Everything is Connected: From Model Learnability to Guessing Entropy

Lichao Wu[1], Léo Weissbart[1,2], Marina Krček[1], Huimin Li[1], Guilherme Perin[1], Lejla Batina[2] and Stjepan Picek[1]

[1] Delft University of Technology, The Netherlands
[2] Digital Security Group, Radboud University, The Netherlands

Abstract. Guessing entropy is a common choice for a side-channel analysis metric, and it represents the average rank position of a key candidate among all possible key guesses. In the profiled side-channel analysis, the guessing entropy behavior can be very informative about the trained or profiled model. However, to achieve reliable conclusions about the profiled model's performance, guessing entropy behavior should be stable to avoid misleading conclusions in the attack phase.
In this work, we investigate this problem of misleading conclusions from the entropy behavior, and we define two new concepts, *simple* and *generalized guessing entropy*. We demonstrate that the first one needs only a limited amount of attack traces but can lead to wrong interpretations about leakage detection. The second concept requires a large (sometimes unavailable) amount of attack traces, but it represents the optimal way of calculating guessing entropy. To quantify the profiled model's learnability, we first define a leakage distribution metric to estimate the underlying leakage model. This metric, together with the generalized guessing entropy results for all key candidates, can estimate the leakage learning or detection when a necessary amount of attack traces are available in the attack phase. By doing so, we provide a tight estimation of profiled side-channel analysis model learnability. We confirm our observations with a number of experimental results.

Keywords: Side-channel Analysis · Deep Learning · Guessing Entropy · Model Learnability

1 Introduction

Side-channel attacks (SCAs) are recognized as powerful attacks on implementations of cryptographic algorithms. Commonly, one divides side-channel attacks into direct attacks like Simple Power Analysis (SPA) and Differential Power Analysis (DPA) [KJJ99] and two-stage (profiled) attacks like template attack [CRR02], stochastic models [SLP05], and machine learning-based attacks [LPB+15, MPP16, PHJ+17]. Direct attacks have an advantage that they do not require access to an identical and open copy of the device under attack, but to break an implementation, such attacks might require tens of thousands of measurements (or more, depending on the level of protection deployed). On the other hand, two-stage attacks assume an "open" device (or a copy of it), but the actual key recovery stage requires only a few measurements or, in some cases, a single trace. In recent years, machine learning-based attacks posed themselves as a strong alternative for profiled SCA. Additionally, the rapid improvements in the deep learning domain also advanced SCA as deep learning methods showed to be capable of breaking even protected implementations [CDP17, KPH+19].

When evaluating the attack's performance, we require reliable metrics, and guessing entropy is commonly accepted as one of the most suitable ones for SCA. More precisely,

the level of convergence of guessing entropy for the correct key candidate indicates how successful an attack is [SMY09]. For profiled attacks to be successful, the number of attack traces in the test or attack phase to reach guessing entropy equal to 1[1] depends on the ability of the profiling model (i.e., template for template attack or machine learning model) to fit existing leakage and generalize to a separate attack set. Moreover, following the assumption that the profiling model can fit the existing leakage, issues like noise (from the environment of countermeasures) also affect the number of required attack traces. Therefore, it appears as a natural choice for the attacker to maximize the amount of processed attack traces, as the correct key guessing entropy convergence might only happen after processing sufficiently many traces.

While guessing entropy is (commonly) a reliable metric to indicate an attack's performance, we also observe it could behave deceptively (see Sections 4 and 6 for details). For instance, if all available attack traces are used for every key rank calculation, and the attack traces are only shuffled for each key rank, the attack can indicate unreliable generalization results resulting in the correct key candidate not converging (in a way that it becomes easily distinguishable from other key candidates). This situation applies especially to the case when the attack is unsuccessful. In this case, there will be several key candidates that will converge (guessing entropy decreases toward 1) or diverge (guessing entropy increases towards the maximum rank).

In this paper, we provide a comprehensive interpretation of guessing entropy in the profiled side-channel analysis. To this end, we define *simple guessing entropy* and *generalized guessing entropy*. The first obtains guessing entropy due to computing key rank multiple times with all available attack traces, where the attack traces are simply shuffled in order of processing for each key rank calculation. The generalized guessing entropy assumes that for each key rank calculation, the attacker randomly selects a portion of the attack traces to have a higher level of randomization and, consequently, an evident convergence of the guessing entropy. We also propose *Leakage Distribution Difference* as a metric to measure the ability of a profiled model to fit an existing leakage. This metric can distinguish a model that shows a preference for key candidates (that are similar to correct key) from a biased model that shows no, or limited, generalization. Finally, by evaluating the correlation between the Leakage Distribution Difference metric and the ranked vector or key guesses, we can properly estimate the profiling *model learnability*.

The main contributions of this paper are:

1. We discuss the types of behaviors of guessing entropy that we might get as attack results. Consequently, we define two new notions of guessing entropy that estimate better the attack performance in SCA. More specifically, we introduce simple guessing entropy and generalized guessing entropy. There, the former one is easier to compute, but it can also lead to misleading conclusions.
2. We propose a new metric, called Leakage Distribution Difference (LDD) that can be used to better understand the relationship between the correct key and guessed keys. This metric represents a fundamental step in building a new metric that enables estimating the profiling model's learnability.
3. We investigate the notion of profiling model learnability in SCA, and we propose a novel way to assess the model learnability. More precisely, we show that the Pearson correlation between LDD and key guessing vector can reliably indicate an attack's performance. This is a first result showing how to estimate the model's learnability in profiled SCA to the best of our knowledge.

We provide extensive experimental results to validate our claims, spanning different datasets, leakage models, and profiling methods.

[1] Guessing entropy equals 1 means that the correct key is ranked as the best key candidate in the attack phase

2 Background

2.1 Notation

Let b be the number of bits in the target cryptographic state, k is a key byte candidate that takes its value from the key space K, and k^* is the correct key byte. A dataset is defined as a collection of a traceset T, where each trace t_i is associated with a plaintext byte pk_i and a key k_i. The number of profiling traces equals N, and the number of attack traces equals Q. When investigating the attack performance, we will use either the original datasets or datasets with added noise. The notation $\mathcal{N}(\mu, \sigma)$ is the level of Gaussian noise added to the traces, where μ is mean, and σ is the standard deviation of the noise distribution. Finally, θ denotes the parameters to be learned in a profiling model, while λ denotes the set of hyperparameters defining the learning model. In this paper, bold letters or bold acronyms, e.g., $\mathbf{a}$ and $\mathbf{AB}$, define vectors of size equal to 2^b.

2.2 Profiled Side-channel Analysis

A profiled side-channel attack is a category of side-channel analysis methods composed mainly of template attacks, machine learning attacks, or deep learning attacks. These techniques depend on sets of parameters θ. When correctly tuned, they produce a model that can predict with a high probability the key associated with a leakage trace. Profiled side-channel attacks consist of two phases:

1. **Learning or profiling phase.** The profiling phase consists of training the set of parameters θ of a model with a dataset of N profiling traces labeled with the leakage value of the keys used to compute the cipher texts of all traces with their corresponding plain texts. The goal is to fit the parameters of a function that maps the traces to the labels of the dataset in the best way. A model's learnability is the capacity of a model's parameters θ to fit the leakage and predict the associated labels of a trace correctly. In template attack and machine learning-based attacks, the learnability is characterized by the points of interest (POIs) selection (as to avoid the curse of dimensionality) and hyperparameters of the adopted algorithm (SVM, random forest, Naive Bayes, etc.). In deep neural networks-based attacks, the learnability depends on selecting the hyperparameters λ, as the neural network implicitly performs the points of interest (or feature) selection.
2. **Test or attack phase.** The attack phase consists of obtaining label predictions for the traces of a different dataset of Q attack traces to test the model. The trained model processes each separate attack trace and predicts with a vector of individual probabilities $p_{i,j}$ that a trace i is associated with the leakage value j linked to the key. An attack's output is the logarithmic sum of all Q probability vectors of single model predictions, where each index is associated with one key hypothesis. Sorting this probability vector by decreasing probabilities leads to a vector of key guesses ranked by increasing prediction confidence of the model. The key rank denotes the position of the correct key. Then, one can use notions of guessing entropy and success rate to estimate the attacker's performance [SMY09].

Definition 1. Key guessing vector. Let us assume that given Q amount of traces in the attacking phase, an attack outputs a key guessing vector $\mathbf{g} = [g_1, g_2, \ldots, g_{|\mathcal{K}|}]$ in decreasing order of probability with $|\mathcal{K}|$ being the size of the key space.

Definition 2. Guessing entropy. The guessing entropy represents the average position of k^* in the key guessing vector ($\mathbf{g}$).

2.3 Supervised Machine Learning

Supervised learning requires learning a mapping between a set of input variables X and an output variable Y. This represents the learning phase. Once this mapping is learned, it is necessary to apply it to predict unseen data outputs. This represents the testing phase. Thus, it is clear that supervised learning can be directly connected with profiled SCAs. Note that in this paper, we consider only the classification task as the goal of the side-channel attack. To make this connection even more explicit, we require several machine learning concepts we define next.

Definition 3. Model learnability. A model is learnable if there exists a polynomial-time algorithm that achieves low error with high confidence for all examples.

Definition 4. Model generalization. It represents the model's ability to adapt properly to new, previously unseen data, drawn from the same distribution used to create the model.

Definition 5. Overfitting. It represents the phenomenon when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on test data.

Definition 6. Bias of a model. It represents the error from wrong assumptions in the learning algorithm.

2.4 Classifiers

We investigate the performance of three common profiled attack methods: template attack, multilayer perceptron, and convolutional neural networks.

Template Attack. Template attack (TA) uses Bayes theorem to obtain predictions, dealing with multivariate probability distributions as the leakage over consecutive time samples is not independent [CRR03]. In the state-of-the-art, template attack relies mostly on a normal distribution. It consists of two phases: the offline phase during which the templates are built, and the online phase where the matching between the templates and unseen power leakage happens.

Multilayer Perceptron. The multilayer perceptron (MLP) is a feed-forward neural network that maps sets of inputs onto sets of appropriate outputs [GD98]. MLP consists of multiple layers of nodes in a directed graph, where each layer is fully connected to the next one, and training of the network is done with the backpropagation algorithm. There are at least three layers: one input layer, one output layer, and one hidden layer. If there is more than one hidden layer, then such an architecture already represents deep learning.

Convolutional Neural Networks. Convolutional neural networks (CNNs) commonly consist of three types of layers: convolutional layers, pooling layers, and fully-connected layers. Convolution layer computes the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. Pooling decreases the number of extracted features by performing a down-sampling operation along the spatial dimensions. The fully-connected layer (the same as in MLP) computes either the hidden activations or the class scores. The batch normalization layer normalizes the input layer by adjusting and scaling the activations.

2.5 ASCAD Dataset

The ASCAD dataset contains 200 000 traces for profiling, with random keys and random plaintexts, and 100 000 for the attack phase, with fixed key and random plaintexts. Side-channel traces in this dataset represent the AES encryption, where the attacked trace interval represents the processing of byte 3 in the S-box operation presented in the first round. A window of 1 400 points of interest is extracted around the leaking spot. The operation is masked, and in our analysis, no knowledge about masks is assumed in the profiling phase.

3 Related Works

When considering profiled side-channel attacks, we divide related works into three directions: 1) classical profiled attacks, 2) machine learning-based attacks, and 3) deep learning-based attacks. The first two directions mainly oriented toward improving the attack performance by selecting discriminating features or strong machine learning models. The third directions extended this by considering topics like explainability, interpretability, data augmentation, the difference between SCA and machine learning metrics, etc.

3.1 Classical Profiled Attacks

In the seminal work of Chari et al., the authors developed the template attack and showed it could break implementations secure against other forms of side-channel attacks [CRR03]. This attack is the most powerful one from the information-theoretic point of view, but to reach its full power, it requires an unbounded number of traces, and that the noise follows the Gaussian distribution [LPB+15]. Instead of using one covariance matrix for each label y, the authors of [CK14] proposed to use one pooled covariance matrix averaged over all labels to cope with statistical difficulties. Finally, the third type of attack called stochastic attack utilizes linear regression instead of probability density estimation (as used in TA) [SLP05].

3.2 Machine Learning-based Attacks

While machine learning techniques are widely used for several decades, the SCA community showed interest in such techniques one decade ago. First, the most interest sparked techniques like random forest [LMBM13, HPGM16] and support vector machines [HZ12, PHJ+17]. Besides those techniques, a multilayer perceptron also showed significant potential [GHO15], but there, the boundary between machine learning and deep learning is often not very clear as the first works do not report precisely the neural network sizes.

3.3 Deep Learning-based Attacks

Today, deep learning represents a common direction for profiled SCA, where several research directions can be followed. Note that we do not aim to provide an extensive overview, but rather, a brief discussion on selected works. For a detailed survey of machine learning in profiled attacks, we refer interested readers to [HGG20a].

Finding Strong Machine Learning Models. The rapid development of deep learning-based SCAs started in 2016 when Maghrebi et al. demonstrated the strong performance of several neural network types, most notably, convolutional neural networks [MPP16]. Deep neural networks configurations are difficult to tune. The good performance of multilayer perceptrons or convolutional neural networks relies on an efficient selection of hyperparameters for specific datasets. In [ZBHV19], the authors proposed a methodology

to select hyperparameters that are related to the size (number of learnable parameters, i.e., weights and biases) of layers in CNNs. This includes the number of filters, kernel sizes, strides, and the number of neurons in fully-connected layers. In [BPS+20], an empirical evaluation for different hyperparameters is conducted for CNNs on the ASCAD database. Still, to the best of our knowledge, there are no publications providing solutions for hyperparameters optimization algorithms for the context of side-channel analysis. Kim et al. investigated how adding noise to the input (thus, serving as regularization) improves the performance of profiled SCAs [KPH+19]. Depending on the number of profiling traces, the number of features (or sample points), and the protection level of the target device (including masking and hiding countermeasures), the number of the machine or deep learning models that can be tested is easily limited by computation power or time budget. In this case, the security evaluator or attacker has two options: 1) train the maximum possible amount of models within the available resources, or 2) train a limited amount of models by using optimization methods. Nevertheless, the best leakage model selection also influences the performance of the deep learning model [PCP19].

Not Enough Measurements. It is intuitive that the number of measurements also limits the performance of a profiled attack. Deep neural networks are known to provide top-level performances in many domains when the amount of training data is sufficiently large. However, it could also provide remarkable performance when the amount of training data is reduced. In the context of profile side-channel attacks, Cagli et al. investigated how to create measurements that improve the attack performance synthetically [CDP17]. Differing from the previous work where the authors developed a specialized data augmentation technique, Picek et al. showed that generic data augmentation techniques help in profiled SCA also [PHJ+18]. Researchers also investigated whether limiting the number of measurements can be beneficial, both from the experimental setup and performance sides [PHPG19].

Differences between SCA and Machine Learning Metrics. Commonly, in machine learning, one estimates the behavior of a profiling model based on statistics of individual observations like accuracy, loss, or recall. Unfortunately, such metrics can be misleading in SCA, as one considers cumulative predictions. Picek et al. showed that common machine learning metrics could suggest radically different performance than the SCA metrics [PHJ+18]. Masure et al. connected the perceived information and negative log-likelihood, which shows there can be common ground when using machine learning metrics in SCA [MDP19b]. Finally, Perin et al. discussed how mutual information could be a good metric to indicate when to stop the machine learning training process [PBP20].

Interpretability and Explainability. Interpretability is the degree to which a human can consistently predict the model's result [KKK16]. Explainability is the extent to which a machine learning system's internal mechanics can be explained in human terms. Several works consider visualization techniques to find the relevant features and improve the interpretability [HGG20b, MDP19a]. Van der Valk and Picek extended guessing entropy to guessing entropy bias-variance decomposition to improve the interpretability of results from machine learning-based attacks [vdVP19]. Van der Valk et al. considered the activation functions in neural networks to interpret what neural networks learn while training on different side-channel datasets [vdVPB19].

4 On Possible Guessing Entropy Behaviors

Recall, guessing entropy is the average rank of the correct key k^* in a key guessing vector $\mathbf{g}$ after processing Q attack traces. Commonly, guessing entropy is computed for every

byte of the key to make predictions.[2] This resulting prediction can be used to determine the remaining brute-force key enumeration effort after finishing the attack.

From a practical perspective, the number of attack traces Q is always limited and defined according to side-channel measurements' availability. When Q is not sufficiently large, a natural choice for attackers or security evaluators is to maximize the amount of attack traces in each key rank calculation used to obtain the guessing entropy for a certain key guess k. This leads to what we define as *simple guessing entropy*. In this case, the final key rank value obtained after the processing of Q attack traces, for all key candidates k, is the same in each key rank calculation.

Definition 7. Simple Guessing Entropy (SGE). Given Q attack traces, the simple guessing entropy results from the average computation of multiple attacks, where each calculation considers the full set of available Q attack traces with shuffling.

Following Definition 1, $\mathbf{g}_{sge}$ represents the key guessing vector obtained from the guessing entropy computed according to Definition 7.

Simple guessing entropy is commonly found in open source codes released with recent publications [ZBHV19, BPS+20] [3] [4]. Several behaviors can be observed from simple guessing entropy results for a single key byte, including:

- GE can continuously **decrease** with increasing the number of attack traces. It is valid to conclude that increasing the number of attack traces will lead to successful key recovery with the trained model as the final guessing entropy value can be very low, shown as the blue line in Figure 1.
- GE can continuously **increase** with increasing the number of attack traces. It is valid to assume that adding more traces would only increase GE toward the worst rank equal to $2^b - 1$, shown as the red line in Figure 1. The GE value can also stabilize for some specific value, but it does not decrease with adding more attack traces.
- GE can **stay close to** $(2^b - 1)/2$, i.e., it behaves like random guessing, shown as the green line in Figure 1. Then, we cannot determine GE's behavior or evolution by adding more attack traces, as the increase or decrease of GE could only be confirmed by processing a very large amount of attack traces that may be unavailable.

We provide more examples of these behaviors in Appendix A.

Intuitively, such simple GE behaviors lead to commonly accepted interpretations about the profiling model performance. When GE increases or stays random, the model is wrong because it does not learn side-channel leakages related to the correct key. On the other hand, when GE decreases, the model is trained correctly and can fit the existing leakage. Adding more attack traces will lead to GE indicating a correct key candidate ranked among the first ones. Such interpretations of the GE behavior (when computed according to Definition 7) could lead to inaccurate estimation about the profiled attack performance. To have consistent conclusions about GE and attack performance, we need to investigate a *generalized way* to compute guessing entropy, and what is possible to conclude when this generalized way cannot be done. Consequently, we present the definition of generalized guessing entropy, as follows:

Definition 8. Generalized Guessing Entropy (GGE). Given Q attack traces, the generalized guessing entropy results from the average computation of multiple attacks, where each key rank calculation considers a randomly selected subset U of Q traces, defined as $U = Q(1 - r)$ traces.

[2] If guessing entropy is calculated for only certain key bytes, it is partial guessing entropy. Nevertheless, we use the two terms interchangeably.

[3] https://github.com/gabzai/Methodology-for-efficient-CNN-architectures-in-SCA

[4] https://github.com/ANSSI-FR/ASCAD

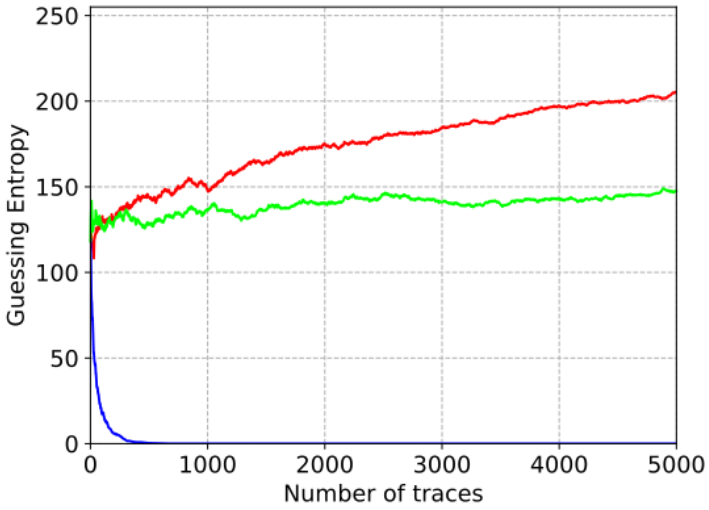


Figure 1: All defined GE behaviors are seen with the architecture from [KPH+19] and the ID leakage model on the synchronized ASCAD dataset with the fixed key without any change in the hyperparameters.

Additionally, we introduce the notion of the randomization factor $r = 1 - U/Q$, for $U \leq Q$ and $0 \leq r < 1$, which defines the level of randomization of attack traces for each separate key rank calculation. The higher the randomization factor r, the higher the generalization level represented by generalized guessing entropy. Following Definition 1, $\mathbf{g}_{gge}$ represents the key guessing vector obtained from the guessing entropy computed according to Definition 8.

Considering the full set of Q attack traces, the shuffling and averaging for SGE does not bring any statistical improvement as one uses all the traces for every key rank calculation for the guessing entropy evaluation. On the other hand, for GGE, the traces selected for each key rank calculation always represent only a smaller subset of the attack trace set (thus with different noise distribution), increasing the GE estimation's reproducibility with higher randomization factor r.

For GGE to provide an accurate estimation of the profiled model's generalization to different attack sets, the number of attack traces must be larger than for a simple analysis. When the number of attack traces is too few to compute GGE, it is still possible to evaluate if a model can be successful. Consequently, we propose in Section 5 a metric to measure the model's learnability of a side-channel leakage.

5 SCA Model Learnability and Estimation

After training a profiling model, it is essential to determine whether the model learned to fit data, i.e., estimate the model learnability. Commonly, this is done by evaluating the performance on the attack dataset. If the model fitted data (i.e., learned from the actual leakage information), its bias is low, and we expect that, by adding more attack traces, an SCA metric indicates improved attack performance. Even if the model makes a wrong key guess, one will hope for some relationship between the guessed key and the correct key. On the other hand, if the model fitted noise, it will either behave as random guessing or point toward wrong key candidates. Then, one would assume there is little or no relationship between the guessed and the correct key.

5.1 Leakage Distribution Difference Metric

We introduce a metric to quantify the relationship between the guessed key candidate k and the correct key k^*. First, to better evaluate the profiling model's preference (i.e., the mapping decision) for specific key candidates, we calculate a hypothetical leakage distribution for every key candidate and all plain texts of a given dataset. We denote this distribution as the leakage distribution of a dataset. The leakage distribution variation between different key candidates represents the Leakage Distribution Difference metric, denoted as **LDD**. **LDD** provides an estimation of the hypothetical label distribution variation between the different key candidates. A specific key will then have a smaller probability to be selected based on a (properly) trained model if **LDD** is large between that key and the correct key.

Eq. (1) calculates the sum of the squared difference between the leakage distribution of all key hypotheses $k \in K$ and the correct key candidate k^* over Q attack traces. Since all dimensions are comparable and we do not require high-dimensional spaces, we use the Euclidean distance (L_2 norm). The larger leakage difference between the two input values will introduce more significant **LDD** variation. Note, **LDD** is a vector of dimension 2^b, one for each key candidate k that is computed as follows:

$$\mathbf{LDD}(k^*, k) = \sum_{i=0}^{Q} \left\| f(d_i, k^*) - f(d_i, k) \right\|^2, k \in K. \tag{1}$$

In the above expression, $f(d_i, k)$ is the leakage model function that returns the leakage value according to a key candidate k and data value $d_i \in Q$, where Q denotes the number of attack traces in the dataset. **LDD** is computed for every key candidate, and it gives a unique distribution of all key candidates based on their difference to the correct key while being equal to zero for the correct key.

For instance, if the leakage function relies on the Hamming weight of a target byte in S-box output, we define the *Hamming Weight Distribution Difference* (**HWDD**) for the correct key candidate k^* and a key candidate k as:

$$\mathbf{HWDD}(k^*, k) = \sum_{i=0}^{Q} \left\| HW(S_{box}(p_i \oplus k^*)) - HW(S_{box}(p_i \oplus k)) \right\|^2. \tag{2}$$

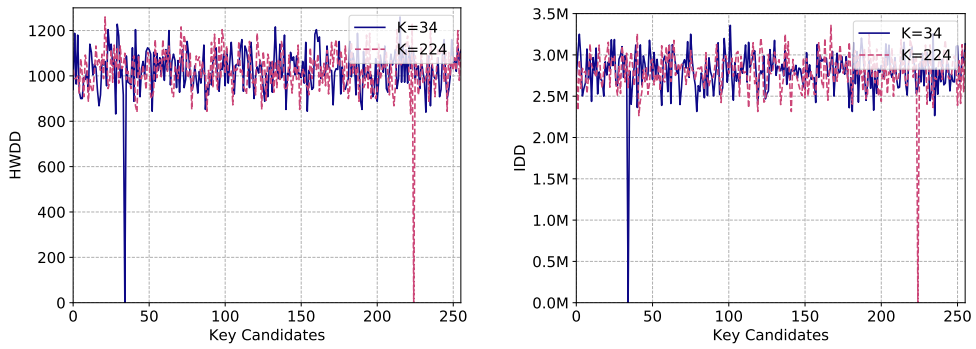
As we can observe, the leakage function in Eq. (2) is set to $HW(S_{box}(p_i \oplus k))$, where $\oplus$ is the exclusive OR operation. Similarly, we could also define the *Identity Leakage Distribution* (**IDD**) where the target state is the S-box output. In this case, the leakage function equals $S_{box}(p_i \oplus k)$ and the **IDD** value for the correct key candidate k^* and a key candidate k is:

$$\mathbf{IDD}(k^*, k) = \sum_{i=0}^{Q} \left\| S_{box}(p_i \oplus k^*) - S_{box}(p_i \oplus k) \right\|^2. \tag{3}$$

When clear from the context, we use the notations $\mathbf{LDD}(k^*, k)$ and **LDD** interchangeably. The **LDD** definition can be extended to any leakage model, e.g., MSB or LSB. This paper restricts the analysis to the Hamming weight and identity leakage models for the AES cipher. [5]

Figure 2 illustrates **HWDD** and **IDD** for the correct key candidates $k^* = 34$ (correct key for the ASCAD dataset with random keys) and $k^* = 224$ (correct key for the ASCAD

[5] The publicly available datasets consider AES and leak mostly in those leakage models. Actually, they leak mostly in the Hamming weight leakage model, and that leakage model will produce the best results. Still, the ASCAD dataset also leaks in the identity leakage model, enabling us to investigate the **IDD** scenario.



(a) **HWDD**. (b) **IDD**.

Figure 2: Illustration for the **HWDD** and **IDD** for key candidates 34 and 224.

dataset with a fixed key). Note that the leakage distribution for each key candidate is unique. The selection of the reference key, therefore, determines the **LDD** value for each key candidate. For instance, the correct key candidate has a distribution difference equal to zero. For the remaining key byte candidates, the lower the distribution difference, the more similar are the key bytes to the correct key candidate.

5.2 Case Study: Attack on a "Noise-free" Dataset

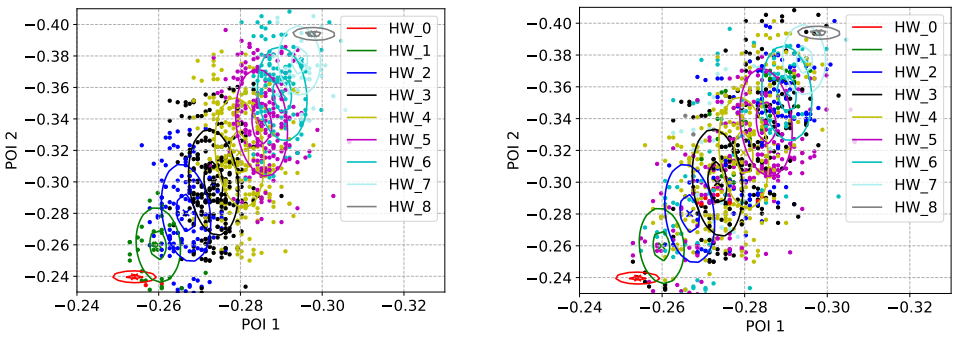
Next, we discuss the most-likely key (ranked as the first in $\mathbf{g}$) and least-likely key (ranked as the last in $\mathbf{g}$) guesses and their relationship to the correct key. We use a ChipWhisperer dataset as the correct key can be retrieved within ten attack traces with a template attack (TA) [OC14]. [6] Additionally, only two points of interest (we denote them as POI1 and POI2) are required to attack this dataset successfully. Consequently, the probability density function (PDF) for each cluster, which is equivalent to the templates built during the profiling phase, can be represented by a 2D image. By visualizing the distribution of different cluster's PDFs, we can better understand the **LDD** metric.

Figures 3a and 3b show PDFs of 1 000 POIs' distribution for most-likely and least-likely key candidates for TA. Note that, in this case, the most-likely key is also the correct key. Since we use the HW leakage model, nine PDFs representing nine HW clusters are built during the profiling phase. Each PDF is represented by two contour lines that denote 0.5 (outer) and 0.9 (inner) of the maximum probabilities. The color of each point is attributed based on their cluster label. The distribution of the POI pairs for the same label is denoted as HW-POI distribution.

From the results, we observe 1) POI1 and POI2 are strongly correlated, 2) the HW-POI distribution for the most-likely (correct) key matches better to its labeled cluster than for the least-likely key, and 3) the distribution of each PDF is separable. Based on these observations, we could conclude that the HW variation is strongly correlated with the PDFs' mean differences. In other words, the mean difference between PDFs and the variation of the corresponding HW values are similar.

We extend the single HW difference to the HW distribution difference (**HWDD**). Note that the only difference is that instead of using a single plain text, we use different plain texts from the attack traces to calculate the HW differences for each key candidate. Since the HW distribution of the most-likely key matches the PDFs distribution of the most-likely key the best, calculating the **HWDD** between the most-likely key and another

[6] Note that this dataset is not perfectly noiseless, but without resorting to simulations, it is difficult to obtain less noisy measurements.



(a) PDFs and HW-POI distribution for the most-likely key.

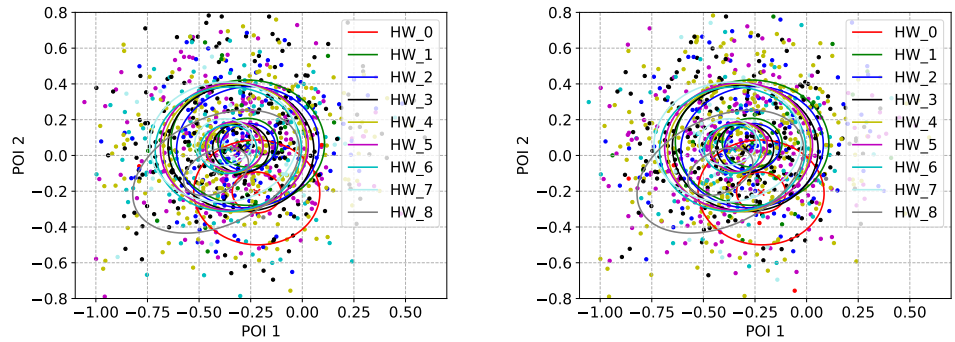
(b) PDFs and HW-POI distribution for the least-likely key

Figure 3: PDFs and HW-POI distribution with different keys without noise.

key candidate k gives a greater **HWDD** deviation, indicating that the HW distribution of k is less fitted to the PDF (also see second observation as an extreme case). Consequently, one could expect a low key rank for k.

Following this, the link between **HWDD** and key guessing vector ($\mathbf{g}$) can be obtained. Indeed, the Pearson's correlation between **HWDD** rank and $\mathbf{g}$ based on the PDFs shown in Figure 3 reaches 0.86, indicating that these two factors are highly correlated. Note that the third observation implies that the PDFs tend to output high probability with certain HW-POI distribution.

However, when noise is introduced, the PDF output is less determined by the leakage but by the random fluctuation of the POIs value (i.e., from the noise), which eventually obscures the output probability of different clusters. To demonstrate this, we introduce Gaussian noise with zero mean and 0.3 variances to the original dataset. Since the deviation between the most- and least-likely keys is discussed before, we present the PDFs and HW-POI distribution for the correct key and most-likely (now, the most-likely key is not the correct one, but simply the first ranked key in $\mathbf{g}$) key. The results are shown in Figures 4a and 4b.



(a) PDFs and HW-POI distribution for the correct key.

(b) PDFs and HW-POI distribution for the more-likely key

Figure 4: PDFs and HW-POI distribution with different keys with noise.

From the figures, the PDFs built with noisy data are similar to the noise distribution added to the traces (thus, SNR becomes very low). Consequently, the overlapping area for

each PDF is increased, and the difference in the output probability for each PDF becomes smaller. Similarly, when comparing the HW-POI distribution in Figures 4a and 4b, one can hardly identify which key (correct or most-likely) fits the PDFs better. Indeed, from the attack perspective, the correct key's rank reaches 4 after applying more than 1 000 attack traces, indicating that the profiling model's learnability is weaker than for the original dataset (reaches 1 within ten attack traces). Moreover, the random addition of the noise could lead to different most-likely keys, even with a fixed amount of attack traces. Although there is always an HW-POI distribution key that fits the PDFs/templates the best, the PDFs' distribution variation manipulated by the random noise makes the most-likely key unpredictable. Eventually, one could expect a reduced correlation between **HWDD** and $\mathbf{g}$, indicating the profiling model's low learnability.

5.3 Correlation between LDD and Key Guessing Vector g

In terms of guessing entropy for each key candidate, a larger **LDD** value indicates that two related key candidates are less likely to have similar GE. In other words, the **LDD** could estimate the GE distribution for each key candidate. One could expect a stronger statistical relation between **LDD** and $\mathbf{g}$ if the model learns from the leakage. In case the model fails to learn from the data, the outputted random GE for all key candidates would lead to a low correlation between **LDD** and $\mathbf{g}$.

Following this, the leakage distribution difference (given by any selected leakage model, e.g., **HWDD** or **IDD**) can be used to define a *learnability metric*, $L_m(\mathbf{LDD}, \mathbf{g})$, as a function of **LDD** and the key guessing vector $\mathbf{g}$:

$$L_m(\mathbf{LDD}, \mathbf{g}) = corr(\text{argsort}\,\mathbf{LDD}, \mathbf{g}) \tag{4}$$

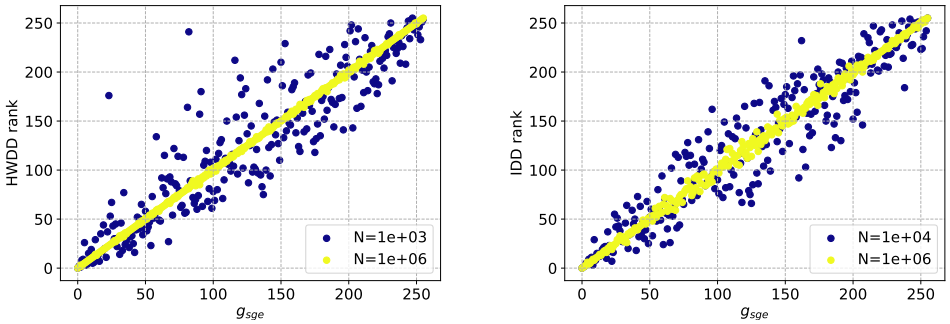
Eq. (4) defines the learnability level of a profiled model with respect to a key candidate k^* for a chosen leakage model. The term *corr* refers to Pearson's correlation. The expression $\text{argsort}\,\mathbf{X}$ returns the rank position by order of magnitude of each element x_i in a vector $\mathbf{X} = [x_0, x_1, \ldots, x_{2^b-1}]$. As a result, the $L_m(\mathbf{LDD}, \mathbf{g})$ metric can sort the key candidates according to their leakage difference from the correct key candidate k^*. Consequently, we can use this metric to determine how correlated the guessed keys are to the correct key. What is more, the **LDD** metric allows us to estimate the learnability of a profiled SCA model.

Definition 9. SCA model perfect learnability. There exists a polynomial-time algorithm that, in the attack phase, reaches $L_m = 1$ (i.e., perfect learnability) for all sets of attack traces.

Figure 5 depicts the perfect learnability for the HW and ID leakage models. We use simulated measurements with strong HW and ID leakages and a controlled Gaussian noise level, normally distributed with a variance of 0.01 around a mean of zero. The simulated trace set consist of traces of one hundred features where all features are equal but for two features that hold the leakage, which is proportional to $HW(S_{box}(p, k))$ or $S_{box}(p, k)$, respectively, for the HW and ID leakages. The profiling set has plain texts p and keys k chosen from a uniformly random distribution. The attack set's plain texts are also selected uniformly at random, and only the attack key is the same for the whole set.

We use the template attack to obtain the $\mathbf{g}_{sge}$. We consider an increasing number of profiling traces N to reduce the noise in the templates. In both figures, the correlation between **LDD** and the the key guessing vector $\mathbf{g}_{sge}$ (i.e., learnability metric) increases w.r.t. the number of profiling traces and reaches the Pearson's correlation of 0.999 and 0.998 for the HW and ID leakage models using one million profiling traces.

Remark 1. Results in Figure 5 confirm the correctness of Definition 9 as the correlation between **LDD** and $\mathbf{g}_{sge}$ tends to increase with better models (as we use template attack, better models are those that are trained with more traces).



(a) HW leakage model.

(b) ID leakage model.

Figure 5: Perfect learnability considering the HW and ID leakage models. Gaussian noise $\mathcal{N}(0, 0.01)$ is applied on the training traces. Increasing the number of training traces (N) reduces the templates noise.

6 Experimental Evaluation

We discussed how guessing entropy calculations could lead to wrong indications about the model learnability. We also discussed how the correlation between the $\mathbf{g}$ and $\mathbf{LDD}$ metrics could serve as a reliable indicator of model learnability. This section presents experimental results on publicly available datasets that confirm the LDD metric performance. First, we discuss how we can estimate the learnability from $\mathbf{g}_{gge}$ or $\mathbf{g}_{sge}$. Finally, we show the robustness of $L_m(\mathbf{LDD}, \mathbf{g})$ by evaluating it for different profiling models or levels of noise.

6.1 Estimating Learnability with Generalized Guessing Entropy (GGE)

We assume that GGE is the optimal way of computing guessing entropy as it considers a sufficient (where sufficient means that the available number of traces is much larger than the number of traces we require to break the target) number of attack traces. While GGE can be infeasible to obtain (as only a limited amount of attack traces could be available in the attack phase and, consequently, a randomization factor close to 1 would be difficult to obtain), simple guessing entropy (SGE) calculation may point to the wrong conclusions about the attack phase, mostly due to the lack of the SGE generalization ability.

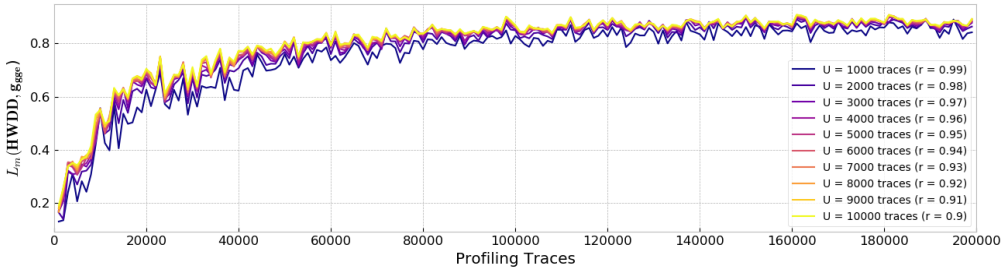
To minimize this issue from an evaluator's perspective, it is important to understand the minimal necessary number of attack traces to properly estimate the model learnability. Unfortunately, this is a difficult task as it depends on 1) selection of a good attack method, 2) usage of the correct leakage model, and 3) characteristics of the dataset (noise, countermeasures). An alternative is to calculate the $L_m(\mathbf{LDD}, \mathbf{g}_{gge})$ metric for the chosen leakage model. [7]

As an example, let us consider the ASCAD dataset (details in Section 2.5, and quantities $U = 1\,000$ and $Q = 100\,000$ (so that the randomization factor equals $r = 0.99$), that are sufficient values for a proper GGE estimation. Figure 6a shows the relationship between $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ (we consider the Hamming weight leakage model) and the number of profiling trace. Figure 6b shows the values of GGE evolution, for the correct key candidate k^*, and different number of profiling traces. These figures contain 200 profiling model results, where we vary the profiling set size from 1 000 to 200 000 traces with a step of 1 000 traces. The profiling model is an MLP with six hidden layers, where each layer contains

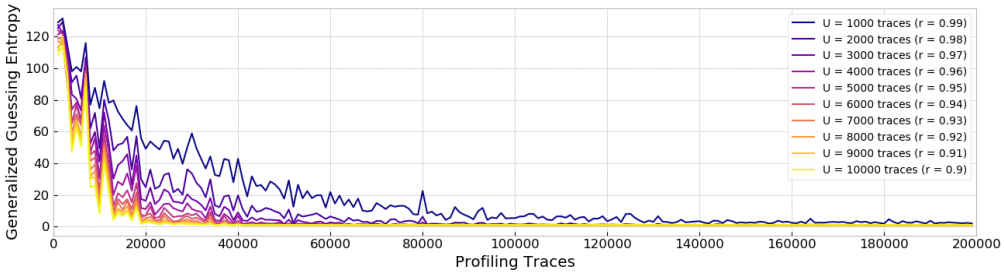
[7] If we had sufficient amount of attack traces to compute GGE, we would not need to obtain the $L_m(\mathbf{LDD}, \mathbf{g}_{gge})$ value, as "enough" randomness in the GGE calculation would be sufficient to indicate the profiled model's performance and generalization.

600 neurons. For this neural network architecture, the learning rate is set to 0.00001 with the *Adam* optimizer. The models are trained for 50 epochs with a mini-batch of 400 traces.

In Figure 6, observe that changing the value U (and, consequently, the randomization factor r) changes the final guessing entropy obtained for a model trained on a certain amount of profiling traces. The differences in the final guessing entropy results for different U values are more significant when the number of profiling traces is lower. For example, when MLP is trained with 20 000 traces, GGE when $U = 1\,000$ is around 50, while GGE for the same model after processing $U = 10\,000$ is already 1. Note that this is expected, as, for a low number of attack traces, the guessing entropy might not reach its lowest possible value for a certain profiled model. Simultaneously, the $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ metric varies marginally for different quantities of U, even for smaller amounts of profiling traces. *This analysis depicts that if Q is sufficiently large (100 000 in the current example), the behavior of generalized guessing entropy with a small U can be considered as a reliable value.* This conclusion can be confirmed by observing Figure 6a, where the value $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ is very similar for different amounts of U attack traces even when the number of profiling traces is low.



(a) $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ vs number of profiling traces.



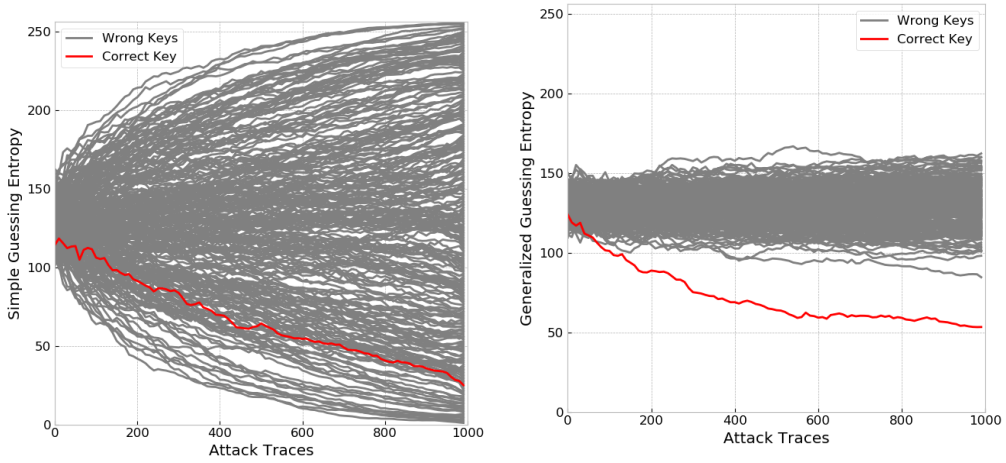
(b) GGE vs number of profiling traces.

Figure 6: $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ and GGE with respect to different amounts of attack traces.

Remark 2. Generalized guessing entropy is a reliable attack metric, but depending on the setup, it may be prohibitively difficult to obtain enough attack traces.

6.2 Estimating Learnability from Simple Guessing Entropy (SGE)

Next, we demonstrate that computing guessing entropy using Definition 7 for a limited amount of attack traces could lead to wrong conclusions about model learnability.Recall, the trained model tends to demonstrate a preference for some key byte candidates, and the reliability of such conclusions can also be estimated from the way SGE is calculated. If the model learns the leakage (i.e., the model learned the correct mapping between X and Y), then the preference happens for the correct key candidate in the attack set. On the other hand, if the preference happens for the wrong key candidate, we can assume that the trained model has no leakage learnability.



(a) SGE computed with $U = Q = 1\,000$

(b) GGE computed with $U = 1\,000$ and $Q = 100\,000$

Figure 7: The effects of simple and generalized guessing entropy for the same profiled model.

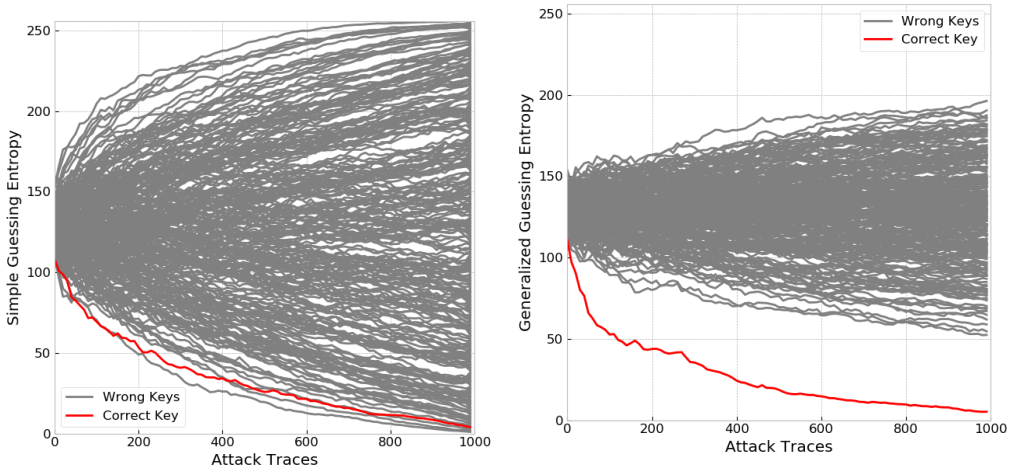
Simple guessing entropy may be insufficient to indicate model learnability when the number of attack traces Q is not large enough. In this case, it is very likely that GE would be computed for $U = Q$ (with a randomization factor $r = 0$). More precisely, we calculate simple guessing entropy, as the attacker would use the maximum amount of available attack traces for the key rank calculation. Therefore, to avoid misleading conclusions about model generalization, it is very important to identify whether SGE shows biased results.

We train the same MLP architecture with six hidden layers, as described in Section 6.1 on 20 000 profiling traces. Next, we compute the simple guessing entropy ($U = Q = 1\,000$) and obtain the results shown in Figure 7a. This figure shows SGE evolution for all 256 key byte candidates. As this figure indicates, even if guessing entropy for the correct key candidate slowly converges, other key candidates show better (faster) convergence. As a result, we could conclude that this profiled attack is not successful. On the other hand, if we compute the generalized guessing entropy with $U = 1\,000$ and $Q = 100\,000$ (cf. Figure 7b), guessing entropy after processing 1 000 attack traces already indicates that this profiled attack is actually successful. In the latter case, the guessing entropy convergence is more pronounced for the correct key candidate than all wrong key candidates.

Increasing the amount of attack traces to $U = Q = 10\,000$ provides more reliable results in the simple guessing entropy calculation. However, the results for SGE still differ from GGE results, as indicated in Figure 8. There are two basic conclusions stemming from these results:

1. SGE for a *small* $U = Q$ (e.g., 1 000 in case of the ASCAD dataset) is not a reliable indication of model's learnability.
2. SGE for a *large* $U = Q$ (e.g., 10 000 traces in case of the ASCAD dataset) presents more indicative results about profiling attack performance, but the SGE results are still insufficient to describe model's learnability.

To estimate more precisely the model's learnability, we use the $L_m(\mathbf{HWDD}, \mathbf{g}_{sge})$ metric. This means that after obtaining SGE results, a proper choice is to calculate $L_m(\mathbf{HWDD}, \mathbf{g}_{sge})$ to confirm if the SGE value for the correct key candidate indicates learnability or biased behavior. In case of results from Figure 8a, $L_m(\mathbf{HWDD}, \mathbf{g}_{sge}) = 0.5791$ for 10 000 attack traces. If we look at Figure 6a, this correlation value is already



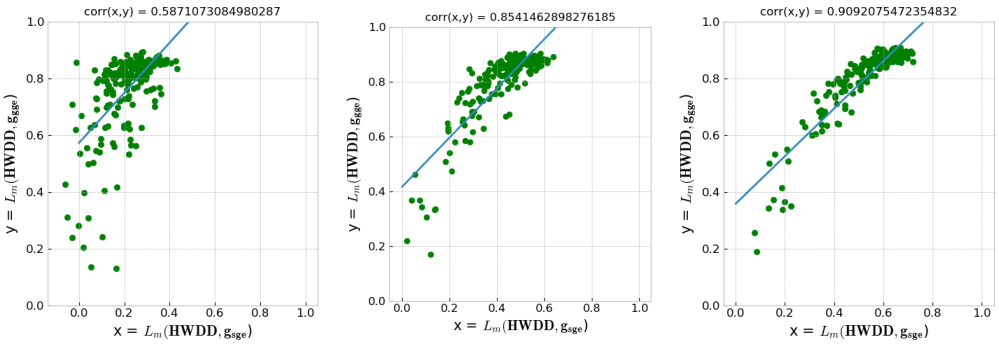
(a) SGE computed with $Q = U = 10\,000$

(b) GGE computed with $Q = 100\,000$ and $U = 10\,000$

Figure 8: The effects of simple and generalized guessing entropy for the same profiled model.

sufficient to indicate there is model learnability.

As GGE represents the reference result for profiled model's performance, we compare the $L_m(\mathbf{HWDD}, \mathbf{g})$ results for $\mathbf{g} = \mathbf{g}_{sge}$ and $\mathbf{g} = \mathbf{g}_{gge}$, as we used the Hamming weight leakage model in our attack. Figure 9 shows results for 500 trained MLP models (with different hyperparameters and different amount of profiling traces) when SGE is computed for $U = Q = 1\,000$, $U = Q = 5\,000$ and $U = Q = 10\,000$ attack traces. The GGE values are calculated for $U = 1\,000$, $U = 5\,000$ and $U = 10\,000$ attack traces and for $Q = 100\,000$ to have a higher randomization factor. As Figure 9c indicates, the values of $L_m(\mathbf{HWDD}, \mathbf{g}_{sge})$ and $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ are highly correlated (correlation of 0.929) when SGE is computed with $U = Q = 10\,000$, which is not the case when $U = Q = 1\,000$ for SGE (correlation of 0.645), as shown in Figure 9a.



(a) $U = Q = 1\,000$ for SGE.

(b) $U = Q = 5\,000$ for SGE

(c) $U = Q = 10\,000$ for SGE.

Figure 9: SGE vs GGE for ASCAD dataset, with Hamming weight leakage model.

We conclude that $L_m(\mathbf{HWDD}, \mathbf{g}_{sge})$, when Q (resp. U) is sufficiently large, is a reliable metric to discern the profiled model learnability, mainly because $L_m(\mathbf{HWDD}, \mathbf{g}_{sge})$ is highly correlated to $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$. This way, the attack phase in a profiled side-channel analysis does not necessarily need to estimate generalized guessing entropy with a

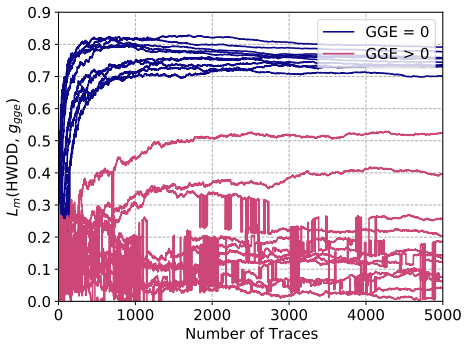
large (and in many cases unavailable) amount of attack traces.

Remark 3. Simple guessing entropy should not be considered as a reliable attack metric in the general case. Instead, one needs to evaluate the correlation between **LDD** and $\mathbf{g}_{sge}$.

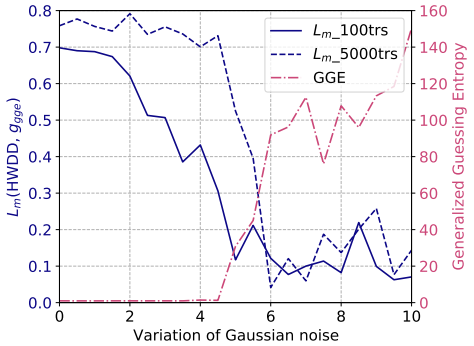
Up to now, we showed that GGE is a proper metric to evaluate the model learnability. When there are no sufficient attack traces available to calculate GGE, we must calculate $L_m(\mathbf{LDD}, \mathbf{g}_{sge})$. Next, the question is how reliable L_m is in the presence of common difficulties one encounters in the profiled side-channel analysis. In the next sections, we discuss the scenarios with different noise levels and profiling models.

6.3 Measuring Learnability for Different Noise Levels

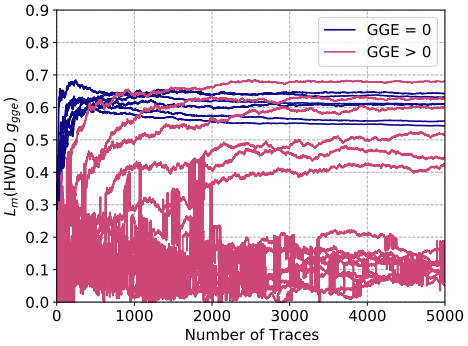
Intuitively, the leakage becomes more challenging to learn with the traces become noisy. One can expect lower model learnability with an increasing amount of noise. To simulate the noise, we consider random values following Gaussian distribution with zero mean and variance ranging from 0 to 10 in steps of 0.5. We add them to the ASCAD dataset with random keys. In terms of attack settings, CNN and template attack use 50 000 traces (randomly selected out of 200 000 profiling traces) for profiling and 5 000 traces (randomly selected out of 100 000 attack traces) for the attack. In terms of attack settings, CNN_best from the ASCAD paper is used for deep learning attack; for TA, 20 POIs are selected from the traces according to the trace variation of the HW of the intermediate data (S-box output). The guessing entropy is averaged on 100 attacks. Moreover, instead of only using final GGE to calculate $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$, we investigate the correlation changes for every attack trace. The results are shown in Figure 10.



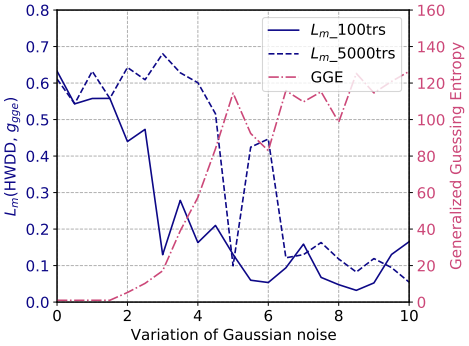
(a) CNN: L_m vs attack traces.



(b) CNN: L_m vs noise variation.



(c) TA: L_m vs attack traces.



(d) TA: L_m vs noise variation.

Figure 10: The relationship between the learnability metric and different level of noise.

As shown in Figures 10a and 10c, for the attacks where the GGE for the correct key reaches zero (in blue), L_m increases quickly when the number of attacks traces increases; the final correlation values are above 0.7 and 0.5 for CNN and template attack, respectively. A detailed analysis is presented in Figures 10b and 10d. Here, L_m is calculated with different numbers of attack traces (highlighted in blue). We see that L_m reduces with the increased variation of noise, which is also reflected in the increasing value of guessing entropy (highlighted in purple). Note that 100 attack traces is far from sufficient for GGE of the correct key to reach zero. However, with these limited amounts of traces, L_m clearly indicates the model's ability to obtain the correct key.

Moreover, by comparing L_m changes with 100 and 5 000 attack traces in these two figures, some of the attacks with higher noise variation could reach similar L_m value by increasing the number of attack traces. Indeed, the accumulating of the model's output class probabilities with more attack traces benefits the classification performance [PCP19]. However, if the model failed to learn from the leakage (e.g., due to the high noise variation), adding more attack traces would not be helpful in retrieving the correct key.

In general, the addition of the noise degrades the model's learnability. We note that there is research showing that the addition of noise could enhance the robustness of the model (as it serves as a regularization factor), eventually becoming beneficial in the process of classification [KPH+19]. Based on our results, the level of "beneficial" noise has an upper limit, and the noise we added is larger than the limitation.

Remark 4. Adding noise negatively influences the model's learnability. L_m can estimate model learnability correctly with a smaller number of traces than we require to reach a GGE of 0.

6.4 Measuring Learnability for Different Number of Epochs and Model Complexities

Model complexity and training epochs are two major contributing factors to the learnability of a model. Specifically, by increasing the number of training epochs, one can expect that the model's learnability becomes better if overfitting does not occur. On the other hand, adjusting the model size will directly influence the capacity of the model. Here, we test different models by independently varying the number of training epochs and model size.

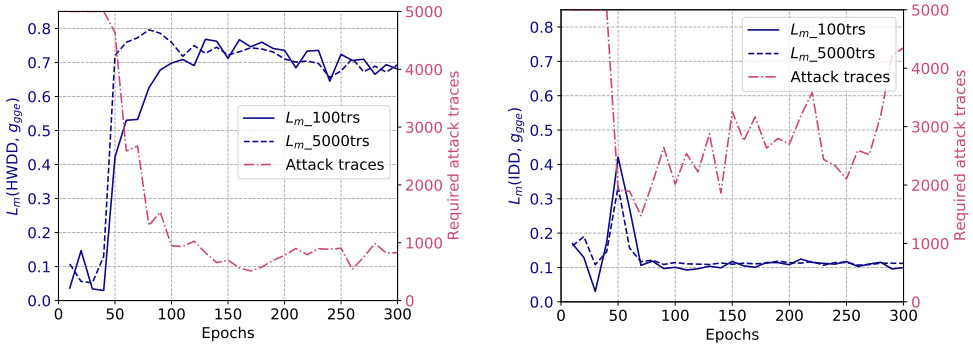
Aligned with the previous sections, CNN used for attacks is listed in Table 1. The variable i determines the number of the filters and neurons in the fully-connected layer. We use i to roughly estimate the complexity of a model. Note, for the CNN_best from the ASCAD paper, i is set to 64.

Table 1: CNN architecture used for attacking.

Layer	**Filter size**	**# of filters**	**Pooling stride**	**# of neurons**
Conv block	11	i*1	2	-
Conv block	11	i*2	2	-
Conv block	11	i*4	2	-
Conv block	11	i*8	2	-
Flatten	-	-	-	-
Fully-connected * 2	-	-	-	i*64

First, CNN_best (i=64) was trained with a different number of epochs ranging from 10 to 300 in steps of 10. For each step of training epochs, L_m is calculated with a different number of attack traces (100 and 5 000) and two different leakage models: HW and identity. The result is shown in Figure 11.

For the HW leakage model, as shown in Figure 11a, the required number of attack traces for the correct key reduces significantly when training epochs increase from 0 to 80, indicating than model gradually learns from the dataset. After that, it becomes



(a) $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ vs training epochs with HW leakage model.

(b) $L_m(\mathbf{IDD}, \mathbf{g}_{gge})$ vs training epochs with identity leakage model.

Figure 11: The relationship between learnability metric and number of training epochs.

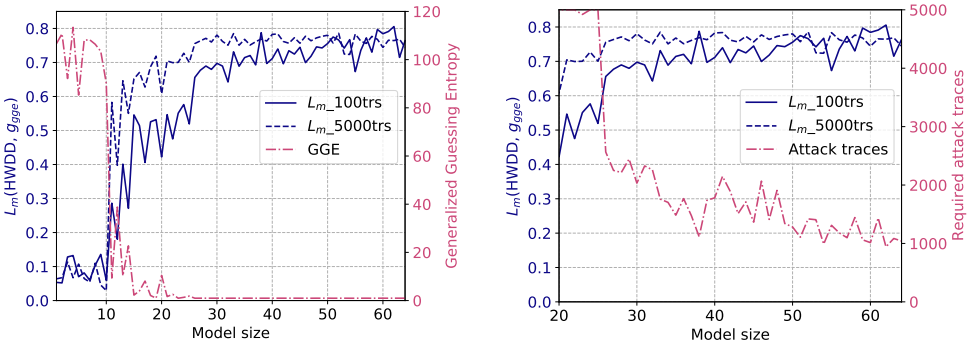
stable when the model is further trained. The same tendency could be identified with the $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ metric with both 100 and 5 000 attack traces. The L_m metric can properly estimate the model's learnability with only 100 traces (at least ten times less than GGE), suggesting the effectiveness of this metric in evaluating the model's learnability.

For the ID leakage model (Figure 11b), we see both $L_m(\mathbf{IDD}, \mathbf{g}_{gge})$ and number of required attack traces behaves unexpectedly when training epoch increases: the number of the required attack traces decreases quickly when training epochs go from 0 to 60, then gradually increase with more training epochs. Indeed, by observing the training accuracy and loss, the overfitting becomes dominant for more than 60 training epochs, which means the model's classification capability is degraded if the model is further trained. Interestingly, this conclusion is perfectly aligned with the L_m tendency for the increase of the epochs: the value reaches a maximum with 60 training epochs, decreases, and becomes stable at around 0.22 when the number of epoch increases further. Therefore, we confirm that L_m quantifies the learnability of the model. Again, 100 attack traces are sufficient for this metric to evaluate the model's learnability.

Remark 5. Adding more epochs improves the model learnability provided that it does not start to overfit. L_m can correctly estimate the model learnability for a small number of attack traces.

After evaluating the model's learnability with different training epochs, we set the number of epochs to 100, and we tune the profiling model. Here, i, the controlling factor of the model's complexity, varies from 1 to 64. We use the HW leakage model for the attack. The results are shown in Figure 12a with GGE of the correct key as a reference. Within the model size range where GGE reaches one, we present a zoom-in view using the required number of attack traces to retrieve the key (instead of GGE) for the evaluation of the attack performance (Figure 12b).

The $L_m(\mathbf{HWDD}, \mathbf{g}_{gge})$ value increases significantly (from 0.1 to 0.68) when i increases from 1 to 11, which is perfectly aligned with the decreasing tendency of GGE. Then, when the L_m is above 0.7 (i>26), GGE reaches the value 1. From the zoom-in view in Figure 12b, one could observe that the L_m is still slowly rising with the increment of i; the required attack traces, on the other hand, decrease gradually and become stable at around 1 100. Indeed, the model's learnability to a certain dataset has its limitation; in our case, it is "saturated" when i is greater than 26. Further increase of the model size will not contribute to a better attack performance. Again, these observations are clearly represented by the L_m. Attack set with 100 traces is sufficient for the evaluation of the model, which is aligned with the previous experiment.



(a) L_m vs model size.

(b) Zoom-in view: L_m vs model size.

Figure 12: The relationship between learnability metric and model complexity.

Remark 6. Increasing the model capacity improves the model learnability. L_m can correctly estimate the model learnability for a small number of attack traces.

7 Conclusions and Future Work

In the profiled side-channel analysis, one commonly uses guessing entropy to estimate the attack performance. Additionally, it is common to use all available attack traces in the guessing entropy calculation to make the evaluation more precise. This paper shows how such a practice can lead to misleading results, where one obtains wrong indication of model learnability. First, we define two guessing entropy notions: simple guessing entropy and generalized guessing entropy. We show that generalized guessing entropy is a more powerful metric but could be difficult to apply in practice. Consequently, to evaluate the attack performance (and implicitly, model learnability) we define the Leakage Distribution Difference metric (LDD). Our results show that by observing the correlation between LDD and key guessing vector, one can reliably and efficiently deduce the attack's performance. Indeed, we verified that this correlation approaches 1 when the model can perfectly learn side-channel leakages for the correct chosen leakage model.

In future work, we plan to examine whether our conclusions hold for another commonly used SCA metric: success rate. Additionally, we plan to investigate the newly proposed correlation metric in the context of leakage assessment. Finally, we believe our results also apply for the non-profiled SCA, representing an interesting research direction.

References

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

[CDP17] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional neural networks with data augmentation against jitter-based countermeasures. In Wieland Fischer and Naofumi Homma, editors, *Cryptographic Hardware and Embedded Systems – CHES 2017*, pages 45–68, Cham, 2017. Springer International Publishing.

[CK14] Omar Choudary and Markus G. Kuhn. Efficient template attacks. In Aurélien Francillon and Pankaj Rohatgi, editors, *Smart Card Research and Advanced Applications*, pages 253–270, Cham, 2014. Springer International Publishing.

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

[CRR03] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[GD98] Matt W Gardner and SR Dorling. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. *Atmospheric environment*, 32(14-15):2627–2636, 1998.

[GHO15] R. Gilmore, N. Hanley, and M. O'Neill. Neural network based attack on a masked implementation of AES. In *2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 106–111, May 2015.

[HGG20a] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Applications of machine learning techniques in side-channel attacks: a survey. *J. Cryptographic Engineering*, 10(2):135–162, 2020.

[HGG20b] Benjamin Hettwer, Stefan Gehrer, and Tim Güneysu. Deep neural network attribution methods for leakage analysis and symmetric key recovery. In Kenneth G. Paterson and Douglas Stebila, editors, *Selected Areas in Cryptography – SAC 2019*, pages 645–666, Cham, 2020. Springer International Publishing.

[HPGM16] Annelie Heuser, Stjepan Picek, Sylvain Guilley, and Nele Mentens. Side-channel analysis of lightweight ciphers: Does lightweight equal easy? In Gerhard P. Hancke and Konstantinos Markantonakis, editors, *Radio Frequency Identification and IoT Security - 12th International Workshop, RFIDSec 2016, Hong Kong, China, November 30 - December 2, 2016, Revised Selected Papers*, volume 10155 of *Lecture Notes in Computer Science*, pages 91–104. Springer, 2016.

[HZ12] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Werner Schindler and Sorin A. Huss, editors, *COSADE*, volume 7275 of *LNCS*, pages 249–264. Springer, 2012.

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

[KKK16] Been Kim, Rajiv Khanna, and Oluwasanmi O Koyejo. Examples are not enough, learn to criticize! criticism for interpretability. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 2280–2288. Curran Associates, Inc., 2016.

[KPH+19] Jaehun Kim, Stjepan Picek, Annelie Heuser, Shivam Bhasin, and Alan Hanjalic. Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 148–179, 2019.

[LMBM13] Liran Lerman, Stephane Fernandes Medeiros, Gianluca Bontempi, and Olivier Markowitch. A Machine Learning Approach Against a Masked AES. In *CARDIS*, Lecture Notes in Computer Science. Springer, November 2013. Berlin, Germany.

[LPB+15] Liran Lerman, Romain Poussier, Gianluca Bontempi, Olivier Markowitch, and François-Xavier Standaert. Template attacks vs. machine learning revisited (and the curse of dimensionality in side-channel analysis). In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 20–33. Springer, 2015.

[MDP19a] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient visualization for general characterization in profiling attacks. In Ilia Polian and Marc Stöttinger, editors, *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, COSADE 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, volume 11421 of *Lecture Notes in Computer Science*, pages 145–167. Springer, 2019.

[MDP19b] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. A comprehensive study of deep learning for side-channel analysis. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):348–375, Nov. 2019.

[MPP16] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In *International Conference on Security, Privacy, and Applied Cryptography Engineering*, pages 3–26. Springer, 2016.

[OC14] Colin O'Flynn and Zhizhang David Chen. Chipwhisperer: An open-source platform for hardware embedded security research. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 243–260. Springer, 2014.

[PBP20] Guilherme Perin, Ileana Buhan, and Stjepan Picek. Learning when to stop: a mutual information approach to fight overfitting in profiled side-channel analysis. Cryptology ePrint Archive, Report 2020/058, 2020. https://eprint.iacr.org/2020/058.

[PCP19] Guilherme Perin, Lukasz Chmielewski, and Stjepan Picek. Strength in numbers: Improving generalization with ensembles in profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/978, 2019. https://eprint.iacr.org/2019/978.

[PHJ+17] Stjepan Picek, Annelie Heuser, Alan Jovic, Simone A. Ludwig, Sylvain Guilley, Domagoj Jakobovic, and Nele Mentens. Side-channel analysis and machine learning: A practical perspective. In *2017 International Joint Conference on Neural Networks, IJCNN 2017, Anchorage, AK, USA, May 14-19, 2017*, pages 4095–4102, 2017.

[PHJ+18] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019(1):209–237, Nov. 2018.

[PHPG19] Stjepan Picek, Annelie Heuser, Guilherme Perin, and Sylvain Guilley. Profiling side-channel analysis in the efficient attacker framework. Cryptology ePrint Archive, Report 2019/168, 2019. https://eprint.iacr.org/2019/168.

[SLP05] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. In Josyula R. Rao and Berk Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, pages 30–46, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

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

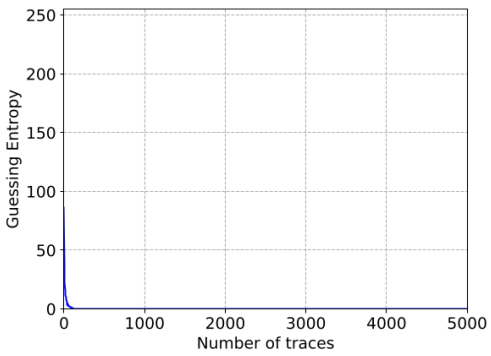
[vdVP19] Daan van der Valk and Stjepan Picek. Bias-variance decomposition in machine learning-based side-channel analysis. Cryptology ePrint Archive, Report 2019/570, 2019. https://eprint.iacr.org/2019/570.

[vdVPB19] Daan van der Valk, Stjepan Picek, and Shivam Bhasin. Kilroy was here: The first step towards explainability of neural networks in profiled side-channel analysis. Cryptology ePrint Archive, Report 2019/1477, 2019. https://eprint.iacr.org/2019/1477.

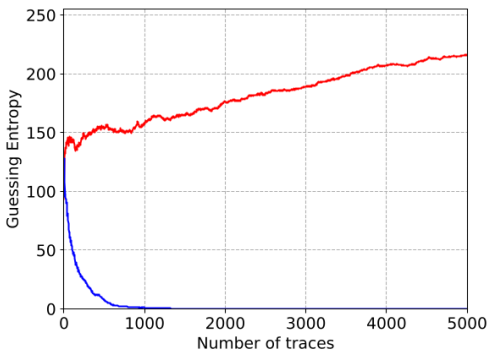
[ZBHV19] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(1):1–36, Nov. 2019.

A Additional Results for Different GE Behaviors

This section (Figures 13 to 16) depicts additional results showing how GE for the correct key can decrease, increase, or stay random. Consequently, we display cases where the model learns the correct leakage, wrong leakage, or no leakage whatsoever. We show this for several publicly available datasets and architectures from related works.

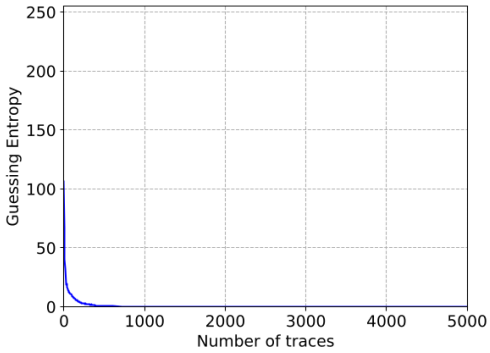


(a) CNN from [ZBHV19] on the synchronized ASCAD dataset with a fixed key with the ID leakage model.

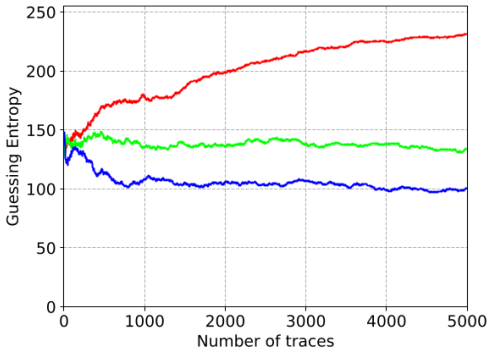


(b) CNN from [ZBHV19] on the synchronized ASCAD dataset with a fixed key with the HW leakage model.

Figure 13: CNN from [ZBHV19] on the synchronized ASCAD dataset with a fixed key with the ID leakage model (Figure 13a) shows GE decreasing, but changing some settings can result in GE that does not converge. Here, we changed the leakage model from ID to HW (Figure 13b), and we noticed cases with GE decreasing and increasing.

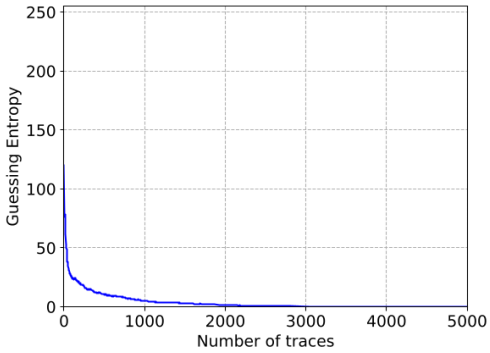


(a) MLP from [BPS+20] on the synchronized ASCAD dataset with fixed key.

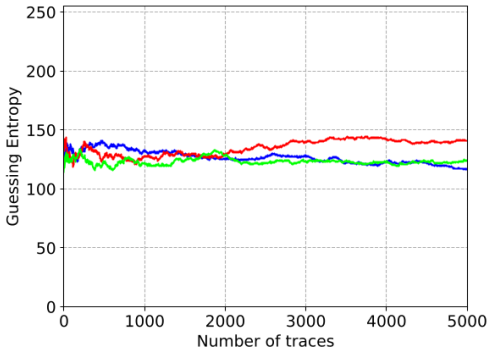


(b) MLP from [BPS+20] on the desynchronized ASCAD dataset with fixed key.

Figure 14: Examples of GE cases with MLP from [BPS+20]. Figure 14a shows results for the synchronized ASCAD dataset with a fixed key where GE is decreasing. Figure 14b shows results for the desynchronized ASCAD ($N^{[0]} = 50$) dataset with a fixed key, where GE is increasing or staying random.

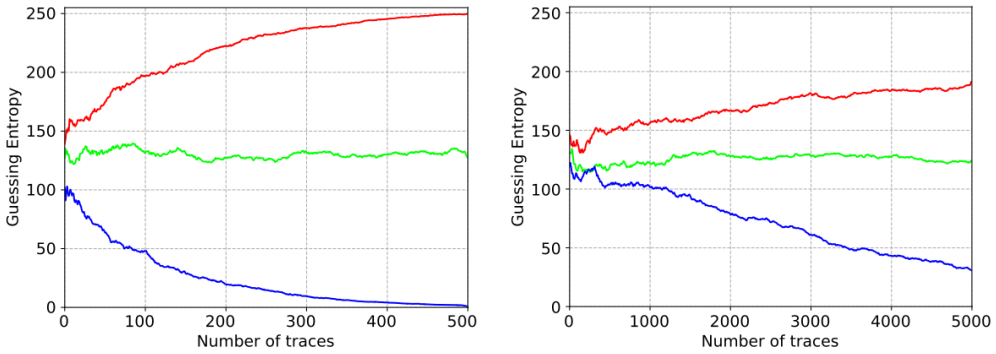


(a) CNN from [BPS+20] on the synchronized ASCAD dataset with random keys.



(b) CNN from [ZBHV19] on the synchronized ASCAD dataset with random keys.

Figure 15: Examples of GE cases with CNNs on a dataset with random keys. Figure 15a shows results with CNN from paper [BPS+20] where GE decreases. Figure 15b shows results on the same dataset but with the architecture from [ZBHV19] where GE stays mostly random.



(a) DPA contest v4 dataset: MLP.

(b) AES_HD dataset: MLP.

Figure 16: Examples of GE cases using MLP architecture for DPA contest v4 dataset [8] and the AES_HD dataset [9]. Figure 16a shows results on DPA contest v4 dataset for MLP with six hidden layers, *he_uniform* weight initializer, *Tanh* inner activation functions, *Softmax* activation function in the last layer, ID leakage model, 50 epoches, 50 batch size, 1e-3 learning rate, 4 000 training traces, and 500 attacking traces. Figure 16b shows results for the AES_HD dataset for MLP with six hidden layers, *glorot_normal* weight initializer, *ELU* inner activation functions, *Softmax* activation function in the last layer, ID leakage model, 50 epoches, 256 batch size, 1e-3 learning rate, 45 000 training traces, and 5 000 attacking traces.