opposed to the actual target [29]. On the other hand, if an attack can be performed with this attack assumption, attacks are significantly easier as an attacker can profile each individual mask share and finally recover the secret with, for instance, Soft Analytical Side-Channel Attacks [39].

This paper introduces a novel framework based on conditional generative adversarial networks (CGAN) to address the overly strong attack assumptions in profiling side-channel analysis (SCA). In line with [29], we assume that the attacker is aware of the cryptographic implementation and masking scheme used on the target device. With this design knowledge, the attacker sets up a similar cryptographic implementation on a *different type* of the device, referred to as the *reference implementation*. Since the attacker is not restricted by the device type, they can freely choose devices that grant full control and access to all internal states of the target algorithm, including mask shares. The proposed CGAN-based structure is then introduced to mimic white-box feature selection performed on the reference implementation and efficiently extract features from a target implementation with unknown masks. This framework transforms a conventional (black-box) profiling attack into a white-box profiling attack but with reduced attack assumptions. Additionally, our framework enhances the interpretability of the attack on the target dataset by splitting the feature extraction and exploitation phases, providing deeper insights into the attack process.

In summary, our main contributions are:

- We propose a novel conditional generative adversarial network-based SCA framework (CGAN-SCA) 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 significantly reduces the difficulties of finding an optimal profiling model in a non-worst-case security evaluation.
- The proposed CGAN-SCA framework can 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.
- Our results indicate that once an efficient CGAN architecture is found, a hyperparameter search for a profiling attack can be done with negligible effort, similar to a worst-case security evaluation.[4]

Our source code is available in a repository at.[5]

2 Background

2.1 GANs and CGANs

Generative models are machine learning models that learn the underlying probability distribution of a given dataset [19]. Their primary objective is to generate new samples that resemble the training data in terms of statistical properties and

[4] We refer to Section 6 of [29] for a discussion about difficulties in finding deep learning-based profiling models in worst and non-worst case security evaluations.

[5] `https://github.com/Sengim/cgan_sca`

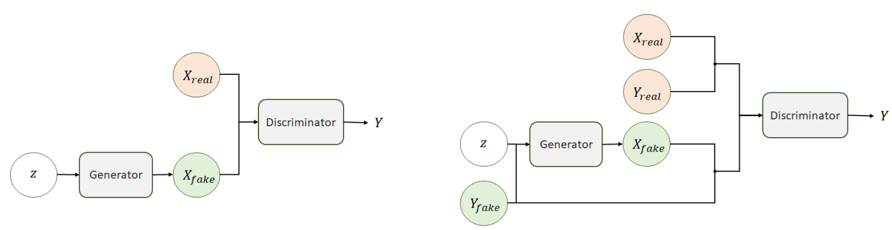
structure. While discriminative models focus on learning the decision boundary between different classes or categories of data, generative models aim to understand and capture the characteristics and patterns of the entire dataset.

Generative adversarial networks proposed a novel way to train generative models [20]. The overall idea is to adversarially train a generator and discriminator where the discriminator attempts to differentiate between real and generated images, and the generator is trained to generate fake images that fool the discriminator. These types of models have been used extensively across a wide variety of domains. Examples include image generation [20], image translation [24], and speech-synthesis [28].

As shown in Figure 1a, 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 the generator. Both generator and discriminator are trained simultaneously in a way that θ_g to minimize $\log(1 - \mathcal{D}(\mathcal{G}(z)))$ and θ_d to minimize $\log \mathcal{D}(x)$, as following a min-max game with value function:

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

Conditional Generative Adversarial Networks (CGANs) [31], illustrated in Figure 1b, 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 generating more targeted and controlled outputs.



(a) GAN (b) CGAN

Fig. 1: GAN and CGAN structures.

2.2 Generative Models for SCA

Generative models in side-channel analysis have been limited to a few applications. In [40], the authors considered generative adversarial networks for data augmentation. Later, a more elaborated analysis with conditional generative adversarial networks also considered data augmentation [32]. Both analyses were applied to protected AES implementations. In [43], the authors considered Variational AutoEncoders (VAE) to generate reconstructed and synthetic traces that model the true conditional probability distribution of real leakage traces. In [13], 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 [10], the authors presented an approach using a GAN-based structure to mitigate the issues related to the portability of profiling models. Like 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 considered only unprotected implementations running on the same device model. Finally, in [16], a GAN structure is used to translate between side-channel domains. To accomplish this, pairs of traces are required. Again, only unprotected implementations are considered. While these works consider GAN structures to transform leakage traces, these require paired measurements in different side-channel domains [16] or only consider portability [10]. As such, the differences between the adversarial and target datasets are fairly minimal, allowing for training the GAN structure without labels. In this work, the differences are more significant, necessitating the inclusion of labels in the discriminator to facilitate the convergence of our models.

When we look at applications in DLSCA where information on other implementations is utilized to build more powerful models, the use cases are still limited. Several works have looked at utilizing transfer learning techniques to limit the profiling complexity of attacking novel targets [37,15,42]. These works generally look at fine-tuning models that were pre-trained on a similar task, which reduces the number of profiling traces required from the profiling device. Similarly, several works incorporate knowledge of the masking scheme to implement tailored DL layers that explicitly recombine secret shares [9,29]. The main benefits here are, again, to reduce the number of required profiling traces. 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 applying deep neural networks for preprocessing leakage traces specifically, we refer to Section 4.2 from [35]. 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.

2.3 Signal-to-Noise Ratio (SNR)

The signal-to-noise ratio (SNR) measures the strength of a desired signal compared to the background noise level, indicating the clarity and quality of the signal in relation to the interference or irrelevant information present. In the side-channel analysis context, SNR is a measure of the amount of leakage that is present in a side-channel trace. It is defined as:

$$SNR = \frac{Var(\mathbf{E}(X|Y))}{\mathbf{E}(Var(X|Y))}. \tag{2}$$

Here, $\mathbf{E}$ represents the arithmetic mean function and Var is the variance. Generally, we assume X to be a single point in a side-channel trace, and we condition on Y, representing the intermediate value leaked.

3 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.

3.1 Threat Model and Notations

The classic threat model for profiled attacks in SCA assumes an attacker has an open and programmable copy of the attacked device [11]. With this privileged access right, an attacker can enable and disable countermeasures and has access to mask shares used to generate masks (on the profiling device). This threat model allows an evaluator to create templates for each secret share straightforwardly and then explicitly recombine the retrieved shares during the attack phase. As a consequence, this attack assumption leads to near-optimal attack performance with relatively limited resources [8]. However, as mentioned in the introduction, access to mask shares is often not possible in practical settings. In line with the scheme-aware threat model proposed in [29], where an attacker has (some) knowledge of the implementation specifics of the masking scheme but does not have access to mask shares, this work extends this threat model by including an additional device (on top of the profiling and attack device) that runs a similar implementation in a fully white-box setting (i.e., with access to mask values during profiling). We refer to this device as the reference implementation. Here, access to mask values is not a problem as the reference is a separate implementation the adversary develops themselves. Therefore, any restrictions that apply to the identical device clone that is used in conventional profiling SCA do not apply to the reference implementation. In scenarios where an attacker has access to the target source code, this can even be used to create the reference dataset by instrumenting it and taking separate measurements. In less permissive scenarios where an attacker only knows the type of countermeasure (i.e., the target is protected with first-order Boolean masking), they can create a different implementation that runs the same (type of) countermeasure. If a similar public dataset is available, the reference implementation could even be a publicly released dataset.

One may doubt the similarity between the reference device and the target device. It is intuitively clear that the more similar the reference and target device are, the better. However, when differences are too large, the framework might not be able to improve over standard profiling attacks. Broadly, it is advisable to take devices (and implementations) that operate on the same word sizes. In practice, this means that it seems unlikely that using a software device that operates on 8 bits of the internal state at a time as a reference can help when we are trying to attack a hardware (or bit-sliced) implementation that operates on larger states. Based on our results, improvements to Black-box attacks can be achieved even when reference and target are running on different device architectures and running different implementations of the same masking scheme. A more in-depth discussion can be found in Section 8.

Based on our threat model, we refer to three categories of trace sets: X_{ref}, X_{prof}, and X_{target}, representing leakages from the reference, profiling, and target devices, respectively. For clarity, X_{ref} has known key(s) and masks, X_{prof} has known key(s) and unknown masks, and X_{target} has unknown key and masks. A feature selection process over X_{ref} results in an adversarial dataset[6] for the proposed CGAN, also referred to as reference features f_{ref}. The features extracted with the proposed CGAN-based architecture from X_{prof} and X_{target} are referred to as target or generated features f_{prof} and f_{target}. N_f, is the number of features in each element of the f_{ref}, f_{prof}, or f_{target} sets. Note that we do not separately denote a set of validation traces as those are part of X_{prof}.

3.2 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 the following main blocks:

1. Feature selection (top of Figure 2): 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. In this paper, we consider different methods for feature selection: SNR, Linear Discriminant Analysis (LDA), and Principal Component Analysis (PCA). The feature selection process must only be done once for each reference dataset and feature selection method.
2. Generator $\mathcal{G}$ (middle of Figure 2): this block receives the side-channel measurements from the target implementation at its inputs, X_{prof} or X_{target}. The generator's output is the set of extracted features, f_{prof} or f_{target}, also a latent representation of the traces. It is trained to generate f_{prof} that looks real to the discriminator.
3. Discriminator $\mathcal{D}$ (upper right of Figure 2): this block receives at its input the set of features (f_{ref} or f_{prof}) and the corresponding set of labels (Y_{ref} or Y_{prof}). The output of the discriminator is a value representation of the

[6] 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 utilizing a GAN.

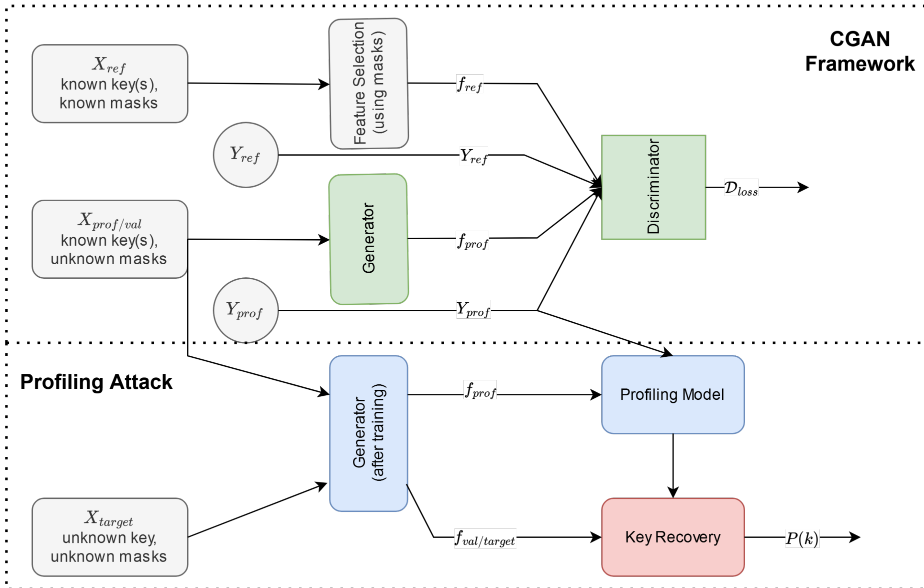


Fig. 2: Proposed CGAN-SCA framework. X_{ref} is the reference dataset with known secrets (i.e., masks and keys), X_{prof} is the profiling set with known key(s) and unknown masks, and X_{target} is the target device with unknown key and mask. Y_{ref} and Y_{prof} are corresponding labels to the X_{ref} and X_{prof} datasets, respectively.

loss function. The discriminator is trained to discriminate between real and generated features (f_{ref} and f_{prof}) using labels Y_{ref} and Y_{prof}.

4. Profiling attack (lower half of Figure 2): after the generator $\mathcal{G}$ is trained, it is used to generate f_{prof} and f_{target} and a profiling attack is applied on these features. The attack follows the same classic profiling attack structure (i.e., profiling and attack phases), where any type of profiling model can be used. The main difference is that the model is profiled with extracted features f_{prof} to attack f_{target}, which should contain only leaky points of interest from the original target traces.

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 and no labels.* This is important as for X_{target}, we do not have labels Y_{target}, and if the generator relies on labels for extracting features, it is not possible to apply the generator to X_{target} as labels are unavailable. We emphasize that this architecture is a new concept that has been 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_{prof} are statistically indistinguishable from samples from reference f_{ref}. In other 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 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, we can only use X_{prof}, as the structure requires the knowledge of its labels Y_{prof}. In this paper, the labels Y_{ref} and Y_{prof} are always the value of the `S-box` output byte from the first AES encryption round, without assuming the knowledge of any mask value.

After training the CGAN structure, the trained generator model is used to transform the profiling, validation, and attack sets from X_{prof} and X_{target} into f_{prof} and f_{target}, as shown in the bottom part of Figure 2. Note how this feature extraction process (i.e., predicting f_{target} from X_{target}) does not involve any label. In the next phase, we utilize the transformed f_{prof} and f_{target} sets to launch standard profiling attacks. 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 [36] of the correct key. Therefore, the main advantage of using the CGAN-SCA framework as a preprocessing step is that feature extraction can be done against a black-box profiling target, allowing key recovery results closer to white-box profiling attack performance without expensive hyperparameter tuning efforts, as shown in Section 7.

3.3 CGAN Architecture

We conducted preliminary experiments on the CGAN-SCA framework to determine well-performing (though not yet optimal) architectures for both the discriminator and generator models. Since this work only addresses synchronized datasets, we verified that small MLP-based architectures for the discriminator and generator already demonstrate satisfactory performance. However, a thorough hyperparameter search is essential to achieve better results. In the case of desynchronized measurements, CNN-based layers are highly recommended, but this is beyond the scope of this paper and will be explored in future works. In this section, we first describe the architectural choices for the discriminator and generator. Then, we discuss how to evaluate the efficacy of CGAN feature selection. We cover the specifics of our hyperparameter searches to find optimal solutions in Section 4.2.

During the training of the CGAN model, the objective of the discriminator is to distinguish between f_{ref} and f_{prof}. Conversely, the objective of the generator is to generate f_{prof} that is similar to f_{ref}. While these objectives will result in realistic-looking f_{prof}, the generator is not forced to extract the side-channel leakages from X_{prof} in any way as it is not conditioned. While a conventional CGAN model, where labels Y_{prof} 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} without labels. As such, we provide labels only to the discriminator, which only received f_{prof}. This choice allows the discriminator to check whether the provided leakages in f_{prof} correspond to the label Y_{prof}. This will then force the generator to use the side-channel leakages in X_{prof} in its generated f_{prof} as otherwise, the discriminator can easily classify f_{prof} as fake.

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_{prof} and does not 'memorize' the correct f_{ref}. Several works have shown the capability of MLPs to learn to classify first-order protected datasets from relatively small intervals containing leaky samples [3] or even raw traces [33]. Thus, 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 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 [10], can also be an option. Learning higher-order schemes can then be accomplished using knowledge of secret shares during training [30,14].

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. If the discriminator cannot classify whether a given combination of features and labels is real or fake, we assume that the generated features, denoted as f_{prof}, 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_{prof} is limited to a maximum of $N_f = 100$, 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 4, 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 more noisy than previous ones.

Figure 3 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_{prof}. For this architecture, we use relatively large, fully connected (dense) layers after the embedding layer of the class label. Later, in Section 4, 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 to 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 for binary classification. Dropout layers are included in the discriminator as a means of regularization. We recommended performing hyperparameter tuning, using random search [33] as detailed in Section 4.2, to determine the optimal number of dense layers, their activation functions, and the corresponding number of neurons. 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 [3,35], and we assume they will also yield favorable results in this case. We emphasize that tuning is performed for the rest of the hyperparameters.

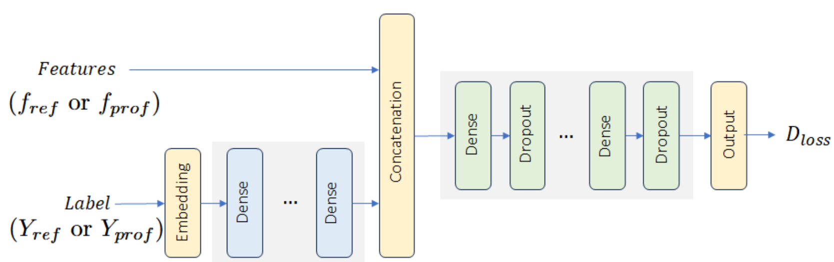


Fig. 3: Generic architecture for the discriminator.

Generator Architecture Different from the originally proposed CGAN structure [31] and its variants [45,12], our generator receives at its input real data X_{prof}/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 $f(x, \theta_{\mathcal{G}}) : X_{target} \rightarrow f_{target}$ representing a feature extraction process. When X_{target} is a set of leakage 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 in the case of the first order masked dataset.

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 true when attacks against the (resampled) full-length traces are considered. In Table 9 of [33], 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.

3.4 Assessing CGAN's Efficiency

Our CGAN-SCA framework assumes that only the reference device is fully controlled and that its secret shares are known. On the other hand, the randomness used to generate masks of the profiling/target dataset is unknown. This creates a challenging situation where accurately verifying the quality of the extracted features from X_{prof}/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-SCA solution, we utilize publicly available AES-128 datasets that also provide access to masks. Consequently, we calculate the 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}. We emphasize that the CGAN model is neither trained nor validated using any information regarding the masks associated with the target dataset. Thus, for the targeted implementation, the threat model always follows the classic black-box profiling attack scenario.

At the end of each CGAN training epoch, we predict the generator with the attack 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 the results from Section 4 confirm, the generator can extract features from X_{target}, and the SNR values of secret masks from f_{target} are high.

4 Experimental Results

This section first introduces the reference implementation we considered in this paper. Then, we perform a hyperparameter search to find generator and discriminator architectures for different reference and target dataset combinations. The best CGAN architectures are used to conduct profiling attacks and compare them with the state-of-the-art.

4.1 Datasets

Our framework requires limited similarity between the reference and target implementations. To illustrate this, this paper considers five publicly available AES software implementations, and each of them can serve as a reference implementation. The implementation details and side-channel measurement setup are detailed in Table 1. The AES is implemented on different platforms with different instruction set architectures and clock speeds. In terms of leakage measurement, besides the difference in the leakage sources, the side-channel acquisition process varies significantly between each implementation: ASCAD datasets were acquired with a sampling rate of 2G samples per second (S/s), the ESHARD-AES128 dataset was measured with a sampling rate of 200MS/s (for other datasets, this information is not available).

Table 1: Dataset setups. All the datasets implement the AES-128 algorithm.

Dataset	Side-Channel Type	Platform and ISA	Clock Speed	Countermeasure
ASCADf [3]	EM	AVR RISC (8 bits)	4MHz	Boolean Masking
ASCADr [3]	EM	AVR RISC (8 bits)	4MHz	Boolean Masking
DPAv4.2 [4]	Power	AVR MIPS (8 bits)	4MHz	RSM Masking
CHES CTF 2018 [22]	Power	ARM Cortex-M4 (32 bits)	168MHz	Boolean Masking
ESHARD-AES128 [38]	EM	ARM Cortex-M4 (32 bits)	30MHz	Boolean Masking

The side-channel leakages of four of them, namely ASCADr, ASCADf, DPAv4.2, and CHES CTF 2018, are the same as adopted for the NOPOI scenario in [33] (see Section 2.3 and Table 2 of [33] for specific details of the selected intervals). 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 a 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 [7] and includes the processing of the masks and all `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.

[7] `https://gitlab.com/eshard/nucleo_sw_aes_masked_shuffled`

ASCADf and ESHARD-AES128 datasets are the only datasets in which the profiling, validation, and attack keys are equal and fixed. For the rest of the datasets, profiling, validation, and attack keys differ.

4.2 CGAN Hyperparameter Search

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 sets of hyperparameters can be found. For this purpose, we employ a random search approach with predefined hyperparameter ranges, as outlined in Table 2. 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 0.0002. The imbalance in learning rates follows [21]. Like other neural network training procedures, the CGAN training process is conducted in batches, with a fixed batch size of 400 measurements across all hyperparameter configurations and experiments from this paper. Although the batch size and learning rates could also be included in the hyperparameter search process, we decided to fix these as using bad learning-rate/batch-size combinations would result in training an unnecessarily large amount of non-converging models.

Table 2: 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

To identify the best hyperparameter setup, we need an evaluation metric. As conventional Machine learning metrics are generally not suitable for assessing models in SCA [34], we perform a profiling attack (on the validation set) on extracted features to evaluate the trained models. The target dataset X_{prof} is split into profiling and validation sets. As illustrated in Figure 2, the input to the generator is only the profiling set from X_{prof}. After the CGAN training is finished for *every hyperparameter search attempt*, we predict on profiling and validation sets from X_{prof} with the trained generator. This gives us f_{prof} and f_{val}, respectively. For both sets, the keys are assumed to be known, allowing us to validate the whole process. To check how well the trained generator can extract leaky features from X_{prof}, we perform a profiling attack by training a profiling model with f_{prof} and by computing guessing entropy from f_{val}. The profiling model consists of a 4-layer MLP (each layer with 100 neurons and elu activation function) trained for 100 epochs. These hyperparameters were defined based on preliminary experiments and delivered relatively efficient profiling attack results. Here, we could also tune the profiling model architecture to find the optimal solution, which is a process that we cover in Section 7. The trained generator that extracts f_{prof} and f_{val} resulting in the most successful profiling attack (i.e., the profiling model that requires the least number of validation traces f_{val} to reach guessing entropy equal to one) is considered the optimal solution.

The inputs to the discriminator in the CGAN architecture include the extracted features (f_{ref} or f_{prof}) and their corresponding labels (y_{ref} or y_{prof}). The labels y_{ref} or y_{prof} refer to one output byte from the first `S-box` in the first AES encryption round: $\texttt{S-box}(d_{i,j} \oplus k_{i,j})$. $d_{i,j}$ (resp. $k_{i,j}$) denote the j-th plaintext byte (resp. j-th key byte) from the i-th side-channel measurement. Only when ESHARD-AES128 is involved, the datasets are 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. Note that y_{ref} or y_{prof} need to be labeled with the same leakage model.

Next, we provide results for ASCADr reference datasets. This dataset was selected as a reference here as it provides the best results across the board. Further results with different reference datasets are provided in Appendix A.

4.3 ASCADr as the Reference Dataset

In our first analysis, ASCADr is considered 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 3.4, 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 3 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 k_2, the first masked key byte in this dataset. For the DPAv4.2, ESHARD-AES128, and CHES CTF 2018 datasets, the target key byte is k_0.

Table 3: Best CGAN hyperparameter for different target datasets when ASCADr is 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

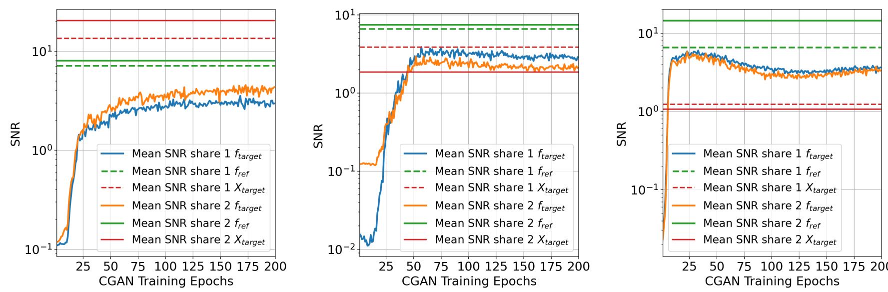
After finding the best CGAN architecture for each target dataset when ASCADr is set as the reference dataset, we repeat the CGAN training plus the profiling attack for the rest of the key bytes in the target dataset.

Our datasets are all first-order masked AES implementations. The extracted features f_{target} should contain leakages from the masked `S-box` output byte and the mask. However, as the Boolean masking operation is commutative, the generator cannot know what secret share should be the first or the second share. However, the order of secret shares in the generator's output has no impact on the whole process as long as the generator can extract leaky features from the two secret shares from X_{target}. Notably, for masking schemes where the order of share values matters for recombination, the generator should learn to order shares accordingly.

Figure 4 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 orange for share 2. The results are provided for ASCADf, DPAv4.2, and ESHARD-AES128 as target datasets.[8] 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 6, a visualization analysis is applied to the generator to express in more detail what features are extracted from X_{target}.

Another interesting outcome from the results shown in Figure 4 is that when DPAv4.2 and ESHARD-128 are the target datasets, the averaged SNR levels from f_{target} (extracted features) are higher than the SNR levels from X_{target} (raw datasets). This occurs due to averaging but also because for some of the target key bytes, the SNR from f_{target} is higher than the SNR from X_{target}.

[8] As masks are unavailable for the CHES_CTF 2018 dataset, we cannot perform this analysis for this target.



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

Fig. 4: Performance of CGAN architecture against different target datasets, X_{target}, when ASCADr is the reference dataset.

This emphasizes the high capability of the trained generator to act as an efficient feature extractor.

4.4 Profiling Complexity of the CGAN-SCA Framework

In this section, we evaluate the profiling complexity of the CGAN as a feature extractor and its impact on the complexity of a black-box profiling attack on the target dataset. This is conducted by varying the number of measurements from X_{prof} that is considered for the training of generator and discriminator architectures. This will allow us to check whether using reduced X_{prof} datasets still provides efficient applications of our proposed CGAN-SCA framework. An attacker that is limited in the number of measurements from the target device is a realistic assumption, and having a framework that works well under these circumstances is beneficial for security assessments. Note that we do not limit the number of traces that can be collected from a reference implementation as in our extended threat model. We assume this is not a serious limitation for an attacker.

In this experimental setup, we first find hyperparameters that work in more limited scenarios. The best architectures found in Section 4.2 using the full target datasets do not generalize well when fewer traces are available. Thus, we repeat the random hyperparameter search using the ranges provided in Section 4.2. We run 100 search attempts combination and select the best generator and discriminator architectures using the validation metric explained in Section 3.4. For this random search, we considered 30 0000 traces from X_{prof}.

Using the best-found generator and discriminator architectures, we then train CGAN models using the reference dataset and 10 000 through 70 000 traces with 10 000 trace steps. To check how the CGAN model trained with different numbers of profiling traces X_{prof} impacts the performance of a black-box profiling attack, we ran a random search using the obtained f_{prof} with varying numbers of profiling traces. 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. The hyperparameter ranges for the profiling attack model search are shown in Table 4.

Table 4: Hyperparameter search ranges for MLP as a profiling attack model.

Hyperparameter	Options
Dense layers	1, 2, 3, 4
Neurons	20, 50, 100, 200, 300, 400, 500
Activation Function	elu, selu, relu, leakyrelu, tanh
Learning Rate	0.001, 0.005, 0.0001, 0.0005
Batch Size	100, 200, 300, 400
Weight Initialization	random (normal/uniform), he (normal/uniform), glorot (normal/uniform)

For comparison, we also run these attacks in a white-box (WB) profiling scenario (following the white-box DL setup in Section 7) where features from X_{prof}/X_{target} are selected based on SNR.

As can be seen in Table 5, the CGAN framework can be used even in scenarios where only limited profiling traces are available from the target device. The columns indicate the number of profiling traces X_{prof} considered for training the CGAN architecture. Successful attacks are possible with only 10 000 profiling traces in both tested scenarios. While the attack results are not as efficient as with more traces, the ability of the CGAN network to learn in this limited scenario is somewhat surprising as the conventional discriminative DLSCA models often require significantly more profiling traces to generate efficient models [29]. In fact, our results are more aligned with the scheme-aware adversary who utilizes knowledge of the masking scheme to explicitly embed the combination of secret shares into a neural network layer (namely, the Grouprecombine- [29] and Bilinear [9] layers). As such, we note that including reference traces has similar benefits to these layers in terms of aiding the networks in learning the secret-share recombination.

Furthermore, in Table 5, we see that training a CGAN model with larger numbers of profiling traces can alleviate the need for using the full profiling set in the subsequent attack phase. The CGAN feature selection has a similar effect to selecting features in a white-box setting. While using more profiling traces has clear benefits regarding attack performance, the feature selection provided by the CGAN makes it significantly easier for attack models to converge in limited scenarios by eliminating the presence of uninformative samples. This emphasizes that the CGAN framework can, to an extent, emulate feature selection effectively without having access to the mask shares of a target device.

5 The Analysis of the Latent Space

In this section, we analyze how variations in the construction of f_{ref} can impact how the generator network performs at extracting features. We first look at

Table 5: Number of traces to reach GE=1 for varying numbers of profiling traces for ASCADr vs. DPAv4.2

	CGAN training traces							
Profiling Traces	10 000	20 000	30 000	40 000	50 000	60 000	70 000	WB
70 000	-	-	-	-	-	-	9	1
67 500	-	-	-	-	-	-	11	1
65 000	-	-	-	-	-	-	10	1
62 500	-	-	-	-	-	-	9	1
60 000	-	-	-	-	-	19	10	2
57 500	-	-	-	-	-	21	10	1
55 000	-	-	-	-	-	20	10	1
52 500	-	-	-	-	-	25	11	2
50 000	-	-	-	-	21	20	11	1
47 500	-	-	-	-	22	25	12	1
45 000	-	-	-	-	22	26	10	2
42 500	-	-	-	-	24	22	11	2
40 000	-	-	-	29	25	25	11	2
37 500	-	-	-	35	27	29	11	2
35 000	-	-	-	31	25	22	11	2
32 500	-	-	-	35	26	23	15	2
30 000	-	-	14	36	25	29	12	2
27 500	-	-	11	35	28	30	15	2
25 000	-	-	15	37	29	29	15	2
22 500	-	-	18	38	27	28	20	2
20 000	-	28	21	39	33	28	19	3
17 500	-	30	21	45	31	34	27	3
15 000	-	37	26	58	45	56	34	3
12 500	-	32	33	72	70	66	60	4
10 000	627	48	38	99	89	155	96	4
7 500	680	56	46	131	143	107	203	5
5 000	797	91	78	251	252	372	285	12

the effect of organizing leaky features in f_{ref} in various ways and whether the generator network can mimic these patterns accurately. Second, we investigate whether f_{ref} can also be created using alternative pre-processing methods, such as PCA and LDA.

5.1 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 is an expected outcome from the generator as it follows the principle of GAN architectures where the generator is trained to produce outputs that are statistically similar to the adversarial dataset (which, in our case, is given by f_{ref}).

This analysis considers ASCADr to be the reference dataset and ASCADf to be the target dataset. This scenario was chosen as these datasets have very high SNR peaks concerning their secret shares and are of the same implementation and device model, simplifying 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 two different patterns, as shown in Figures 5a and 5c. 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 the results given in Figures 5b and 5d 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 epochs (i.e., we compute the SNR at the end of each CGAN training epoch). The solid lines represent the mean SNR values. These results confirm that our generator can extract leaky features 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.

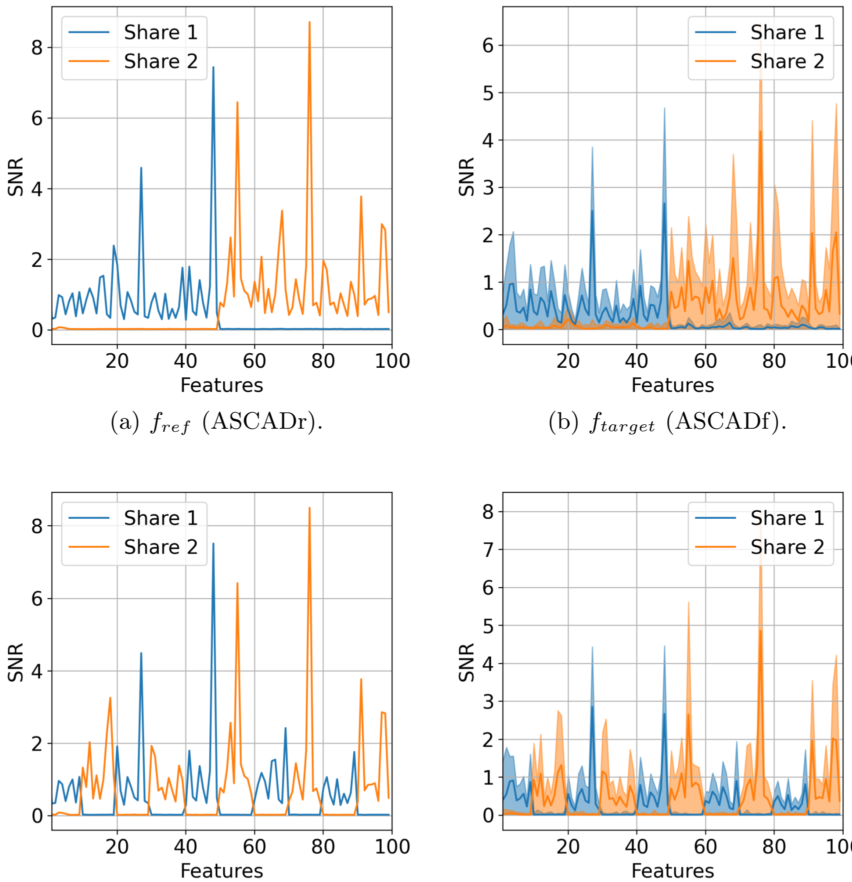
5.2 Varying Reference Feature Selection Method

While it is clear that the generator can effectively emulate feature selection of SNR peaks, this method is relatively straightforward when compared with methods currently used in literature, like LDA [6] or PCA. It is interesting to verify whether our framework allows alternative feature selection methods to be emulated. To this end, we run experiments using LDA and PCA for constructing f_{ref}. For both methods, we first select the 100 highest SNR features for each share and then transform these features into 5 components per share. Thus, in total, the number of features becomes $N_f = 10$. To test whether the framework can also emulate these methods, we run attacks against DPAv4.2 using ASCADr as a reference. To tune models for these cases, we run a hyperparameter search using the same ranges as in Section 4.2.

In Figure 6, we see that the more complex feature selection methods used for the reference dataset still result in converging generators. After training generator and discriminator models, when both PCA and LDA are taken into account for feature selection from the reference dataset, we apply profiling attacks on extracted features f_{prof} and f_{target}. We can retrieve the correct key byte with 4 and 3 traces for PCA and LDA, respectively. The final performance is similar to the performance of the generators in Section 4, and the attack performance is comparable to the attacks with the same datasets in Table 6. From these results, we can conclude that the CGAN framework is not limited to only using SNR-based feature selection and also performs well for alternative solutions.

6 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}. Additionally,



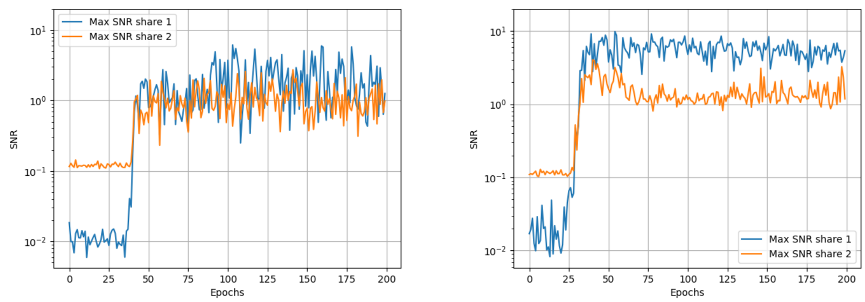
(a) f_{ref} (ASCADr). (b) f_{target} (ASCADf).

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

Fig. 5: SNRs of f_{ref} (left) with the corresponding f_{target} (right).

we also verified that the feature selection method to produce f_{ref} has little impact on the whole CGAN-SCA results. This section applies the Layer-wise Relevance Propagation (LRP) [2] 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, can extract features from the high-order leaky points of interest rather than functioning solely as a preprocessing step that leads to dimensionality reduction.

In Figure 7, we provide two scenarios. The figure on the top-left shows the LRP values obtained from the trained generator when the reference dataset is



(a) PCA (b) LDA

Fig. 6: Maximum SNR evolution for the best model hyperparameter search ASCADr vs. DPAv4.2 using LDA/PCA for generating f_{ref}.

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 5a. Thus, as the generated features f_{target} have the same shape as shown in Figure 5b, 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 7), we see that the generator extracts the correct features from X_{target}.

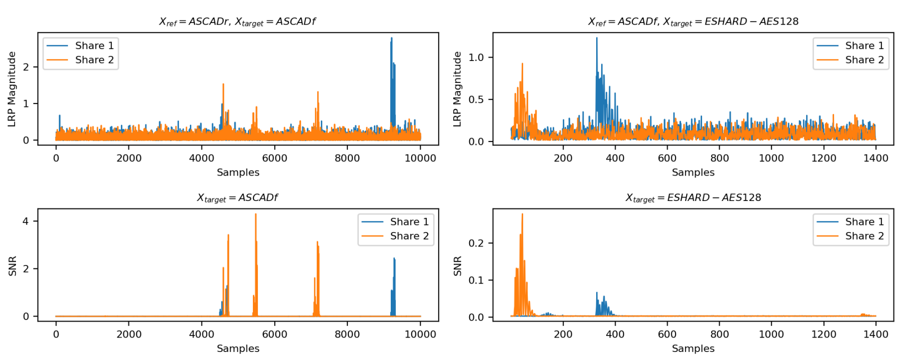


Fig. 7: Comparison between LRP magnitude and SNR values from secret shares obtained for a single target key byte.

Furthermore, we present an example using the ESHARD-128 dataset. In this case, the generator is trained with ASCADf as the reference dataset. Following the same process as in the previous example, we obtain the results depicted on the right side of Figure 7. It is noteworthy how the generator can extract features that align with the location of SNR peaks concerning 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. 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 intuitively conclude that the generator learns to extract input features from specific positions. The results in this section provide conditions to make the application of the 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.

7 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. Moreover, we compare the success of a hyperparameter 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 3.2. The CGAN-SCA architecture is trained to achieve an efficient generator model that converts X_{prof} and X_{target} traces into f_{prof},f_{target}. After obtaining these extracted features, 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 a 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 is reduced with linear discriminant analysis (LDA) to 10 points of interest. Afterward, we build Gaussian templates with them.

The first three profiling methods, which consist of deep learning-based profiling models, include a hyperparameter tuning process for a small MLP model.

For each of the 16 target key bytes from the full AES 128-bit key, we search for 100 random MLP architectures using the same hyperparameter ranges from Table 4. Each of these MLP architectures is then trained, validated, and tested separately with:

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

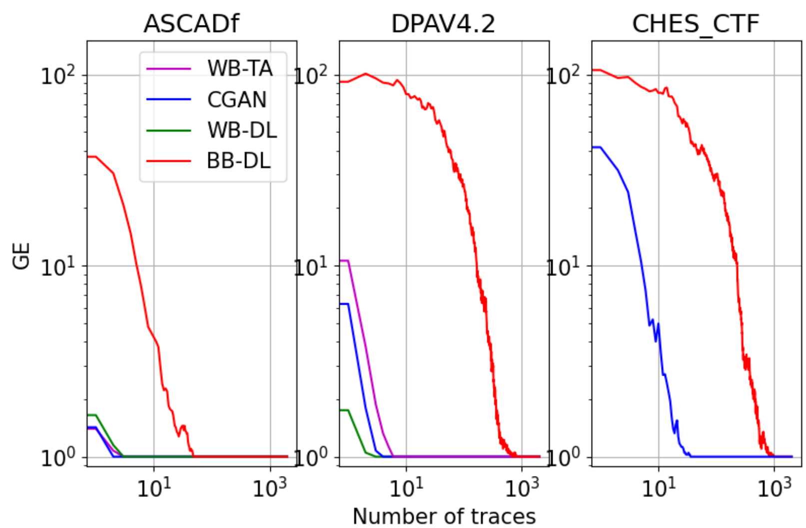


Fig. 8: GE results for key-byte 2 for various targets and methods (ref: ASCADr)

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 and WBTA. Table 6 provides the performance of the five aforementioned profiling attack methods on datasets listed in Section 4.1. 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.

As can be seen in Figure 8 and Table 6, the attacks using ASCAdr as a reference for all targets improve substantially over the BBDL attacks. Furthermore, in the best-case scenarios for ASCAD(r/f) and DPAv4.2, results are compara-

Table 6: 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
	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
	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	1454	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
	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	1327	304	1216	1369	957	83	662	x	459	413	380
	ESHARD-AES128															
CGAN-SCA (ref: ASCADr)	556	1105	312	224	709	257	396	206	967	385	244	272	309	294	292	299
CGAN-SCA (ref: ASCADf)	491	1248	528	357	1539	532	641	493	622	552	373	513	732	406	454	572
CGAN-SCA (ref: DPAv4.2)	1353	x	x	1242	x	x	x	1051	x	1787	1643	x	x	x	x	x
White-Box DL	640	1546	875	727	x	774	799	667	x	846	487	745	1162	649	818	1037
Black-Box DL	758	748	625	616	1957	950	536	700	x	769	846	479	1527	769	572	462
White-Box TA	67	81	97	89	110	75	123	107	152	100	100	89	127	87	111	111

Table 7: 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%

ble to attacks following white-box assumptions.[9] Only for ESHARD, and when DPAv4.2 is used as a reference, we see that white-box attacks still substantially outperform our attacks. We mostly attribute this to the larger difference in implementations/devices, which we discuss in more depth in Section 8.

Table 7 shows the search success from the hyperparameter search part of DL-based profiling attack 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 finding is impressive if we remember that CGAN-

[9] While it seems likely that CHES_CTF 2018 results are competitive with white-box attacks, we cannot verify this as mask values are not available.

SCA is a black-box (i.e., non-worst case) profiling approach. The results from Table 7 corroborate what was already shown in [33]: 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 attacks 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 a reference and DPAv4.2 is set as a target, the search success for a black-box DL is 0.74%, while for CGAN-SCA is 65.07%. 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 our 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.

8 Discussion

Profiling attack results presented in this paper are aligned with the state-of-the-art for the evaluated datasets (see [33] 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. **Using 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 3.2, 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 [33] where the 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.

Our results present three broad categories of 'similar' implementations, allowing us to give some takeaways on how similar the reference implementation must be:

1. **ASCAD(f/r) vs. ASCAD(f/r)**: The reference device and implementation are the exact same. This scenario can occur when an attacker/evaluator has access to the source code of an implementation but cannot alter this implementation on the target device. In such a scenario, the attacker/evaluator could utilize an instrumented version of the source to create a reference dataset. Our results in Section 4 show that the inclusion of this reference implementation results in significantly improved attack results over Black-Box DL attacks, and the results are competitive with White-Box approaches.
2. **ASCAD(f/r) vs. DPAv4.2**: The reference and target devices are similar in that both are 8-bit micro-controllers with RISC-based micro-architectures. The measurements for these targets are in different side-channel domains (EM for ASCAD vs Power for DPA). Both implementations incorporate Boolean masking-based countermeasures, although the specifics of the implementations differ somewhat. For DPAv4.2, an RSM-based masking scheme is employed, which results in 16 possible mask values, while for both ASCAD versions, we have 256 possible mask values. Results here still showcase strong improvements over Black-Box DL, especially when DPAv4.2 is the target, but the attacks are somewhat less efficient than White-Box attacks.
3. **(ASCAD/DPA) vs. (CHES_CTF/ESHARD)**: We target 32-bit microcontrollers with ARM micro-architectures while using 8-bit AVR microcontrollers as the reference. The implementations are broadly similar in that these are all software AES implementations protected with first-order Boolean masking. As we see in Table 6, the attacks against both CHES_CTF and ESHARD are better than the Black-Box DL attacks when ASCAD is used as the reference, while the results are similar to (CHES_CTF), or worse than(ESHARD) Black-Box attacks when DPA is used as the reference. Additionally, we see that for ESHARD, the performance of white-Box attacks is still significantly better than that of our CGAN-SCA setups.

In ideal cases where the reference implementation only differs in terms of allowing the knowledge of mask values,[10] we see that results are competitive with White-Box attacks. The device model and architecture similarity are more important than countermeasure implementation for other settings. While our results do not allow for strong requirements on the reference implementation, overall, the necessary 'similarity' to improve over Black-Box attacks is not extremely stringent. In some of the tested settings where the devices differ in terms of micro-architecture and implementation, we still see improvements over Black-Box attacks although the performance in these cases is worse than their white-box counterparts. In addition, it is more important to ensure the devices are similar in terms of, e.g., micro-architecture or bus size, over specific countermeasure implementation details (i.e., RSM vs. Bolean masking).

9 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 black-box security assessment and rival the performance of worst-case (white-box) 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 potentially 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.

Acknowledgements

This work was performed using the ALICE compute resources provided by Leiden University.

[10] The ASCAD(f/r) vs. ASCAD(f/r) scenario.

References

1. Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P.: The EM side-channel(s). In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 29–45. Springer (2002). https://doi.org/10.1007/3-540-36400-5_4
2. Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.R., Samek, W.: On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. PLOS ONE **10**(7), 1–46 (07 2015). https://doi.org/10.1371/journal.pone.0130140
3. Benadjila, R., Prouff, E., Strullu, R., Cagli, E., Dumas, C.: Deep learning for side-channel analysis and introduction to ASCAD database. J. Cryptographic Engineering **10**(2), 163–188 (2020). https://doi.org/10.1007/s13389-019-00220-8
4. Bhasin, S., Bruneau, N., Danger, J., Guilley, S., Najm, Z.: Analysis and improvements of the DPA contest v4 implementation. In: Chakraborty, R.S., Matyas, V., Schaumont, P. (eds.) Security, Privacy, and Applied Cryptography Engineering - 4th International Conference, SPACE 2014, Pune, India, October 18-22, 2014. Proceedings. Lecture Notes in Computer Science, vol. 8804, pp. 201–218. Springer (2014). https://doi.org/10.1007/978-3-319-12060-7_14, https://doi.org/10.1007/978-3-319-12060-7_14
5. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: Joye, M., Quisquater, J. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings. Lecture Notes in Computer Science, vol. 3156, pp. 16–29. Springer (2004). https://doi.org/10.1007/978-3-540-28632-5_2
6. Bronchain, O., Cassiers, G., Standaert, F.: Give me 5 minutes: Attacking ASCAD with a single side-channel trace. IACR Cryptol. ePrint Arch. p. 817 (2021), https://eprint.iacr.org/2021/817
7. Bronchain, O., Durvaux, F., Masure, L., Standaert, F.: Efficient profiled side-channel analysis of masked implementations, extended. IEEE Trans. Inf. Forensics Secur. **17**, 574–584 (2022). https://doi.org/10.1109/TIFS.2022.3144871
8. Bronchain, O., Standaert, F.: Side-channel countermeasures' dissection and the limits of closed source security evaluations. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2020**(2), 1–25 (2020). https://doi.org/10.13154/tches.v2020.i2.1-25
9. Cao, P., Zhang, C., Lu, X., Gu, D., Xu, S.: Improving deep learning based second-order side-channel analysis with bilinear CNN. IEEE Trans. Inf. Forensics Secur. **17**, 3863–3876 (2022). https://doi.org/10.1109/TIFS.2022.3216959
10. Cao, P., Zhang, H., Gu, D., Lu, Y., Yuan, Y.: AL-PA: cross-device profiled side-channel attack using adversarial learning. In: Oshana, R. (ed.) DAC '22: 59th ACM/IEEE Design Automation Conference, San Francisco, California, USA, July 10 - 14, 2022. pp. 691–696. ACM (2022). https://doi.org/10.1145/3489517.3530517
11. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: Jr., B.S.K., Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers. Lecture Notes in Computer Science, vol. 2523, pp. 13–28. Springer (2002). https://doi.org/10.1007/3-540-36400-5_3
12. Chen, X., Duan, Y., Houthooft, R., Schulman, J., Sutskever, I., Abbeel, P.: Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In: Lee, D.D., Sugiyama, M., von Luxburg, U., Guyon, I.,

Garnett, R. (eds.) Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain. pp. 2172–2180 (2016), https://proceedings.neurips.cc/paper/2016/hash/7c9d0b1f96aebd7b5eca8c3edaa19ebb-Abstract.html
13. Cristiani, V., Lecomte, M., Maurine, P.: The evil machine: Encode, visualize and interpret the leakage. In: Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing. p. 1566–1575. SAC '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3555776.3577688
14. Dubrova, E., Ngo, K., Gärtner, J.: Breaking a fifth-order masked implementation of crystals-kyber by copy-paste. IACR Cryptol. ePrint Arch. p. 1713 (2022), https://eprint.iacr.org/2022/1713
15. Genevey-Metat, C., Gérard, B., Heuser, A.: On what to learn: Train or adapt a deeply learned profile? IACR Cryptol. ePrint Arch. p. 952 (2020), https://eprint.iacr.org/2020/952
16. Genevey-Metat, C., Heuser, A., Gérard, B.: Trace-to-trace translation for SCA. In: Grosso, V., Pöppelmann, T. (eds.) Smart Card Research and Advanced Applications - 20th International Conference, CARDIS 2021, Lübeck, Germany, November 11-12, 2021, Revised Selected Papers. Lecture Notes in Computer Science, vol. 13173, pp. 24–43. Springer (2021). https://doi.org/10.1007/978-3-030-97348-3_2
17. Genkin, D., Shamir, A., Tromer, E.: RSA key extraction via low-bandwidth acoustic cryptanalysis. In: Garay, J.A., Gennaro, R. (eds.) Advances in Cryptology - CRYPTO 2014 - 34th Annual Cryptology Conference, Santa Barbara, CA, USA, August 17-21, 2014, Proceedings, Part I. Lecture Notes in Computer Science, vol. 8616, pp. 444–461. Springer (2014). https://doi.org/10.1007/978-3-662-44371-2_25
18. Gierlichs, B., Batina, L., Tuyls, P., Preneel, B.: Mutual information analysis. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2008, 10th International Workshop, Washington, D.C., USA, August 10-13, 2008. Proceedings. Lecture Notes in Computer Science, vol. 5154, pp. 426–442. Springer (2008). https://doi.org/10.1007/978-3-540-85053-3_27
19. Goodfellow, I., Bengio, Y., Courville, A.: Deep Learning. MIT Press (2016), http://www.deeplearningbook.org
20. Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A.C., Bengio, Y.: Generative adversarial nets. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. pp. 2672–2680 (2014), https://proceedings.neurips.cc/paper/2014/hash/5ca3e9b122f61f8f06494c97b1afccf3-Abstract.html
21. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. In: Guyon, I., von Luxburg, U., Bengio, S., Wallach, H.M., Fergus, R., Vishwanathan, S.V.N., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA. pp. 6626–6637 (2017), https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html
22. Hu, Y., Zheng, Y., Feng, P., Liu, L., Zhang, C., Gohr, A., Jacob, S., Schindler, W., Buhan, I., Tobich, K.: Machine learning and side channel analysis in a CTF

competition. IACR Cryptol. ePrint Arch. p. 860 (2019), https://eprint.iacr.org/2019/860
23. Hutter, M., Schmidt, J.: The temperature side channel and heating fault attacks. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications - 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers. Lecture Notes in Computer Science, vol. 8419, pp. 219–235. Springer (2013). https://doi.org/10.1007/978-3-319-08302-5_15
24. Isola, P., Zhu, J., Zhou, T., Efros, A.A.: 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. pp. 5967–5976. IEEE Computer Society (2017). https://doi.org/10.1109/CVPR.2017.632
25. Kocher, P.C.: Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In: Proceedings of CRYPTO'96. LNCS, vol. 1109, pp. 104–113. Springer-Verlag (1996)
26. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M.J. (ed.) Advances in Cryptology - CRYPTO '99, 19th Annual International Cryptology Conference, Santa Barbara, California, USA, August 15-19, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1666, pp. 388–397. Springer (1999). https://doi.org/10.1007/3-540-48405-1_25
27. Kocher, P.C., Jaffe, J., Jun, B.: Differential power analysis. In: Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology. pp. 388–397. CRYPTO '99, Springer-Verlag, London, UK, UK (1999), http://dl.acm.org/citation.cfm?id=646764.703989
28. Kong, J., Kim, J., Bae, J.: Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual (2020), https://proceedings.neurips.cc/paper/2020/hash/c5d736809766d46260d816d8dbc9eb44-Abstract.html
29. Masure, L., Cristiani, V., Lecomte, M., Standaert, F.: 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). https://doi.org/10.46586/tches.v2023.i1.32-59
30. Masure, L., Strullu, R.: Side channel analysis against the anssi's protected AES implementation on ARM. IACR Cryptol. ePrint Arch. p. 592 (2021), https://eprint.iacr.org/2021/592
31. Mirza, M., Osindero, S.: Conditional generative adversarial nets. CoRR **abs/1411.1784** (2014), http://arxiv.org/abs/1411.1784
32. Mukhtar, N., Batina, L., Picek, S., Kong, Y.: Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. In: Galbraith, S.D. (ed.) Topics in Cryptology - CT-RSA 2022 - Cryptographers' Track at the RSA Conference 2022, Virtual Event, March 1-2, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13161, pp. 297–321. Springer (2022). https://doi.org/10.1007/978-3-030-95312-6_13
33. Perin, G., Wu, L., Picek, S.: 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). https://doi.org/10.46586/tches.v2022.i4.828-861, https://tches.iacr.org/index.php/TCHES/article/view/9842

34. Picek, S., Heuser, A., Jovic, A., Bhasin, S., Regazzoni, F.: The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. IACR Transactions on Cryptographic Hardware and Embedded Systems **2019**(1), 209–237 (Nov 2018). https://doi.org/10.13154/tches.v2019.i1.209-237, https://tches.iacr.org/index.php/TCHES/article/view/7339
35. Picek, S., Perin, G., Mariot, L., Wu, L., Batina, L.: Sok: Deep learning-based physical side-channel analysis. ACM Comput. Surv. (oct 2022). https://doi.org/10.1145/3569577, just Accepted
36. Standaert, F.X., Malkin, T.G., Yung, M.: A unified framework for the analysis of side-channel key recovery attacks. In: Joux, A. (ed.) Advances in Cryptology - EUROCRYPT 2009. pp. 443–461. Springer Berlin Heidelberg, Berlin, Heidelberg (2009)
37. Thapar, D., Alam, M., Mukhopadhyay, D.: Deep learning assisted cross-family profiled side-channel attacks using transfer learning. In: 22nd International Symposium on Quality Electronic Design, ISQED 2021, Santa Clara, CA, USA, April 7-9, 2021. pp. 178–185. IEEE (2021). https://doi.org/10.1109/ISQED51717.2021.9424254
38. Vasselle, A., Thiebeauld, H., Maurine, P.: Spatial dependency analysis to extract information from side-channel mixtures: extended version. J. Cryptogr. Eng. **13**(4), 409–425 (2023). https://doi.org/10.1007/S13389-022-00307-9, https://doi.org/10.1007/s13389-022-00307-9
39. Veyrat-Charvillon, N., Gérard, B., Standaert, F.X.: Soft analytical side-channel attacks. In: Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014. Proceedings, Part I 20. pp. 282–296. Springer (2014)
40. Wang, P., Chen, P., Luo, Z., Dong, G., Zheng, M., Yu, N., Hu, H.: Enhancing the performance of practical profiling side-channel attacks using conditional generative adversarial networks. CoRR **abs/2007.05285** (2020), https://arxiv.org/abs/2007.05285
41. Wu, L., Perin, G., Picek, S.: Not so difficult in the end: Breaking the lookup table-based affine masking scheme. In: Carlet, C., Mandal, K., Rijmen, V. (eds.) Selected Areas in Cryptography - SAC 2023 - 30th International Conference, Fredericton, Canada, August 14-18, 2023, Revised Selected Papers. Lecture Notes in Computer Science, vol. 14201, pp. 82–96. Springer (2023). https://doi.org/10.1007/978-3-031-53368-6_5
42. Yu, H., Shan, H., Panoff, M., Jin, Y.: Cross-device profiled side-channel attacks using meta-transfer learning. In: 58th ACM/IEEE Design Automation Conference, DAC 2021, San Francisco, CA, USA, December 5-9, 2021. pp. 703–708. IEEE (2021). https://doi.org/10.1109/DAC18074.2021.9586100
43. Zaid, G., Bossuet, L., Carbone, M., Habrard, A., Venelli, A.: Conditional variational autoencoder based on stochastic attacks. IACR Trans. Cryptogr. Hardw. Embed. Syst. **2023**(2), 310–357 (2023). https://doi.org/10.46586/tches.v2023.i2.310-357
44. Zaid, G., Bossuet, L., Habrard, A., Venelli, A.: Methodology for efficient cnn architectures in profiling attacks. IACR Transactions on Cryptographic Hardware and Embedded Systems **2020**(1), 1–36 (Nov 2019). https://doi.org/10.13154/tches.v2020.i1.1-36, https://tches.iacr.org/index.php/TCHES/article/view/8391

45. Zhu, J., Park, T., Isola, P., Efros, A.A.: 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. pp. 2242–2251. IEEE Computer Society (2017). https://doi.org/10.1109/ICCV.2017.244

A Additional Results from Section 4

A.1 ASCADf as the Reference Dataset

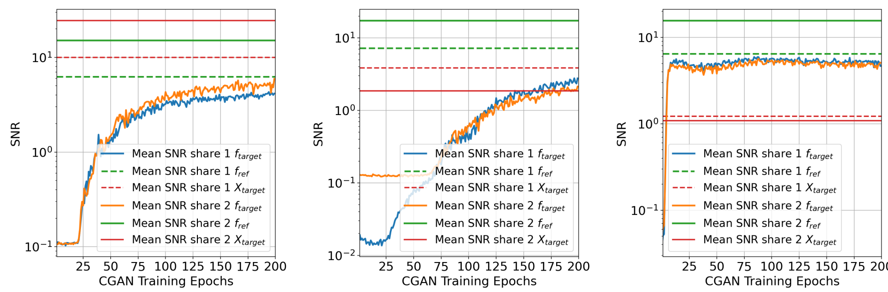
Table 8 lists the best-found CGAN architectures when ASCADf is set as the reference dataset and ASCADr, DPAv4.2, ESHARD-AES128, and CHES CTF 2018 are set as target datasets.

Table 8: Best CGAN hyperparameter for different target datasets when ASCADf is 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

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 can also extract high-SNR features from target datasets, as shown in Figure 9. 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.

The evolution of the SNR values from Figure 9 indicates that our hyperparameter search process also finds efficient CGAN architectures when ASCADf is considered as the reference. Some CGAN models require more epochs to converge due to the specific hyperparameters that were selected as the best models. Therefore, we concluded with this analysis that it is important to train the CGAN longer (i.e., from 100 to 200 epochs) to increase the chances of finding satisfactory results.



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

Fig. 9: Performance of CGAN architecture against different target datasets, X_{target}, when ASCADf is the reference dataset.

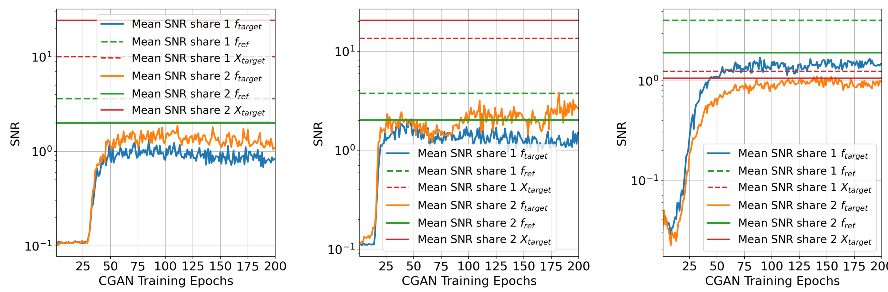
A.2 DPAv4.2 as the Reference Dataset

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 9.

Table 9: Best CGAN hyperparameter for different target datasets when DPAV4.2 is 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

Results provided in Figure 10 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 quickly for all target datasets, providing evidence that our proposed framework may not be very time-consuming.



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

Fig. 10: Performance of CGAN architecture against different target datasets, X_{target}, when DPAv4.2 is the reference dataset.

B Varying the Number of Features N_f

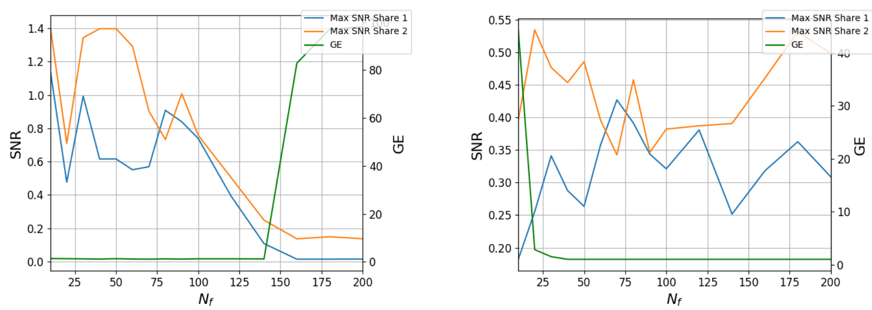
We analyze the performance of the CGAN-SCA framework for varying sizes of the feature space. We explore two scenarios: (1) ASCADr vs. DPAv42, in which the reference dataset (ASCADr) shows higher SNR peaks, with respect to the processing of two secret shares, than the target dataset (DPAv42) and (2) ESHARD-AES128 vs. ASCADf, in which the reference dataset (ESHARD-AES128) presents lower SNR peaks compared to the target dataset (ASCADf).

The CGAN architectures for this analysis come from the best-found models from Section 4.2 when ASCADr is the reference and DPAv4.2 is the target dataset. When ESHARD-AES128 is the reference dataset, we use a 4-layer MLP with 100 neurons in the first three layers and N_f in its last layer with linear or identity activation function. For the discriminator, we set one embedding dense layer and one dropout dense layer with 100 neurons. Subsequently, the attacks are performed with an MLP with 4 hidden layers with 100 neurons, the Adam optimizer with a learning rate of 0.001, using the elu activation function. We train this MLP for 50 epochs with a batch size of 400. We take the best results out of 10 CGAN training runs.

In Figure 11a, 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, which is the case when ASCADr is set as reference and DPAv4.2 as target dataset.

In Figure 11b, 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 11b is that successful attacks are possible 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 careful consideration is required here.



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

Fig. 11: Training results for different N_f. The SNR levels represent the maximum SNR value obtained from f_{target} with respect to each of the two processed secret shares.

C Simulations as the Reference Dataset

In this section, we assess whether the real reference implementation can be replaced by a simulation. As the only aspects of the reference implementation that are relevant for the training of our framework are the leaky features f_{ref}, it is relatively straightforward to implement basic simulations, as we only need to simulate leaky features for each secret share. We simulate using various leakage models to test whether simulations are a suitable replacement for an actual reference implementation.

C.1 Simulation Setup

In our tests, we always use $N_f = 100$, and we generate 200 000 simulated reference traces. The Gaussian noise that is added to the implementations is kept relatively low, and we additionally vary the level of noise for each feature. Specifically, for the i-th feature in f_{ref}, we have $Z_1^i = \mathcal{N}(0, Z_2^i)$, where $Z_2^i = \mathcal{U}(0.15, 0.5)$

(here $\mathcal{N}$ and $\mathcal{U}$ are the normal and uniform distributions respectively). This setup was based on some preliminary testing to allow for the CGAN models to actually learn, which was difficult for more basic simulations with the same level of noise for each feature.

To obtain a realistic view of the efficacy of replacing an actual reference implementation with simulations, we test various leakage models. In all of the below cases we simply simulate a first-order Boolean masked implementation by generating two random values $S_1, S_2 \in \mathbf{Z}_{256}$[11]. The labels are then generated by $Y_{ref} = S_1 \oplus S_2$. We include leakages for S_1 in the first 50 features of f_{ref} and leakages for S_2 in the last 50. Below, we define how we generate the 50 features for a single share. Here, $b_j(S)$ is the j-th bit of share S, and f_i is the i-th (or $(i+50)$-th) feature in f_{ref}.

- **Hamming weight (HW):** $f_i = \sum_{j=1}^{8} b_j(S) + Z_1^i$
- **Most Significant Bit (MSB):** $f_i = b_1(S) + Z_1^i$
- **Least Significant Bit (LSB):** $f_i = b_8(S) + Z_1^i$
- **Bit:** $f_i = b_j(S) + Z_1^i$ where $j = \frac{i \times 8}{50} \mod 8$. i.e., bit 1 is leaked in the first 6 features, bit 2 in the second 6, etc.
- **Real:** $f_i = \sum_{j \in bits_i} b_j(S) + Z_1^i$, where $bits_i$ is a list of 3-8 random integers between 1 and 8 representing the bits of the share to leak for f_i, i.e., for each feature we leak the Hamming weight of 3-8 randomly selected bits of the share.
- **4Bit:** $f_i = \sum_{j=1}^{4} b_j(S) + Z_1^i$ for $i < 25$ and $f_i = \sum_{j=5}^{8} b_j(S) + Z_2^i$ for $i \geq 25$ i.e., we leak the Hamming weight of the 4 most significant bits of the share S in the first 25 features of the trace, and the Hamming weight of the 4 least significant bits in the second 25 features.

C.2 Simulation Results

The hyperparameters of the CGAN models are tuned independently for each dataset/simulation pair over 100 training runs using the ranges and selection criteria described in Section 4.2. The subsequent attacks follow the CGAN-SCA attack setup described in Section 7, comprising a hyperparameter search of 100 models, using the ranges from Table 4.

In Table 10, the results for each tested scenario can be seen. Overall, we see that reasonable attacks can be achieved using a simulated reference dataset, which indicates the powerful capacity of the CGAN-SCA structure as a feature extraction process. Especially for DPAv4.2, CHES CTF 2018, and ASCADf, the attack results are similar to those given in Section 7 using real reference datasets. For the ESHARD-AES128 dataset, we see that the attacks are significantly worse than in Section 7. However, it seems likely that tuning the simulation parameters for this specific case will allow better attack performance as the simulations with the used parameters resemble the ASCAD and DPAv4.2 targets much more

[11] For DPAv4.2 $S_1 \in \{3, 12, 53, 58, 80, 95, 102, 105, 150, 153, 160, 175, 197, 202, 243, 252\}$ to simulate the RSM masking scheme used in this implementation.

closely in terms of SNR levels. We chose not to fine-tune simulations for each individual target to simplify experiments and to highlight the difference between an optimized simulation (ASCADf, DPAv4.2) vs. a non-optimized simulation (ESHARD-AES128).

Overall, simulating reference targets is clearly possible. However, constructing suitable simulations is non-trivial and dependent on the characteristics of the target device. Notably, in our experiments, we could easily create simulations with similar noise levels to (some of) the target implementations as we had access to the secret shares and could, therefore, determine the levels of leakage present in the targets. This knowledge is not available for real targets in the black-box setting. While the parameters for the simulation could be included in the random hyperparameter searches in a less informed setting, this would significantly increase the search space, making it harder to find good configurations. As such, including a real reference target (naturally) simplifies the attack process for the CGAN framework.

	HW	Bit	MSB	LSB	Real	4BIT
DPAv4.2	x	**6**	148	7	9	16
ASCADf	468	**1**	6	6	**1**	4
CHES CTF 2018	x	**20**	40	32	21	79
ESHARD-AES128	1924	**856**	x	1973	x	x

Table 10: Number of traces to reach GE=1 for various simulated leakage models.

D Unprotected Target

To verify the framework can also work for unprotected targets, we mount attacks against ASCADr where we assume masks are known (we label with $Sbox(p_i \oplus k_i) \oplus r_i$). We simulate the reference here, with the Bit leakage model from Appendix C. Note that we use 50 features here as there is only 1 relevant share. The architectures we use are the architectures found with ASCADf as reference and ASCADr as target. In this setup, the target key-byte can be retrieved in 1 trace.