Gradient Visualization for General Characterization in Profiling Attacks

Loïc Masure[1,2], Cécile Dumas[1], and Emmanuel Prouff[2,3]

[1] Univ. Grenoble Alpes, CEA, LETI, DSYS, CESTI, F-38000 Grenoble
{loic.masure, cecile.dumas}@cea.fr
[2] Sorbonne Universités, UPMC Univ Paris 06, POLSYS, UMR 7606, LIP6, F-75005, Paris, France
[3] ANSSI, France emmanuel.prouff@ssi.gouv.fr

Abstract. Past few years have seen the emergence of Machine Learning and Deep Learning algorithms as promising tools for profiling attacks, especially Convolutional Neural Networks (CNN). The latters have indeed been shown to overcome countermeasures such as de-synchronization or masking. However, CNNs are not widely used yet and Gaussian Templates are usually preferred. Though their efficiency is highly impacted by the countermeasures previously mentioned, their relevance relies on theoretical and physical justifications fairly recognized among the Side Channel community. Instead, the efficiency of CNNs still raises a certain scepticism as they act as a black-box tool. This scepticism is not specific to the Side Channel Analysis context: understanding to what extent CNNs would be so powerful and how they learn to recognize discriminative features for classification problems is still an open problem. Some methods have been proposed by the computer vision community, without satisfying performance in this field.

However, methods based on *Sensitivity Analysis* particularly fit our problem. We propose to apply one of them called *Gradient Visualization* that uses the derivatives of a CNN model with respect to an input trace in order to accurately identify temporal moments where sensitive information leaks. In this paper, we theoretically show that this method may be used to efficiently localize Points of Interest in the SCA context. The efficiency of the proposed method does not depend on the particular countermeasure that may be applied to the measured traces as long as the profiled CNN can still learn in presence of such difficulties. In addition, the characterization can be made for each trace individually. We verified the soundness of our proposed method on simulated data and on experimental traces from a public Side Channel database. Eventually we empirically show that Sensitivity Analysis is at least as well as state-of-the-art characterization methods, in presence (or not) of countermeasures.

Keywords: Side Channel Analysis · Profiling Attacks · Deep Learning · Points of Interest · Characterization

1 Introduction

Side Channel Analysis (SCA) is a class of cryptanalytic attacks that exploits weaknesses of a physical implementation of a cryptographic primitive. During its execution, the primitive will process a piece of plaintext and of secret key into a *sensitive variable*. As the processing is invertible, knowing the value of this variable (or at least having some information about it) and the plaintext enables an attacker to recover the piece of secret key. Secure cryptographic algorithms such as the *Advanced Encryption Standard* (AES) can then be defeated by recovering each byte of secret key separately thanks to a *Divide and Conquer* strategy, thereby breaking the high complexity usually required to defeat such an algorithm. This information is usually gathered thanks to physical leakages such as the power consumption or the electromagnetic emanations measured on the target device. Actually, conducting an SCA is equivalent as studying the conditional probability of the sensitive variable given the physical measure. It can be done for example through the computation of statistics such as a difference of means [16] or a correlation coefficient [3].

For the specific type of SCA called *profiling attacks*, an attacker will try to estimate the whole conditional distribution thanks to a profiling phase during which he has access to an open sample for which he knows the value of the target variable. Such an access allows him to estimate the conditional distribution. Historically, *Gaussian Template Attacks* (GTA) have first been proposed in the early 2000's [7]. Their efficiency however, requires an amount of traces that highly increases with the number of considered time samples. Therefore a first pre-processing step is hence required to extract few points called *Points of Interest* (PoIs). Tools like *Signal-to-Noise Ratio* (SNR) can efficiently extract those PoIs [22]. However, in presence of countermeasures such as masking or de-synchronization [30], both estimation with GTA and PoIs extraction with SNR are no longer efficient (or at least much less). Other dimensionality reduction techniques like dedicated variants of Principal Component Analysis (PCA) [4,34,11,9,8] or *Kernel Discriminant Analysis* (KDA) [6] can be used, whithout guarantee that relevant components will be extracted.

Recently, the SCA community has benefited the resurgence of *Convolutional Neural Networks* (CNNs) in the 2010's [17] to apply them to profiling attacks, as first proposed in [12,20,23]. They are seen as a black-box tool and their results have been afterwards experimentally shown to be robust to the most common countermeasures, namely masking [21] and de-synchronization [5]. Their main advantage is that they may not require pre-processing, and are at least as efficient as the other state-of-the-art profiling attacks. However, in an evaluator point-of-view, this is not sufficient. On the one hand he wants to make sure that a CNN attack succeeded for good reasons *i.e.* that the rules learned by the model are generalizable. On the other hand the evaluator also wants to help the developer to localize and understand where the vulnerability comes from in order to remove or at least reduce it. This issue is part of a more general problematic for Deep Learning based systems, namely their *explainability* and *interpretability*. On the one hand, a theoretical framework has been proposed in this context [24]. On the other hand, several methods have been tried to tackle this problem. Some computer vision research groups have studied *Sensitivity Analysis* [32,33] derived from the computation of the loss function gradient with respect to the input data during the training phase.

In this paper, we propose to apply a particular Sensitivity Analysis method called *Gradient Visualization* (GV) in order to better understand how a CNN can learn to predict the sensitive variable based on the analysis of a single trace. The main claim is that CNN based models succeed in discriminating PoIs from non-informative points, and their localization can be deduced by simply looking at the gradient of the loss function with respect to the input traces for a trained model. We theoretically show that this method can be used to localize PoIs in the case of a perfect model. The efficiency of the proposed method does not decrease when countermeasures like masking or misalignment are applied. In addition, the characterization can be made for each trace individually. We verified the efficiency of our proposed method on simulated data and on experimental traces from a public Side Channel database. We empirically show that Gradient Visualization is at least as well as state-of-the-art characterization methods, in presence or not of different countermeasures.

The paper is organized as follows. In Section 3 we start by considering the optimal model an ideal attacker may get during profiling, and we deduce some properties of its derivatives with respect to the input traces that can be related to the PoIs. In Section 4 we use these properties on a model estimated with CNNs and we explain how to practically implement the visualization method. A toy example applied on simulated data is proposed for illustration. Sections 5 and 6 are eventually dedicated to an experimental validation of the effectiveness of our proposal in realistic attacks scenarios.

2 Preliminaries

2.1 Notations

Throughout the paper we use calligraphic letters as $\mathcal{X}$ to denote sets, the corresponding upper-case letter X to denote random variables (resp. random vectors $\mathbf{X}$) over $\mathcal{X}$, and the corresponding lower-case letter x (resp. $\mathbf{x}$ for vectors) to denote realizations of X (resp. $\mathbf{X}$). The i-th entry of a vector $\mathbf{x}$ is denoted by $\mathbf{x}[i]$. We denote the probability space of a set $\mathcal{X}$ by $\mathcal{P}(\mathcal{X})$. If $\mathcal{X}$ is discrete, it corresponds to the set of vectors $[0,1]^{|\mathcal{X}|}$ such that the coordinates sum to 1. If a random variable X is drawn from a distribution $\mathcal{D}$, then $\mathcal{D}^N$ denotes the joint distribution over the sequence of N i.i.d. random variables of same probability distribution than X. The symbol $\mathbb{E}$ denotes the expected value, and might be subscripted by a random variable $\mathbb{E}_X$, or by a probability distribution $\underset{X\sim\mathcal{D}}{\mathbb{E}}$ to specify under which probability distribution it is computed. Likewise, Var denotes the variance of a random variable. If $f : x, y \mapsto f(x, y)$ is a multivariate function, $\frac{\partial}{\partial x}$ denotes the partial derivative with respect to the input variable x. Likewise, if f if a function from $\mathbb{R}^n$ to $\mathbb{R}$, then $\nabla f(\mathbf{x})$ denotes the gradient of

f computed in $\mathbf{x} \in \mathbb{R}^n$, which corresponds to the vector of the partial derivatives with respect to each coordinate of $\mathbf{x}$ respectively. If there is an ambiguity, the gradient will be denoted $\nabla_{\mathbf{x}} f(\mathbf{x}, \mathbf{y})$ to emphasize that the gradient is computed with respect to $\mathbf{x}$ only. Eventually if$f$ is a function from $\mathbb{R}^n$ to $\mathbb{R}^m$, then $J_f(\mathbf{x}) \in \mathbb{R}^{m,n}$ denotes the (m, n) matrix whose rows are the transposed gradient of each elementary function $\mathbf{x} \mapsto f(\mathbf{x})[i] \in \mathbb{R}$. The output of a cryptographic primitive $\mathbf{C}$ is considered as the target sensitive variable $Z = \mathbf{C}(P, K)$, where P denotes some public variable, e.g. a plaintext chunk, where K denotes the part of secret key the attacker aims to retrieve, and where Z takes values in $\mathcal{Z} = \{s_1, \ldots, s_{|\mathcal{Z}|}\}$. Among all the possible values K may take, $k^\star$ will denote the right key hypothesis. A side-channel trace will be viewed as a realization of a D-dimensional random column vector $\mathbf{X} \in \mathcal{X} \subset \mathbb{R}^D$.

2.2 Profiling Attacks

We will consider attacking a device through a profiling attack, made of the following steps:

- *Profiling acquisition*: a dataset of N_p *profiling traces* is acquired on the prototype device: $S_p \triangleq \{(\mathbf{x}_1, z_1), \ldots, (\mathbf{x}_{N_p}, z_{N_p})\}$.
- *Model building*: a model that returns a discrete probability distribution (pdf) $F(\mathbf{x})$ is built. If the model is accurate, the returned discrete pdf, viewed as a vector, is assumed to be a good approximation of the conditional pdf $\Pr[Z|\mathbf{X} = \mathbf{x}]$.
- *Attack acquisition*: a dataset of N_a *attack traces* is acquired on the target device: $S_a \triangleq \{(\mathbf{x}_1, z_1), \ldots, (\mathbf{x}_{N_a}, z_{N_a})\}$.
- *Predictions*: a prediction vector is computed on each attack trace, based on the previously built model: $\mathbf{y}_i = F(\mathbf{x}_i), i \in [|1, N_a|]$. It assigns a score to each key hypothesis, for each trace.
- *Guessing*: the scores are combined over all the attack traces to output a *likelihood* for each key hypothesis; the candidate with the highest likelihood is predicted to be the right key.

Let us denote by $g_{S_a}(k^\star)$ the actual rank of the correct key hypothesis returned by the attack. If $g_{S_a}(k^\star) = 1$, then the attack is considered as *successful.* More details about the score vector and the rank definitions can be found in Appendix A. The difficulty of attacking the target device is often defined as the number of traces required to get a successful attack. As many random factors may be involved during the attack, it is preferred to study its expected value, the so-called ***G****uessing* ***E****ntropy* (GE) [35]:

$$\mathrm{GE}(N_a) \triangleq \mathop{\mathbb{E}}_{S_a} [g_{S_a}(k^\star)] \quad . \tag{1}$$

The goal of an evaluator is therefore to find a model F that minimizes N_a such that $\mathrm{GE}(N_a) < 2$. We will assume that this is equivalent to the problem of accurately estimating the conditional probability distribution $\mathbf{x} \mapsto \Pr[Z|\mathbf{X} = \mathbf{x}]$.

As mentioned in the introduction, we distinguish the security evaluator as a particular attacker who additionally wishes to interpret the attack results. One step of this diagnosis is to temporally localize where the information leakage appeared in $\mathbf{x}$. This task is usually called *characterization*. It consists in emphasizing *Points of Interest* (PoIs) where the information leakage may come from. Section 4.3 will present an usual characterization technique while a new method will be introduced through this paper.

3 Study of an Optimal Model

In this section, we address the evaluator interpretation problem in the ideal situation when the conditional distribution is known (*i.e.* when the model is perfect). The latter will be denoted as F^*. We will show how the study of the derivatives of such a model with respect to each coordinate of an input trace can highlight information about our PoIs. To this end, we need two assumptions.

Assumption 1 (Sparsity) *There only exists a small set of coordinates* $\mathcal{I}_Z \triangleq \{t_1, \ldots, t_C | C \ll D\}$ *such that* $\Pr[Z|\boldsymbol{X}] = \Pr\left[Z|\boldsymbol{X}[t_1], \ldots, \boldsymbol{X}[t_C]\right]$.

Assumption 2 (Regularity) *The conditional probability distribution* F^* *is differentiable over* $\mathcal{X}$ *and thereby continuous.*

Informally, Assumption 1 tells that the leaking information is non-uniformly distributed over the trace. Both assumptions are realistic in a SCA context (this point is discussed in Appendix B).

Once Assumptions 1 and 2 are stated, we may observe the impact on the properties verified by the optimal model derivatives. For such a purpose we start by considering a trace $\mathbf{x}$. Figure 1 (top) illustrates such a trace in blue, and the green line depicts $\mathcal{I}_Z$ which is assumed to be a singleton here. The prediction pdfs $F^*(\mathbf{x})$ are given at the bottom of the same figure. We may fairly suppose that a slight variation on one coordinate that does not belong to $\mathcal{I}_Z$ (circled in gray in Figure 1, top) should not radically change the output of the optimal model. The pdf remains the same, as the gray bars are exactly covered by the blue ones in Figure 1 (bottom). However, applying a slight variation on the coordinate from $\mathcal{I}_Z$ (circled in Figure 1, top) may radically change the output distribution (red bars in Figure 1, bottom).

As a consequence, F^* is differentiable with respect to the input trace (Assumption 2) so there should exist $s \in \mathcal{Z}$ such that:

$$\frac{\partial}{\partial \mathbf{x}[t]} F^*(\mathbf{x})[s] \begin{cases} \neq 0 & \text{if } t \in \mathcal{I}_Z \\ \approx 0 & \text{if } t \notin \mathcal{I}_Z \end{cases} . \tag{2}$$

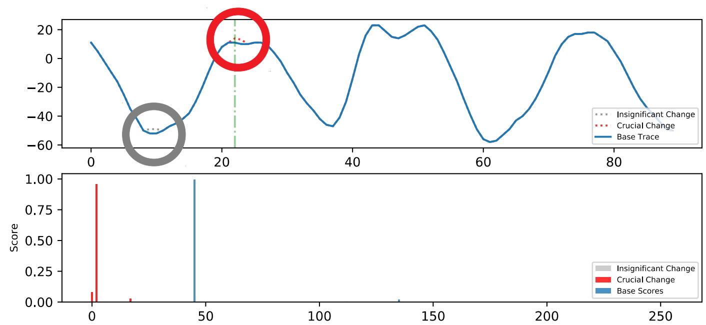

Fig. 1: Illustration of the Sensitivity Analysis principle. Top: a piece of trace. $t \in \mathcal{I}_Z$ is depicted by the green line, and slight variations circled in red and gray. Bottom: predictions of the optimal model.

The previous observation can be stated in terms of the Jacobian matrix $J_{F^*}(\mathbf{x})$: its coefficients should be zero almost everywhere, except in columns $t \in \mathcal{I}_Z$:

$$J_{F^*}(\mathbf{x}) = \begin{pmatrix} 0 \dots & \frac{\partial}{\partial \mathbf{x}[t]} F^*(\mathbf{x})[s_1] & 0 \dots 0 \\ 0 \dots & \frac{\partial}{\partial \mathbf{x}[t]} F^*(\mathbf{x})[s_2] & 0 \dots 0 \\ \vdots & \vdots & \vdots \quad \vdots \\ 0 \dots & \frac{\partial}{\partial \mathbf{x}[t]} F^*(\mathbf{x}) \left[s_{|\mathcal{Z}|}\right] & 0 \dots 0 \end{pmatrix} \tag{3}$$

The properties verified by the Jacobian matrix in (3) form the cornerstone of this paper, as it implies that we can guess from this matrix whether a coordinate from an input trace belongs to $\mathcal{I}_Z$, i.e. whether a coordinate has been recognized as a PoIs when designing the optimal model F^*. Such a technique is part of *Sensitivity Analysis*.[1] Moreover it requires no assumption on the type of countermeasure implemented in the device.

4 Our Characterization Proposal

So far we have shown that the Jacobian of an optimal model may emphasize PoIs. In practice however, the evaluator does not have access to the optimal model, but a trained estimation of it. A natural idea is hence to look at the Jacobian matrix of the model estimation, hoping that its coefficients will be close to the optimal model derivatives. Here we follow this idea in contexts where the approximation is modeled by training *Convolutional Neural Networks* (CNN), described in Section 4.1 (discussions can be found in Appendix C about this approximation). Section 4.2 illustrates our claim with a toy example. Finally

[1] A general definition of *Sensitivity Analysis* is the study of how the uncertainty in the output of a mathematical model or system (numerical or otherwise) can be apportioned to different sources of uncertainty in its inputs [1].

we relate our study with state-of-the-art methods for leakage characterization in Section 4.3.

4.1 Gradient Approximation with Neural Networks

Neural Networks (NN) [19] aim at constructing a function $F\colon \mathcal{X} \to \mathcal{P}(\mathcal{Z})$ composed of several simple operations called *layers*. All the layers are entirely parameterized by (a) a real vector called *trainable weights* and denoted by θ that can be automatically set; (b) other parameters defining the general architecture of the model are gathered under the term *hyper-parameter*. The latters are defined by the attacker/evaluator.

Convolutional Neural Networks (CNN) is a specific type of Neural Network where particular constraints are applied on the trainable weights [18]. The training phase consists in an automatic tuning of the trainable weights and it is done *via* an iterative approach that locally applies the Stochastic Gradient Descent algorithm to minimize a *loss function* quantifying the classification error of the function F over the training set. For further details, the interested reader may refer to [13].

To accurately and efficiently compute the Jacobian matrix of a CNN, an algorithm called *backward propagation* (or backprop) can exactly compute the derivatives with the same time complexity as computing $F(\mathbf{x}, \theta)$ [13]. As a consequence, computing such a matrix can be done with a negligible cost during an iteration of Stochastic Gradient Descent.

Actually the modern Deep Learning libraries [26,2] are optimized to compute the required derivatives for Stochastic Gradient Descent in the backprop so the Jacobian matrix is never explicitly stored, and it is easier to get the loss function gradient with respect to the input trace $\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta), z^*)$, where $\ell : \mathcal{P}(\mathcal{Z}) \times \mathcal{Z} \to \mathbb{R}_+$ denotes the loss function, and z^* denotes the true sensitive value. Hopefully, studying either the latter one or $J_F(\mathbf{x})$ is fairly equivalent, as one coordinate of the loss function gradient is a function of elements from the corresponding column in the Jacobian matrix:

$$\nabla_{\mathbf{x}} \ell(F(\mathbf{x}, \theta), z) = J_F(\mathbf{x}, \theta)^T \nabla_{\mathbf{y}} \ell(F(\mathbf{x}, \theta), z). \tag{4}$$

That is why we propose to visualize the latter gradient to characterize PoIs in the context of a CNN attack, instead of the Jacobian matrix (unless explicit mention). To be more precise, we visualize each coordinate of the gradient in absolute value in order to get the sensitivity magnitude. In the following, such a method will be denoted as *Gradient Visualization* (GV). Discussions about other visualization methods can be found in Appendix D.

4.2 Example on Simulated Data

To illustrate and explain the relevance of GV, and before going on experimental data, we first apply our method on a toy example, aiming at simulating simple

D-dimensional leakages from an n-bit sensitive variable Z. The traces are defined such that for every $t \in [|1, D|]$:

$$\mathbf{x}_i[t] = \begin{cases} U_i + B_i, \text{ if } t \notin \{t_1, \ldots, t_m\} \\ z_{t,i} + B_i \text{ otherwise} \end{cases}, \tag{5}$$

where $(U_i)_i, (B_i)_i$ and all $(z_{t,i})_i$ are independent, $U_i \sim \mathcal{B}(n, 0.5)$, $B_i \sim \mathcal{N}(0, \sigma^2)$ and where $(z_{1,i}, \ldots, z_{m,i})$ is a m-sharing of z_i for the bitwise addition law. This example corresponds to a situation where the leaks of the shares are hidden among values that have no relation with our target.

Every possible combination of the m-sharing has been generated and replicated 100 times before adding the noise, in order to have an exhaustive dataset. Therefore, its size is 100×2^{mn}. We ran the experiment for $n = 4$ bits, $m \in \{2, 3\}, D = 100$, and a varying noise $\sigma^2 \in [0, 1]$. Once the data were generated, we trained a neural network with one hidden layer made of D neurons. Figure 2

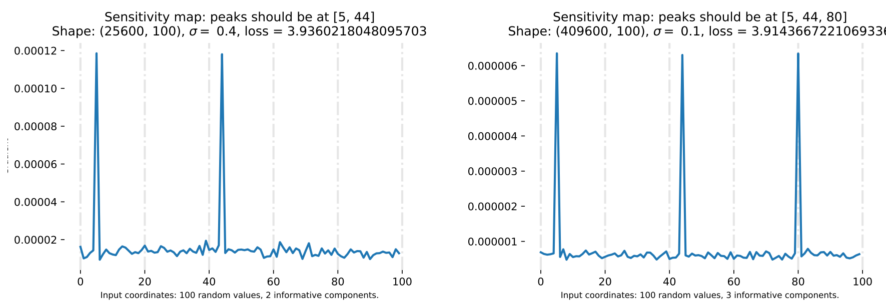

Fig. 2: Gradient of the loss function respectively for two and three shares.

presents some examples obtained for 2 (left) and 3 shares (right). We clearly see some peaks at the coordinates where the meaningful information have been placed. Interestingly, this simulation shows that a second order masking has been defeated, though it required 16 more simulated data and less noised data ($\sigma^2 \geq 0.1$) than for the same experiment with only a first order masking. A further work might study the impact of the noise magnitude σ^2 and of the number of bits considered on the efficiency of the training. It is however beyond the scope of this paper.

4.3 Comparison with another Characterization Method: the SNR

Now we have shown that Gradient Visualization is relevant for characterization on simulated data, one may wonder to what extent this method would be useful compared to other characterization techniques. In this section, we compare our contribution to a state-of-the-art method used for characterization in SCA. The

Signal-to-Noise Ratio (SNR) is estimated by the following pointwise statistics:

$$\mathrm{SNR}[t] \triangleq \frac{\underset{Z}{\mathrm{Var}}\left(\mathbb{E}\left[\mathbf{X}[t]|Z=z\right]\right)}{\underset{Z}{\mathbb{E}}\left[\mathrm{Var}\left(\mathbf{X}[t]|Z=z\right)\right]} \quad , \tag{6}$$

where the numerator denotes the signal magnitude and the denominator denotes the noise magnitude estimate. More on the SNR can be found in [22]. One has to keep in mind that the SNR is a statistical tool, and produces one characterization from the profiling traces; whereas our method gives one map for each trace, though we might average them. It has two consequences, in an SNR characterization with masking, every trace coordinate $\mathbf{X}[t]$ will be independent from Z, leading the numerator to converge toward 0. Similarly, de-synchronization is equivalent to having a stronger noise on the traces, leading the denominator to explode (see Section 3.2 in [30]). It explains why it cannot highlight higher order leakages. Some solutions have been proposed, especially by using functions combining every couple of points [28] or realignment techniques [36,25,10].

5 Experiment Description

So far we have claimed that relevant information can be extracted from the loss gradient of a differentiable model. Following this idea, it has been shown to be efficient to localize PoIs on simulated data and validated that this method might overcome some weaknesses of state-of-the-art techniques. We now plan to experimentally verify these claims. Before introducing the results in Section 6, we first describe our investigations. In Section 5.1, we present the CNN architecture that will be used for profiling. Finally, Section 5.2 gives an exhaustive description of all the considered parameters for our experiments.

5.1 CNN Architecture

For these experiments, we will consider the same architecture as proposed in [5,29] up to slight modifications:

$$s \circ [\lambda]^{n_1} \circ \delta_G \circ [\delta \circ [\sigma \circ \mu \circ \gamma]^{n_2}]^{n_3} \quad , \tag{7}$$

where γ denotes a convolutional layer, σ denotes an elementwise non-linearity, μ denotes a batch-normalization layer, λ denotes a dense layer and s denotes the softmax layer. Compared to the original architecture proposed in those papers, a lighter baseline CNN architecture has been chosen:

- The number of filters in the first layers has been decreased (10 instead of 64), though it is still doubled between each block; and the filter size has been set to 5.
- The dense layers contain less neurons: 1,000 instead of 4,096.

– A *global pooling layer* δ_G, has been added at the top of the last block. Its pooling size equals the width of the feature maps in the last convolutional layer, so that each feature maps are reduced to one point. While it acts as a regularizer (since it will drastically reduce the number of neurons in the first dense layer), the global pooling layer also forces the convolutional filters to better localize the discriminative features [37].

5.2 Settings

Our experiments have been done with the $50,000$ EM traces from ASCAD database [29]. Each trace is made of 700 time samples.[2] Here-after, the three different configurations investigated in this paper are presented with the notations taken from [29]:

– Synchronized traces, with the masked data as a target: $Z = \mathrm{Sbox}(P \oplus K) \oplus r_{out}$ (in other terms, r_{out} is assumed to be known, like P).
– Artificial shift: same as previously, but the traces are artificially shifted to the left of a random number of points drawn from a uniform distribution over $[|0, 100|]$. The traces correspond to the dataset `ASCAD_desync100`.
– Synchronized traces, with unknown masking: we have no knowledge of the masks r_{out} (neither during profiling or attack phase). The traces correspond to the dataset `ASCAD`.

The Neural Networks source code is implemented on Python thanks to the Pytorch [26] library and is run on a workstation with a Nvidia Quadro M4000 GP-GPU with 8 GB memory and 1664 cores.

We will use the *Cross-Entropy* (also known as Negative Log Likelihood) as a loss function. It particularly fits our context as it is equivalent as minimizing the Kullback-Leibler divergence, which measures a divergence between two probability distributions, namely $F^*(\mathbf{x})$ and $F(\mathbf{x}, \theta)$ in our case. Therefore, the model $F(., \theta)$ will converge towards F^* during the training.

For each tested neural network architecture, a 5-fold cross-validation strategy has been followed. Namely, the ASCAD database has been split into 5 sets $S_1, \dots, S_5$ of $10,000$ traces each, and the i-th cross-validation, denoted by CV_i, corresponds to a training dataset $S_p = \cup_{j \neq i} S_j$ and a validation dataset $S_v = S_i$. The given performance metrics and the visualizations are averaged over these 5 folds. The optimization is actually done with a slight modification of Stochastic Gradient Descent called *Adam* [15]. The learning rate is always set to 10^{-4}. Likewise, the batch size has been fixed to 64. For each training, we operate 100 epochs, *i.e.* each couple $(\mathbf{x}_i, z_i)$ is passed 100 times through an SGD iteration, and we keep as the best model the one that has the lowest loss function on the validation set.

[2] It corresponds to 26 clock cycles.

6 Experimental Results

This section presents experimentations of the GV in different contexts, namely (Exp.1) when the implementation embeds no countermeasure, (Exp.2) when traces are de-synchronized and (Exp.3) when masking is applied. The methods used to train the CNNs, to tune their hyper-parameters and to generate the GV have been presented in Section 5.

6.1 Application Without Countermeasure

Parameter	Value	n_1	Validation Loss (bits)	N^*	n_1	Validation Loss (bits)	N^*
n_3	5	0	6.40	3.25	0	6.64	4.0
n_2	1	1	6.15	3	1	6.46	3.6
n_1	{0, 1, 2}	2	6.35	3.25	2	6.90	5.4

Table 1: Architecture hyper-parameters (left) and performance metrics without countermeasure (center) and with de-synchronization (right).

In application context (Exp.1) (*i.e.* no countermeasure) several CNNs have been trained with the architecture hyper-parameters in (7) specified as listed in Table 1 (left). Since the masked data are here directly targeted (*i.e.* the masks are supposed to be known), no recombination (thereby no dense layer) should be required [29]. The parameter n_1 should therefore be null. However, to validate this intuition we let it vary in $\{0, 1, 2\}$. The validation loss corresponding to these values is given in Table 1 (center), where N^* denotes the minimum number of traces required to have a GE lower than 1. Even if this minimum is roughly the same for the 3 different configurations, we selected the *best* one (i.e. $n_1 = 1$) for our *best* CNN architecture. Figure 3 (left) presents the corresponding GV, and the corresponding SNR (right).[3] It may be observed that the peaks obtained with GV and SNR are identical: the highest point in the SNR is the second highest point in GV, whereas the highest point in GV is ranked 7-th in the SNR peaks. More generally both methods target the same clock cycle (the 19-th). These observations validate the fact that our characterization method is relevant for an unprotected target.

6.2 Application with an Artificial De-synchronization

We now add a new difficulty by considering the case of de-synchronization as described in Section 5.2. The hyper-parameter grid is exactly the same as in

[3] An alternative representation with the Jacobian matrix is given in Appendix E, Figure 8.

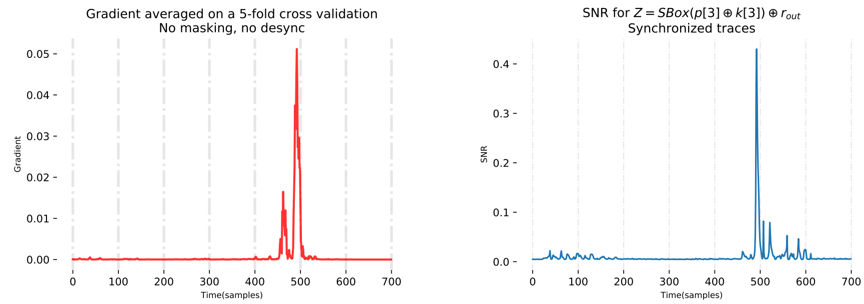

Fig. 3: Case where no countermeasure is considered. Left: GV for the trained model with 1 dense layer. Right: SNR.

Section 6.1, and the corresponding loss is given in Table 1 (right). Faced to misalignment, the considered architectures have still good performances, and the attacks succeeded in roughly the same number of traces than before. Interestingly, Figure 4 shows that the GV succeeds to recover the leakage localization while the SNR does not (see Figure 9 in Appendix E). Actually, the gradient averaged over the profiling traces Figure 4 (left) shows that, instead of having a small number of peaks, a band is obtained whose width approximately equals the maximum quantity of shift applied in the traces, namely 100 points. Moreover, individual gradients Figure 4 (right) bring a single characterization for each trace, enabling to guess approximately the shift applied to each trace.

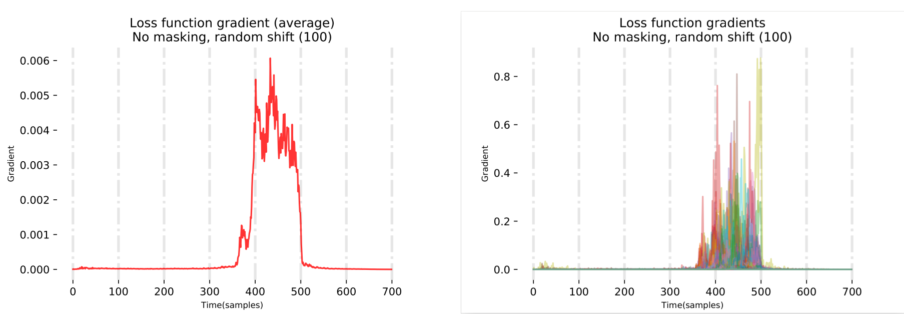

Fig. 4: Case where de-synchronization is considered. GV for each trace separately (right) and averaged (left).

6.3 Application with a First Order Masking

The last experiment concerns the application of GV in presence of masking. Several model configurations have been tested which correspond to the hyper-parameters listed in Table 2 (left). We eventually selected the 8 models that achieved the best GE convergence rate (right).

Parameter	Value
n_3	$\{5, 6, \mathbf{7}, 8\}$
n_2	1
n_1	$\{\mathbf{2}, 3\}$
`n_filters_1`	10
`kernel_size`	$\{3, 5, \mathbf{11}\}$

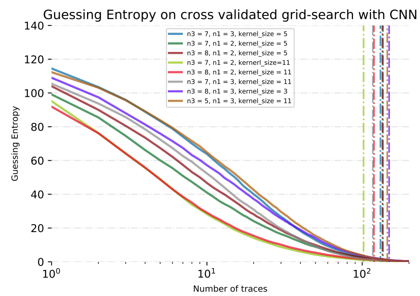

Table 2: Masking Case. Left: architecture hyper-parameters (bold values refer to the best choices). Right: GE for the 8 best architectures.

For the selected architectures, our first attempt to use GV did not give full satisfaction. As an illustration, Figure 5 (left) presents it for one of the tested architectures (averaged over the 5 folds of the cross-validation). Indeed, it looks difficult to distinguish PoIs (i.e. those identified by our SNR characterization, see the right-hand side of Figure 6) from *ghost* peaks (*i.e.* peaks *a priori* independent of the sensitive target). To explain this phenomenon, we decided to study the validation loss of the trained models. Figure 5 (right) presents it for one model and for each of the 5 cross-validation folds CV_i, $i \in [0..4]$.

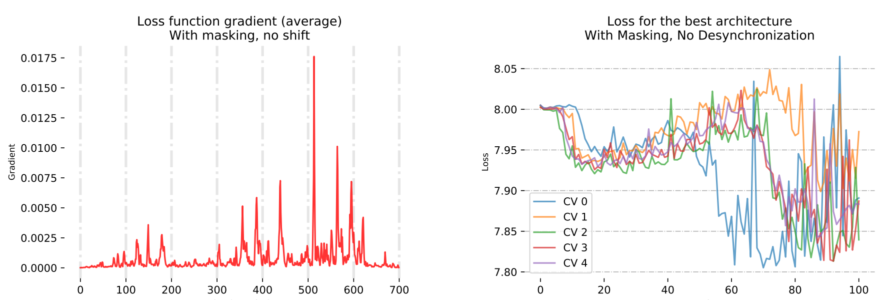

Fig. 5: Left: GV in presence of masking (without early-stopping). Right: validation loss for each fold.

It may be observed in Figure 5 (right) that the validation loss curve proceeded a fast big decrease after an initial plateau during the first 15 epochs. After that, the validation loss starts increasing before going on a regime with unstable results (after 50 epochs). These observations are clues of *overfitting*. It means that the model exploits (non-informative) leakage not localized in the PoIs to memorize the training data and to improve the training loss. Such a strategy should not generalize well on the validation traces. As we are looking for models that implement a generalizable strategy, we propose to use a regularization technique called *early-stopping* [13]: the training is stopped after a number of epochs called *patience* (in our case 10) if no remarkable decrease (*i.e.* up to a *tolerance term*, 0.25 bits here) is observed in the validation loss. With this slight modification, the previous architectures are trained again from scratch, and a better GV is produced (see the left-hand side of Figure 6). As the main peaks are separated enough, an evaluator may conclude that they represent different leakages.

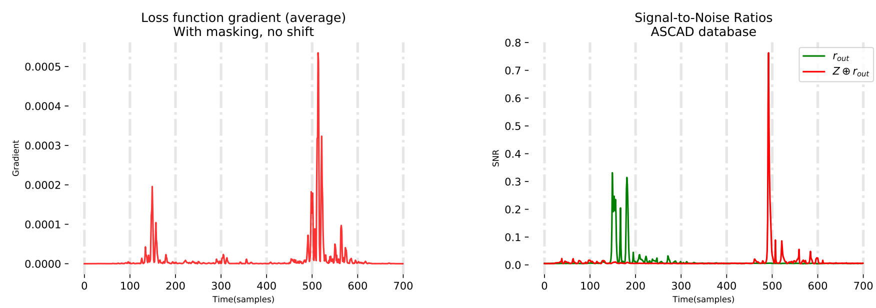

Fig. 6: Early-stopping is applied. Left: GV. Right: corresponding SNR.

6.4 Comparison with SNR in the Context of Template Attacks

A careful observation of Figure 6 shows that the main peaks given by the GV are not exactly aligned with those given by the SNR characterization (performed under the hypothesis that the masks are known). For GV, the main peak appears at the points corresponding to the 20-th clock cycle, which is one cycle after the one previously targeted by both the GV and the SNR in the previous case where no countermeasure was considered (Section 6.1). We validated that this phenomenon occurred for every successful visualization produced by GV. Furthermore, concerning the peaks related to the mask leakage, the GV only emphasizes one clock cycle (the 6-th) whereas the SNR highlights two of them: the 6-th and the 7-th.

It implies that the GV should not be taken as an exact equivalent to the SNR. We have not found any track of explanation to justify this slight shift but it raises the question to know whether such a change in the targeted clock cycles has a sense. To give an answer, we decided to use our characterization method as a dimensionality reduction method. To make a fair comparison in the context of a first order masking, we assume that we know the mask during the characterization phase, so that we can localize PoIs for the mask and the masked data. Notice that we do not assume the mask knowledge during the profiling phase.[4]

The input dimension of the traces are reduced to $2^n, n \in \{1, 2, 3, 4, 5\}$ points, based on the following methods: (a) the 2^{n-1} highest PoIs from the mask SNR and the 2^{n-1} highest PoIs from the masked data SNR are selected; (b) the 2^{n-1} highest PoIs from the GV are selected from the area around the 6-th clock cycle. Likewise, the other half comes from the peaks in the area around the 20-th clock cycle. Once reduced, the traces are processed with a first order Template Attack [7], and the GE is estimated. The results are given on Figure 7. The plain curves denote the GE for GV whereas the dotted curves denote either GE obtained with SNR.

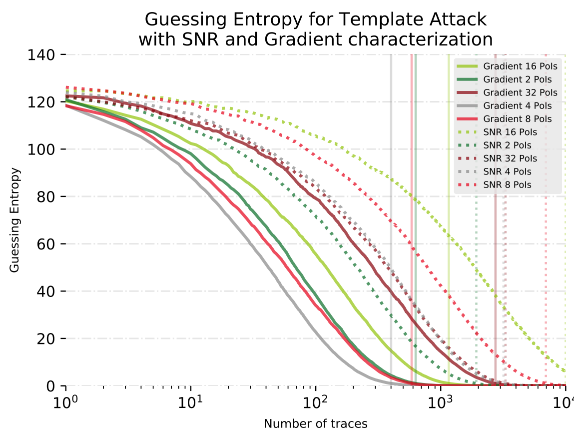

Fig. 7: Comparison of the guessing entropy for GV based and SNR based attacks.

As expected, the GE converges towards 0 when the extraction is made with GV. In addition, when comparing the SNR and the GV it may be remarked that for a same number of extracted points, the latter method always outperforms the former one, *i.e.* the guessing entropy converges faster towards 0 (Figure 7). This may emphasize that a trained CNN is able to carefully select the couples of points corresponding to the leakages of the mask and of the masked data among all the possible couples to get the best possible recombination.

[4] Obviously, this scenario is not realistic as if one has access to the mask during characterization, then the latter one is very likely to be also available during the profiling phase.

Conclusion

In this paper, we have theoretically shown that a method called Gradient Visualization can be used to localize Points of Interest. This result relies on two assumptions that can be considered as realistic in a Side Channel context.

Generally, the efficiency of the proposed method only depends on the ability of the profiling model to succeed the attack. In the case where countermeasures like masking or misalignment are considered, CNNs are shown to still build good pdf estimations, and thereby the Gradient Visualization provides a good characterization tool. In addition, such a visualization can be made for each trace individually, and the method does not require more work than needed to perform a profiling with CNNs leading to a successful attack.

We verified the efficiency of our proposed method on simulated data. It has been shown that as long as a Neural Network is able to have slightly better performance than randomness, it can localize points that contain the informative leakage.

On experimental traces, we have empirically shown that Gradient Visualization is at least as good as state-of-the-art characterization methods, in different cases corresponding to the presence or not of different countermeasures. Not only it can still localize Points of Interest in presence of desynchronization or masking but it has also been shown in the latter case that a CNN is able to select the optimal couples of Points of Interest in order to recombine the shares instead of just trying the best Points of Interest when separately characterizing the mask and the masked data leakages. To the best of our knowledge, this would not be possible with state-of-the-art characterization methods.

References

1. Sensitivity analysis - Wikipedia, `https://en.wikipedia.org/wiki/Sensitivity_analysis`
2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A system for large-scale machine learning. arXiv:1605.08695 [cs] (2016-05-27), `http://arxiv.org/abs/1605.08695`
3. Brier, E., Clavier, C., Olivier, F.: Correlation power analysis with a leakage model. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 16–29. Springer (2004)
4. Cagli, E., Dumas, C., Prouff, E.: Enhancing dimensionality reduction methods for side-channel attacks. In: Homma, N., Medwed, M. (eds.) Smart Card Research and Advanced Applications. pp. 15–33. Lecture Notes in Computer Science, Springer International Publishing (2016)
5. Cagli, E., Dumas, C., Prouff, E.: Convolutional neural networks with data augmentation against jitter-based countermeasures. In: International Conference on Cryptographic Hardware and Embedded Systems. pp. 45–68. Springer (2017)
6. Cagli, E., Dumas, C., Prouff, E.: Kernel discriminant analysis for information extraction in the presence of masking. In: Lemke-Rust, K., Tunstall, M. (eds.) Smart

Card Research and Advanced Applications. pp. 1–22. Lecture Notes in Computer Science, Springer International Publishing (2017)
7. Chari, S., Rao, J.R., Rohatgi, P.: Template attacks. In: International Workshop on Cryptographic Hardware and Embedded Systems. pp. 13–28. Springer (2002)
8. Choudary, M.O., Kuhn, M.G.: Efficient stochastic methods: Profiled attacks beyond 8 bits. In: Joye, M., Moradi, A. (eds.) Smart Card Research and Advanced Applications. pp. 85–103. Lecture Notes in Computer Science, Springer International Publishing (2015)
9. Choudary, O., Kuhn, M.G.: Efficient template attacks. In: Francillon, A., Rohatgi, P. (eds.) Smart Card Research and Advanced Applications. pp. 253–270. Lecture Notes in Computer Science, Springer International Publishing (2014)
10. Durvaux, F., Renauld, M., Standaert, F., van Oldeneel tot Oldenzeel, L., Veyrat-Charvillon, N.: Efficient removal of random delays from embedded software implementations using hidden markov models. In: Mangard, S. (ed.) Smart Card Research and Advanced Applications - 11th International Conference, CARDIS 2012, Graz, Austria, November 28-30, 2012, Revised Selected Papers. Lecture Notes in Computer Science, vol. 7771, pp. 123–140. Springer (2012). https://doi.org/10.1007/978-3-642-37288-9_9, https://doi.org/10.1007/978-3-642-37288-9_9
11. Eisenbarth, T., Paar, C., Weghenkel, B.: Building a side channel based disassembler. In: Gavrilova, M.L., Tan, C.J.K., Moreno, E.D. (eds.) Transactions on Computational Science X: Special Issue on Security in Computing, Part I, pp. 78–99. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2010). https://doi.org/10.1007/978-3-642-17499-5, https://doi.org/10.1007/978-3-642-17499-5_4
12. Gilmore, R., Hanley, N., O'Neill, M.: Neural network based attack on a masked implementation of AES. In: 2015 IEEE International Symposium on Hardware Oriented Security and Trust (HOST). pp. 106–111 (2015-05). https://doi.org/10.1109/HST.2015.7140247
13. Goodfellow, I., Bengio, Y., Courville, A.: Deep learning. Adaptive computation and machine learning series, MIT Press (2017)
14. Hardt, M.: Off the convex path, http://offconvex.github.io/
15. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv:1412.6980 [cs] (2014-12-22), http://arxiv.org/abs/1412.6980
16. Kocher, P., Jaffe, J., Jun, B.: Differential power analysis. In: Wiener, M. (ed.) Advances in Cryptology — CRYPTO' 99. pp. 388–397. Lecture Notes in Computer Science, Springer Berlin Heidelberg (1999)
17. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
18. LeCun, Y., Bengio, Y.: The handbook of brain theory and neural networks. chap. Convolutional Networks for Images, Speech, and Time Series, pp. 255–258. MIT Press, Cambridge, MA, USA (1998), http://dl.acm.org/citation.cfm?id=303568.303704
19. LeCun, Y., Bengio, Y., Hinton, G.: Deep learning. Nature **521**(7553), 436–444 (2015-05). https://doi.org/10.1038/nature14539, http://www.nature.com/articles/nature14539
20. Lerman, L., Bontempi, G., Markowitch, O.: A machine learning approach against a masked AES: Reaching the limit of side-channel attacks with a learning model. Journal of Cryptographic Engineering **5**(2), 123–139 (2015). https://doi.org/10.1007/s13389-014-0089-3

21. Maghrebi, H., Portigliatti, T., Prouff, E.: Breaking cryptographic implementations using deep learning techniques. In: Carlet, C., Hasan, M.A., Saraswat, V. (eds.) Security, Privacy, and Applied Cryptography Engineering. pp. 3–26. Lecture Notes in Computer Science, Springer International Publishing (2016)
22. Mangard, S., Oswald, E., Popp, T.: Power analysis attacks: revealing the secrets of smart cards. Springer (2007), OCLC: ocm71541637
23. Martinasek, Z., Dzurenda, P., Malina, L.: Profiling power analysis attack based on MLP in DPA contest v4.2. In: 2016 39th International Conference on Telecommunications and Signal Processing (TSP). pp. 223–226 (2016-06). https://doi.org/10.1109/TSP.2016.7760865
24. Montavon, G., Samek, W., Müller, K.R.: Methods for interpreting and understanding deep neural networks. Digital Signal Processing **73**, 1–15 (2018-02). https://doi.org/10.1016/j.dsp.2017.10.011, http://linkinghub.elsevier.com/retrieve/pii/S1051200417302385
25. Nagashima, S., Homma, N., Imai, Y., Aoki, T., Satoh, A.: DPA using phase-based waveform matching against random-delay countermeasure. In: 2007 IEEE International Symposium on Circuits and Systems. pp. 1807–1810 (2007-05). https://doi.org/10.1109/ISCAS.2007.378024
26. Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., Lerer, A.: Automatic differentiation in pytorch. In: NIPS-W (2017)
27. Picek, S., Samiotis, I.P., Heuser, A., Kim, J., Bhasin, S., Legay, A.: On the performance of deep learning for side-channel analysis. IACR Cryptology ePrint Archive **2018**, 4 (2018), http://eprint.iacr.org/2018/004
28. Prouff, E., Rivain, M., Bevan, R.: Statistical analysis of second order differential power analysis. IEEE Transactions on Computers **58**(6), 799–811 (2009-06). https://doi.org/10.1109/TC.2009.15, http://ieeexplore.ieee.org/document/4752810/
29. Prouff, E., Strullu, R., Benadjila, R., Cagli, E., Dumas, C.: Study of deep learning techniques for side-channel analysis and introduction to ASCAD database. IACR Cryptology ePrint Archive **2018**, 53 (2018), http://eprint.iacr.org/2018/053
30. Rivain, M., Prouff, E., Doget, J.: Higher-order masking and shuffling for software implementations of block ciphers. In: Clavier, C., Gaj, K. (eds.) Cryptographic Hardware and Embedded Systems - CHES 2009. pp. 171–188. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2009)
31. Shalev-Shwartz, S., Ben-David, S.: Understanding Machine Learning: From Theory to Algorithms. Cambridge University Press (2014). https://doi.org/10.1017/CBO9781107298019, http://ebooks.cambridge.org/ref/id/CBO9781107298019
32. Simonyan, K., Vedaldi, A., Zisserman, A.: Deep inside convolutional networks: Visualising image classification models and saliency maps. arXiv:1312.6034 [cs] (2013-12-20), http://arxiv.org/abs/1312.6034
33. Springenberg, J.T., Dosovitskiy, A., Brox, T., Riedmiller, M.: Striving for simplicity: The all convolutional net. arXiv:1412.6806 [cs] (2014-12-21), http://arxiv.org/abs/1412.6806
34. Standaert, F.X., Archambeau, C.: Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In: Oswald, E., Rohatgi, P. (eds.) Cryptographic Hardware and Embedded Systems – CHES 2008. pp. 411–425. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2008)

35. 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. Lecture Notes in Computer Science, Springer Berlin Heidelberg (2009)
36. van Woudenberg, J.G.J., Witteman, M.F., Bakker, B.: Improving differential power analysis by elastic alignment. In: Proceedings of the 11th International Conference on Topics in Cryptology: CT-RSA 2011. pp. 104–119. CT-RSA'11, Springer-Verlag (2011), http://dl.acm.org/citation.cfm?id=1964621.1964632
37. Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., Torralba, A.: Learning deep features for discriminative localization. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2921–2929 (2016-06). https://doi.org/10.1109/CVPR.2016.319

A Profiling Attacks

As the model is aiming at approximating the conditional pdf, a Maximum Likelihood score can be used for the guessing:

$$\mathbf{d}_{S_a}[k] \triangleq \sum_{i=1}^{N_a} \log\left(\mathbf{y}_i[z_i]\right) \text{ where } z_i = \mathbf{C}(p_i, k). \tag{8}$$

Based on these scores, the key hypotheses are ranked in a decreasing order. Finally, the attacker chooses the key that is ranked first (resp. the set of first o ranked keys). More generally, the *rank* $g_{S_a}(k^\star)$ of the correct key hypothesis $k^\star$ is defined as:

$$g_{S_a}(k^\star) \triangleq \sum_{k \in \mathcal{K}} 1_{\mathbf{d}_{S_a}[k] > \mathbf{d}_{S_a}[k^\star]}. \tag{9}$$

Remark 1. In practice, to compute $\mathrm{GE}(N_a)$, sampling many attack sets may be very prohibitive in an evaluation context, especially if we need to reproduce the estimations for many values of N_a; one solution to circumvent this problem is, given a validation set S_v of N_v traces, to sample some attack sets by permuting the order of the traces into the validation set. $\mathbf{d}_{S_a}$ can then be computed with a cumulative sum to get a score for each $N_a \in [|1, N_v|]$, and so is $g_{S_a}(k^\star)$. While this trick gives good estimations for $N_a \ll N_v$, one has to keep in mind that the estimates become biased when $N_a \to N_v$. This problem also happens in Machine Learning when one lacks data to validate a model. A technique called *Cross-Validation* [31] enables to circumvent this problem by splitting the dataset into q parts called *folds*. The profiling is done on $q-1$ folds and the model is evaluated with the remaining fold. By repeating this step q times, the measured results can be averaged so that they are less biased.

B Study of an Optimal Model

Informally, Assumption 1 tells that the leaking information is non-uniformly distributed over the trace $\mathbf{X}$, *i.e.* only a few coordinates contain clues about the attacked sensitive variable. Assumption 1 has been made in many studies such as [4]. Depending on the countermeasures implemented into the attacked device, the nature of $\mathcal{I}_Z$ may be precised. Without any countermeasure, and supposing that the target sensitive variable only leaks once, Assumption 1 states that $\mathcal{I}_Z$ is only a set of contiguous and constant coordinates, regardless the input traces.

Adding masking will split $\mathcal{I}_Z$ into several contiguous and fixed sets whose number is equal to the number of shares in the masking scheme (or at least equal to the number of shares if we relax the hypothesis of one leakage per share). For example if M (resp. $Z \oplus M$) denotes the mask (resp. masked data) variable leaking at coordinate t_1 (resp. t_2), then M and $X[t]$ with $t \neq t_1$ are independent (resp. Z and $X[t]$ with $t \neq t_2$ are independent). The conditional probability $\Pr[Z = z|\mathbf{X} = \mathbf{x}]$ satisfies:

$$\Pr[Z = z|\mathbf{X} = \mathbf{x}] = \sum_{m} \Pr[Z \oplus M = z \oplus m|\mathbf{X}[t_1] = \mathbf{x}[t_1]]\Pr[M = m|\mathbf{X}[t_2] = \mathbf{x}[t_2]] \quad (10)$$

Adding de-synchronization should force $\mathcal{I}_Z$ to be non-constant between each trace.

Likewise, Assumption 2 is realistic because it is a direct corollary of a Gaussian leakage model for the traces [7,9]. Such an hypothesis is common for Side Channel Analysis [7]. It implies that $\mathbf{x} \mapsto \Pr[\mathbf{X} = \mathbf{x}|Z = z]$ is differentiable and:

$$\nabla_{\mathbf{x}} \Pr[\mathbf{X} = \mathbf{x}|Z = z] = \Sigma_z^{-1}(\mathbf{x} - \mu_z)\Pr[\mathbf{X} = \mathbf{x}|Z = z] \quad (11)$$

where μ_z and Σ_z^{-1} respectively denote the mean vector and the covariance matrix of the normal probability distribution related to the target sensitive value hypothesis z. Then, from Bayes' theorem, (11) and the basic rules for derivatives computation, it gives an analytic expression of $\nabla_{\mathbf{x}} F^*(\mathbf{x})$, thereby proving that F^* is differentiable with respect to the input trace.

C Neural Networks

Neural Networks (NN) are nowadays the privileged tool to address the classification problem in Machine Learning [19]. They aim at constructing a function $F\colon \mathcal{X} \to \mathcal{P}(\mathcal{Z})$ that takes data $\mathbf{x}$ and outputs vectors $\mathbf{y}$ of scores. The classification of $\mathbf{x}$ is done afterwards by choosing the label $z^* = \mathrm{argmax}_{z\in\mathcal{Z}}\, \mathbf{y}[z]$, but the output can be also directly used for soft decision contexts, which corresponds more to Side Channel Analysis as the NN outputs on attack traces will be used to compute the score vector in (8). In general F is obtained by combining several simpler functions, called *layers*. An NN has an *input layer* (the identity over the input datum $\mathbf{x}$, an output layer (the last function, whose output is the scores vector $\mathbf{y}$ and all other layers are called *hidden* layers. The nature (the number and the dimension) of the layers is called the *architecture* of the NN. All the parameters that define an architecture, together with some other parameters that govern the training phase, have to be carefully set by the attacker, and are called *hyper-parameters*. The so-called *neurons*, that give the name to the NNs, are the computational units of the network and essentially process a scalar product between the coordinate of its input and a vector of *trainable weights* (or simply *weights*) that have to be *trained*. We denote θ the vector containing all the trainable weights. Therefore, for a fixed architecture, an NN is completely parameterized by θ. Convolutional Neural Networks (CNN) implement other operations, but can be rewritten as regular NN with specific constraints on the weights [18]. Each layer processes some neurons and the outputs of the neuron evaluations will form new input vectors for the subsequent layer.

The ability of a Neural Network to approximate well a target probabilistic function F^* by minimizing a loss function on sampled training data with Stochastic Gradient Descent is still an open question. This is what we call the *mystery of Deep Learning*. It theoretically requires a huge quantity of training data so that the solution obtained by loss minimization generalizes well, though it empirically works with much less data. Likewise, finding the minimum with Stochastic Gradient Descent is theoretically not proved, but has been empirically shown to be a good heuristic. For more information, see [14]. Indeed, though it raises several theoretical issues, it has been empirically shown to be efficient, especially in SCA with CNN based attacks [5,27].

D What about Other Visualization Methods?

The idea to use the derivatives of differentiable models to visualize information is not new. Following the emergence of deep convolutional networks, [32] have first proposed a so-called *Sensitivity Map* for image recognition. The idea was more motivated by the fact that such a map can be computed for free. Nevertheless, this method did not fit well the explainability problem for image recognition. Actually [24] states that visualizing the gradient only tracks an explanation to the variation of a final decision ($F(\mathbf{x})$ in our context), and not directly the decision itself.[5]

In a computer vision context, data are highly structured (e.g. pixels giving edges, giving primar shapes, giving more sophisticated ones, forming faces). The decision is then highly diluted among lots of pixels, and the decision surface might be locally flat, though very high. Hopefully, Assumption 1 states that in a SCA context it is reasonable to consider that the information is very localized. Actually the less PoIs we have, the more likely a small variation on a relevant feature is to impact the decision. That is why we are in a particular case where looking at the output sensitivity may be more interesting than other visualization methods.

E Experimental Results

E.1 The Jacobian matrix

In this appendix, we present the Jacobian matrix visualization, equivalent to the GV. It shows, in addition, that some target values seem more sensitive, especially those whose Hamming weight is shared by only few other values (so it gives clues about how the traces leak sensitive information). Figure 8 (top) shows such a matrix in application context (Exp.1) as described in Section 6, while Figure 8 (bottom) shows the Jacobian matrix corresponding to the application context (Exp.2). Figure 9 shows the SNR computed on de-synchronized traces.

[5] To this end, they propose a visualization method called *Layerwise Relevance Propagation* (LRP). We tested it without relevant results.

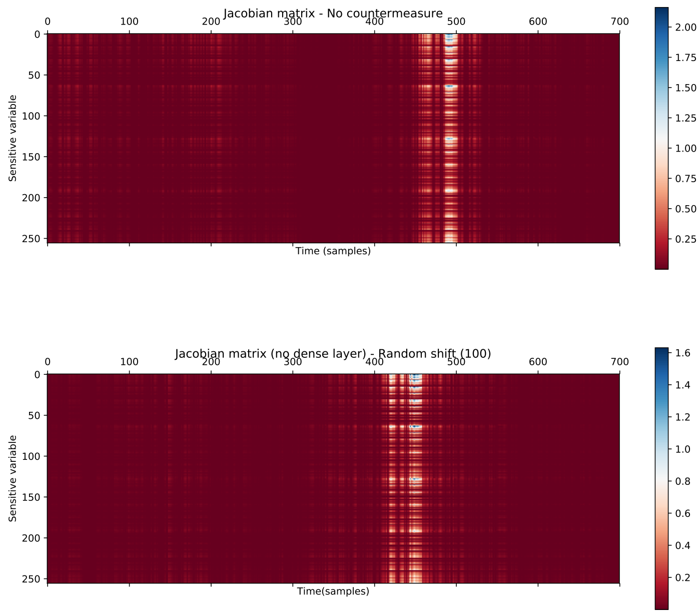

Fig. 8: Jacobian matrix for the best models in application contexts (Exp.1)(top) and (Exp.2) (bottom).

E.2 Comparison with a Principal Component Analysis

PCA has been widely used as a dimensionality reduction method. See for example [4,31] for more details. This idea is to project the data into a smaller linear subspace, namely the eigenspaces of the empirical covariance matrix corresponding to the N highest eigenvalues, where $N \ll D$ denotes the new dimension of the projected traces.

This method often meets success thanks to its simplicity and its absence of requirement in prior knowledge of the problem. That is why it is commonly used in presence of masking, where other methods such as SNR or Linear Discriminant Analysis (LDA) become inefficient. However, such a success is not guaranteed, as discriminative features may be at coordinates of lower variance. The consequence is that there is no guarantee that the most informative components are in the very first principal ones [4].

As we compared GV with SNR for characterization, we may compare GV with PCA in the context of dimensionality reduction. The question is roughly the same: how to project the input traces to a lower dimension space in order to make a Gaussian Template Attack feasible? In that sense, Gradient Visualization is reduced to project the traces on only few coordinates, while PCA projects the traces on linear subspaces.

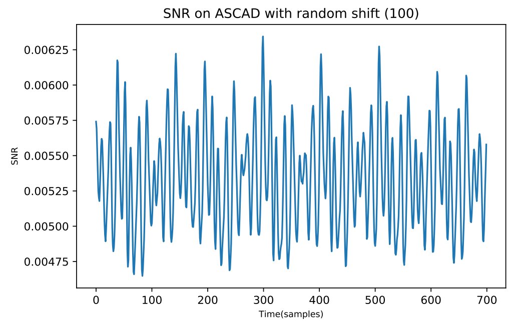

Fig. 9: The SNR in the case where de-synchronization is considered.

We proceeded the same experiments as done in Section 6.4, by replacing dimensionality reduction from the SNR with a PCA. The main difference is that the mask is not assumed to be known for the PCA, and the first components are extracted without following a specific rule. The results are given in Figure 10. Both PCA and GV obtain roughly the same results. However, PCA is more likely to not converge when the reduced dimension is very low, *i.e.* under 8, whereas GV always makes the Guessing Entropy converge towards 0 within the 10,000 validation traces.

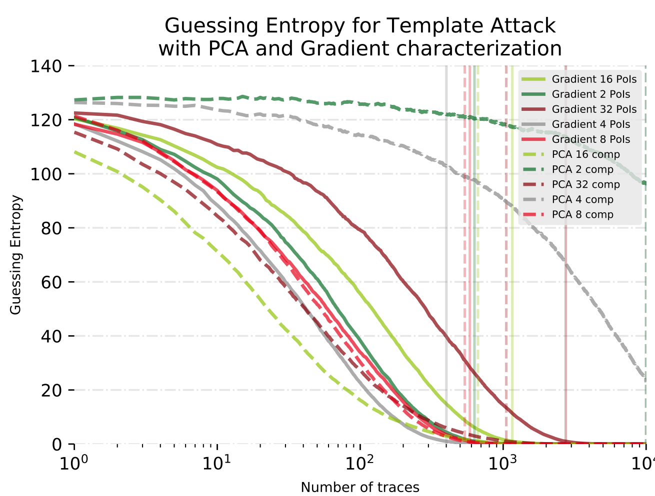

Fig. 10: Comparison of the guessing entropy for GV based and PCA based attacks.