UCLouvain

École polytechnique de Louvain

# How to fool Machine Learning based side-channel attacks

Author: **Charles-Henry Bertrand Van Ouytsel**
Supervisor: **François-Xavier Standaert**
Readers: **Gaëtan Cassiers, Christophe De Vleeschouwer**
Academic year 2018–2019
Master [120] in Electrical Engineering

# Abstract

Side-channel attack is defined as any attack taking advantage of information leaking from the physical implementation of a cryptographic system, like its power consumption, to retrieve the secret it hides. Several methods to exploit these information have been proposed in literature. Recently, new methods involving machine techniques have been presented. This work introduces a setup to mislead these ML-techniques in a side-channel context. In this setup, training data are composed of two parts : one part leaking information inside its mean value and controlled by a parameter $\delta_1$ and a second part leaking information in its variance. During the attack phase, a slight change of the parameter $\delta_1$ into $\delta_2$ leads ML-techniques to retrieve the wrong key as these methods seems to prior the simplest model based on the mean over the second based on the variance. A parametric study is proposed, followed by implementation of this setup on FPGA. These results highlight the need to be in similar conditions between training and testing when using ML-techniques. Exploiting such differences and the lack of diversity of a training set in this way could be an interesting track to defeat ML-techniques in many others applications.

# Acknowledgements

First of all, I would like to thanks my advisor Prof. François-Xavier Standaert for his support, advice, availability and enthusiasm during our meetings. He gave me the opportunity to work on an incredible and original subject.

I also would like to thanks Gaëtan Cassiers for his advice and feedback on this master thesis. As well as Olivier Bronchain for his explanations about side-channel attacks as well as his help to obtain traces used for the FPGA part of this master thesis.

Finally I would like to thanks all my friends, my family and my girlfriend for their support during this thesis as well as these five years at UCLouvain.

# Contents

# Chapter 1

# Introduction

Today, cryptographic algorithms are present everywhere: phones, laptops, smartcard, camera or any type of connected device. They are utilized to protect your privacy, your data, your bank account,... Some of these devices use software implementation of these algorithms when others use dedicated hardware implementation to improve efficiency. Where cryptography makes interest in ensuring that an algorithm is secure, embedded security is focused on the security of the physical device running the algorithm itself. Indeed, information from these devices can leak in various ways like power consumption or electromagnetic field, these ways are called *side-channels*. An attacker could use the information contained in these leakages to compromise the device and its secret (like its cryptographic key).

Several methods already exist to exploit side-channels leakages (template attack, DPA, CPA, MCP-DPA,...). Moreover, some new methods implying machine learning (ML) techniques like multilayer perceptron, random forest or SVM, have been demonstrated to be quite efficient and easy to put in place due to their capacity to learn a model from leakages by themselves. It results, as in many other areas, that machine learning is used nearly "blindly", without understanding which model is built and how the leakage is exploited.

**State of the art** Recently some methods have been presented to fool machine learning techniques. *Adversarial inputs* are crafted input developed to evade its good classification (for instance, malware could be classified as cleanware) by slightly modifying its features. *Data poisoning* involves feeding training data with specific data in order to shift the boundary of the constructed model in the adversary's advantage (like making some competitors' data being misclassified by a model). These techniques have already been applied such as in image recognition tasks ( see for example [1] or [9] for more details about these techniques and some of their applications). However, up to our knowledge, no example has been provided for side channel attack.

**Contribution** This work explains and studies practical settings introduced in a side channel attack context to mislead machine learning techniques. By feeding these techniques with specific data including information from a simple but wrong model and from a more complex but correct one, we show how machine learning could be fooled if used blindly. This way of misleading ML techniques could be implemented in real circuits as a basic protection against ML. These crafted data will also be used with classical side channel attack techniques to demonstrate the benefits of the knowledge of the model used. Resulting protection against machine learning methods could be seen as a new type of attack on ML similar to a kind of data poisoning.

This master thesis will begin by introducing, in chapter 1, useful background about side channel attack and machine learning techniques applied in this context. After, in chapter 2, settings used for the simulation will be exposed (ML parameters, profile of the attacker, metrics used,...) and the actual setup imagined in order to fool machine learning based attack will be presented. In chapter 3, results of the proposed countermeasure as well as a deep study of its parameters and their influence on ML-methods will also be exposed. Chapter 4 will conclude by testing our setup on a real-world implementation on FPGA to confirm simulated experiments.

# Chapter 2

# Background

This section introduces the background information required to fully understand this work.

## 2.1 Side channel attack

*Side channel attacks* (SCA) is a term used to describe any attack based on information which leaks from the physical implementation of a cryptosystem. The expression *side channel* refers to any physical way independent of the mathematical definition of the algorithm itself which can lead to a leakage of internal variables of the algorithm. Examples of such channels are power consumption, electromagnetic fields, timing information,... In this master thesis, the focus was placed on power analysis attack (PAA) where the power consumption of the device running the crypto algorithm was recorded. The cryptographic key used by the utilised device was then deduced from those records.

## 2.2 Power analysis attack

In 1999, Kocher et al. published an article ([12]) in which they showed how dangerous a power analysis attack could be. For example, they demonstrated that monitoring the power consumption of a smartcard was often enough to completely reveal the secret key hidden in it. Leakages exploited to recover the key are independent of the cryptographic algorithm used but come from the physical implementation of the circuit itself.

These attacks use the chosen-plaintext attack model. In this model, the attacker $\mathcal{A}$ is given access to an encryption E /decryption D oracle which can be used to encrypt/decrypt any pair of plaintext-ciphertext (p-c). The analogy could be made

between this oracle and a black box cryptosystem with an embedded secret key k that returns a ciphertext c given a plaintext $cE(p) = Enc(k, p)$ or a plaintext p given a ciphertext c $D(c) = Dec(k, c) \rightarrow p$. $\mathcal{A}$ uses the knowledge of the pair (p,c) and the associated leakage in order to try to recover the key k.

### 2.2.1 Origin of the leakage

A large part of today's digital circuits is based on CMOS gates. One of the component of their power consumption is due to the charge and discharge of a load capacitance $C_L$ when the logic state of the circuit changes. This phenomenon can be observed in figure 2.1. This power consumption can be calculated as : $P = C_L V_{DD} P_{0\rightarrow1} f$ where $P_{0\rightarrow1}$ is the switching activity which corresponds to the probability of a transition of the logic state of the circuit from 0 to 1, $f$ is the frequency of operation and $V_{DD}$ is the supply voltage.

This power consumption is generally very interesting because of its direct dependency with the internal variable manipulated by the device (i.e directly linked to the logic state of the CMOS gate). Different internal variables manipulated imply different logic states and so different power consumptions. Moreover, in CMOS devices, when measuring consumption, higher peaks appear during the charge of the capacitance $C_L$ (i.e. $0 \rightarrow 1$ transition event) than during the discharge. It constitutes another element linked to internal variables which directly impact the power consumption. More information about different power consumptions and how they could be exploited could be find in [5], [16] and [20].

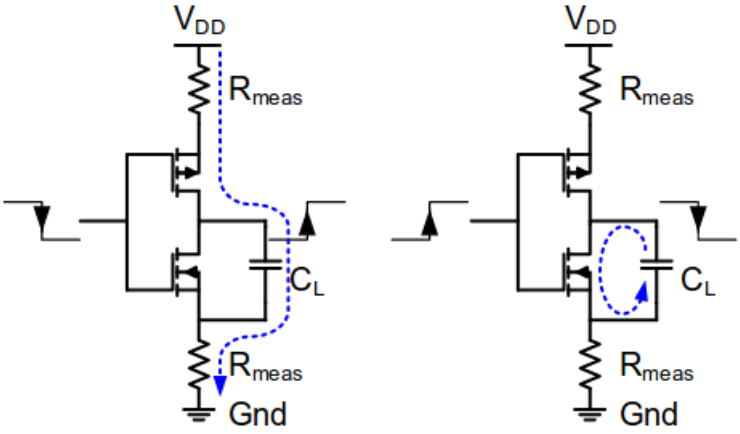


Figure 2.1: Illustration of the charge and discharge of the load capacitance in a CMOS inverter leading to a dynamic power consumption. Image from [20].

In this work, the term *trace* will refer to the power consumption measurement done on a device during the execution of its cryptographic algorithm. Moreover, leakage of order n will be defined as a leak of information in statistical moment of order n (e.g leakage of order 1 means that the information is hidden in the mean value of the leakage)

### 2.2.2 Hamming weight and hamming distance model

Consequently to the above-mentioned observations about the power consumption, adversaries have derived leakage models to imitate power consumption of devices.

**Hamming weight model** assumes that, when a value $x_0$ is manipulated by a device, the side-channel leakages are correlated with the hamming weight of this value $H_W(x_0)$ (hamming weight means the number of bits set to 1 in binary representation). Link could be done with previous observations about dynamic power consumption : the number of transitions $0 \rightarrow 1$ are equals to the hamming weight of $x_0$ if the preceding value contained in CMOS gate was 0.

**Hamming distance model** assumes that when a value $x_0$ contained in a CMOS device is switched to a value $x_1$, the side-channel leakages are correlated with the hamming distance of these value, namely $H_D(x_0, x_1) = H_W(x_0 \oplus x_1)$. Again, it represents the number of transitions occurring.

These two models are quite simple as they make some important assumptions : first, they assume no difference between $0 \rightarrow 1$ and $1 \rightarrow 0$ transitions (while there is a difference as shown in previous section). They also assume that every bit in an implementation makes the same contribution to the overall power consumption. Nevertheless, they are quite commonly used by attackers to map data value manipulated in the device to part of the execution's consumption. More details are available in [16].

## 2.3 Block cipher

Block cipher is one of the most important building blocks in modern cryptography. It provides secure communication between two parties using symmetric encryption. This means that the parties possess the same key k used for encryption and decryption. In this section, block cipher will be briefly explained before presenting AES, a famous block cipher.

### 2.3.1 Block cipher

Block cipher is used for secure communication between two parties, so that the exchanged message, called the plaintext, can not be retrieved by someone trying to spy on them. To this purpose, the sender of the message encrypts its plaintext using the block cipher and its secret key k to create the encoded message, the ciphertext. From this ciphertext, the receiver computes the original plaintext by inverting the block cipher using the known used key. Without knowing this key, it is supposed to be unfeasible to retrieve the plaintext.

According to [11], block ciphers are designed to be secure instantiation of a pseudorandom permutation with fixed key and block length $\mathcal{F} : \{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$. In other words, there exists $2^n$ possible permutations $\mathcal{F}_k : \{0,1\}^n \rightarrow \{0,1\}^n$. Choosing a key is equivalent to choose one of these permutations to encrypt/decrypt the plaintext.

**General structure** Block cipher must behave like a pseudorandom permutation. This implies that changing a single bit at the input of $\mathcal{F}_k$ (.) must affect all bits of the output. In order to build random looking permutations for block ciphers, Shannon introduced the confusion-diffusion paradigm. Substitution-permutation network (SPN) is a direct implementation of this paradigm and serves as a basis for typical block ciphers. It is built by repeating this series of operations in multiple rounds ( see [11], p. 204-206).

- *Key mixing*: Set $x := x \oplus k$
- *Confusion*: Set $x := f_0(x_0)||f_1(x_1)||...||f_{15}(x_{15})$
- *Diffusion*: Set $x := P(x)$ , where P permutes the bits of x.

The concept of *diffusion* is to build a random permutation $\mathcal{F}(.)$ on long strings (i.e $x\{0,1\}^{128}$) based on smaller known permutations $f(.)$ (working on $x\{0,1\}^8$) also called SBox. All the individual small outputs of these SBox are concatenated to form the output. However, with this scheme, changing an input byte $x_0$ only impacts the first byte of the output. To spread the change of an input byte on all output bytes, *diffusion* is used. It mixes bits before repeating the others operations. Finally, *key mixing* is used to have a keyed permutation.

## 2.4 AES

The Advanced Encryption Standard (AES), also originally known as Rijndael, is a specification for the encryption and decryption of electronic data established by the

U.S. National Institute of Standards and Technology (NIST) in 2001 to succeed to DES. This block cipher operates with keys of 128, 192 or 256 bits. In this work, focus will be made on the 128 bits version.

AES implements confusion and diffusion principle thanks to several building blocks that are repeated over 10 rounds[1]. Each round makes use of a round key derived from the input 128 bits key thanks to an expansion key algorithm.

AES operates on 4x4 arrays of bytes ($b_0$,$b_1$,$b_2$,..,$b_{15}$) called state :

$$\text{state} = \begin{bmatrix} b_0 & b_4 & b_8 & b_{12} \\ b_1 & b_5 & b_9 & b_{13} \\ b_2 & b_6 & b_{10} & b_{14} \\ b_3 & b_7 & b_{11} & b_{15} \end{bmatrix}$$

The initial state is constructed with the plaintext encrypted ($b_0$ corresponds to the first plaintext and so on..)

### 2.4.1 Building block of AES

The building blocks of the AES are based on arithmetic in $\mathcal{GF}(2^8)$ as defined in [7]. A quick overview will be given here under :

**Add Round Key :** The add round key is a bitwise XOR between the actual state and the round key. This block ensures that the key is required to reverse AES encryption.

**ShiftRows :** In this block, rows of the actual state are shifted according to the figure 2.2. This step, part of the linear layer of the AES, is just a wire crossing and contributes to the *diffusion* of the block cipher.

**MixColumns :** An other part of the linear layer of the cipher, MixColumns consists in a matrix multiplication in $\mathcal{GF}(2^8)$. The entry bytes of a column constitutes a four terms polynomials $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ and is multiplied by another fixed polynomial $c(x) = \textbf{'03'}x^3 + \textbf{'01'}x^2 + \textbf{'01'}x + \textbf{'02'} \mod x^4 + 1$. This transformation can be written as :

$$\text{mixcolumns(state)} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ a_4 \end{bmatrix}$$

More details can be found in [7].

[1] 10 rounds for 128 bits keys, 12 for 192 bits keys and 14 for 256 bits keys

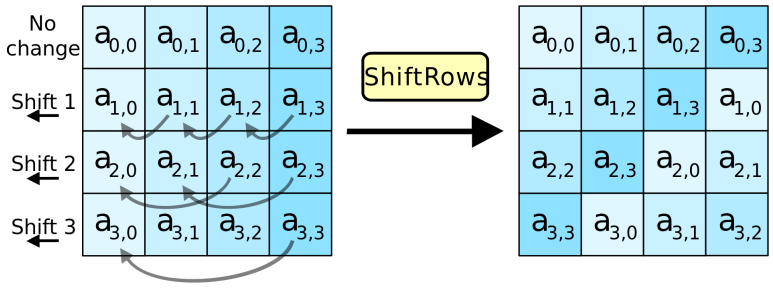


Figure 2.2: Shift rows operation, By User:Matt Crypto - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1118782

**SubBytes :** This block is the only non-linear block of the AES block cipher and is used for *confusion*. Each byte of the state $a_{i,j}$ undergoes an inversion in $\mathcal{GF}(2^8)$ followed by the same affine transformation. Generally this operation is precomputed for all kind of different input values to create *substitution box* also called *SBox*. There is a bijection (i.e one to one relation) between the input byte of the SBox and its output.

## 2.5 Countermeasures

Power analysis attack works because there exists a dependency between the intermediate value of the executed cryptographic algorithm and the power consumption of the device. The aim of countermeasures is to make them independent. There exist two principal kinds of countermeasures: Hiding and masking.
The concept of hiding is to make the power consumption at each cycle either random or constant by implementing specific logic gates. Attacks thus become more difficult in practice, but still feasible because there will always be a part of power consumption which is data-dependent. The other method, masking was used in this work. More pieces of information can be found in [16].

### 2.5.1 Masking

Masking randomises all intermediate values that are manipulated by the algorithm in order to make the consumption independent of the manipulated value.

Each intermediate value $v$ is concealed by a random value $m$ called the mask. Mask $m$ is generated by the device and changes from one execution to another. $v$ and $m$ can be concealed by exclusive xoring (boolean masking) or by arithmetic operation (arithmetic masking). In this master thesis, boolean masking was used. In this case, the masked intermediate value is defined as $v_m = v \oplus m$.

When both $v_m$ and $m$ are available, $v$ could be computed. Masking thus corresponds to a secret sharing scheme using two shares ($v_m$ and m). Secret sharing scheme could be extend by applying multiple masks on one intermediate value. For n masks, it gives :

$$v = v_1 \oplus v_2 \oplus v_3 \oplus v_4 \oplus v_5 \oplus v_6 ... \oplus v_n$$

When an implementation is protected by an n-shares scheme, attacks are prevented up to a $n^{th}$ order attack. However, keeping track of multiples shares increases the cost of implementation. More memory is needed to save different shares values and more computation time is needed to calculate them. That is why masking with two shares is generally used. Protection against higher order attacks is then achieved by combining masking and hiding.

## 2.6 SCA attacks studied in this document

All methods presented below aim at recovering which intermediate value Z is manipulated at a specific point of the encryption algorithm and from this value deducing the key $k^*$ by using Bayes' law. For this purpose, in the profiling phase, a model is built for each Z. During the attack phase, the adversary uses this model to compute a probability[2] for each of these intermediate values given traces $(l_i)_{1 \leq i \leq n}$ he has recorded. Finally, to recover the correct key, he computes the maximum likelihood (i.e for each key possibility, an intermediate value is deduced from the known plaintext and the corresponding probability is associated with the likelihood of the key)

$$\Pi_{j=1}^{n} P(L|Z = \hat{z}_j)$$

Note : to ease calculations, it is common to use the maximum log likelihood :

$$\Sigma_{j=1}^{n} log(P(L|Z = \hat{z}_j))$$

### 2.6.1 Moments-Correlating profiled DPA

MCP-DPA (see [17]) is a SCA method based on a chosen statistical moment. Let X be a random variable. The dth-order raw statistical moments are defined as $M_x^d = E(X^d)$ where $E(.)$ is the expectation operator. The dth-order central moments are defined as $CM_x^d = E((X - \mu)^d)$ with $\mu = E(X)$,the mean and $\sigma^2 = E(((X - \mu)^2)$ the variance.

[2]In the case of MCP-DPA, he does not compute probability but correlation.

In the profiling phase, an estimation of one of the previous statistical moments is done for each targeted intermediate value Z from a set of leakage with known key k and plaintext p. For example, estimating the mean for the output of the first SBox, $z = SBox(k \oplus p)$ gives $\hat{\mu}_{p,k} = [\mu_1, \mu_2, ..\mu_z]$ where $\mu_z$ denotes the mean value of the leakage for the target value z. The vector $M^d_{p,k}$ (i.e in this example $\hat{\mu}_{p,k}$) obtained for each Z is used to recover the correct key. The key is selected according to :

$$\hat{k} = argmax_{k^*} \rho(M^d_{k^*,p}, (L_{p,k})^d)$$

where $L_{p,k}$ is the set of leakages used for the attack with a known plaintext p and the unknown key k. These leakages are used to approximate the targeted statistical moment. The Pearson's correlation between these moments and estimations calculated during the profiling is then evaluated, permuting the estimation vector according to a key hypothesis. The best correlation obtained is the key guess of the method for the chosen moment.

The power of this method is to choose which moment will be attacked to recover the key. If the target implementation is known to use a masking scheme, attacker can directly target statistical moment not protected by the masking and try to recover the key.

In this document, MCP-DPA on order 1 refers to MCP-DPA targeting the mean of the leakage (first statistical moment) and MCP-DPA on order 2 refers to MCP-DPA targeting the variance of the leakage (second central moment).

### 2.6.2 Template attack

Template attack is a well-known SCA considering the Gaussian assumption. It consists first in a profiling phase where a set of traces and their corresponding intermediate values are used to estimate the probability density function (pdf) $f(L|Z = z)$ of the leakage L of the device when manipulating the target value Z. Traces used to estimate this pdf use random plaintext and random keys. Only the intermediate value present in the trace is important. Under the Gaussian assumption, this pdf is estimated by a normal distribution:

$$f(L|Z = z) \simeq \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(x-\mu)^2}{2\sigma^2}}$$

where $\mu$ is the mean and $\sigma$ the standard deviation. After having built these templates, they can be used to attack the targeted device with a new set of traces $(l_i)_{0 \leq i \leq n}$ for which the key and so the $z_i$ value is unknown. The attack works as follows : a key hypothesis $\hat{k}$ is chosen and allow to deduce a prediction $z_i$. From

this prediction and the leakage, a probability is obtained thanks to the template. The process is repeated for each key hypothesis and each leakage. Probabilities obtained for each key are finally combined to obtain the more likely key.

### 2.6.3 Multivariate template attack

The multivariate template is an extension of the classical template attack. It relies on the use of multiple points for the creation of the template. In a multivariate attack, instead of estimating a single variance $\sigma$, the whole matrix of covariances is evaluated. For example, considering 3 points of interest in the trace ($Z_1$, $Z_2$, $Z_3$), covariance matrix is calculated as :

$$\mathbf{\Sigma} = \begin{bmatrix} Var(Z_1) & Cov(Z_1, Z_2) & Cov(Z_1, Z_3) \\ Cov(Z_2, Z_1) & Var(Z_2) & Cov(Z_2, Z_3) \\ Cov(Z_3, Z_1) & Cov(Z_3, Z_2) & Var(Z_3) \end{bmatrix}$$

The mean value for each of the random variables considered is also required:

$$\mu = \begin{bmatrix} \mu_{Z1} \\ \mu_{Z2} \\ \mu_{Z3} \end{bmatrix}$$

Using multiple variables and under the Gaussian assumption, the pdf for n random variables ($Z = [Z_1, Z_2, Z_3, ..., Z_N]^T$) becomes :

$$f(L|Z) = \frac{1}{\sqrt{(2\pi)^n |\mathbf{\Sigma}|}} e^{-((\mathbf{x}-\mu)^T \mathbf{\Sigma}^{-1} (\mathbf{x}-\mu)/2}$$

## 2.7 Machine learning based attack

Recent publications have investigated new profiled attacks based on machine learning and even deep learning techniques, see [6] , [13] or [14]. In this work, focus will be made on *supervised* machine learning techniques.

*Supervised learning* refers to techniques involving a labeled training dataset that is given to the learning algorithm to build a model. After the learning phase, new input samples could be given to the created model to output the most likely output. These techniques includes *multilayer perceptron*, *random forest* or *Support vector machine*.

These ML-based attacks make no assumption on the data distribution and build a model from the raw data contrary to the template attack and its Gaussian assumption[3].

### 2.7.1 Multilayer perceptron

Before explaining the multilayer perceptron, let's introduce the *perceptron*. The perceptron is a linear classifier and represents the simplest neural network model. An n-inputs $(x_1, x_2, .., x_n)$ perceptron with one output $o$ is defined by its n weights $(w_1, w_2, .., w_n)$, its bias $w_0$ and its activation function $f$ (generally f is defined as the Heaviside function in single perceptron and as a sigmoid function in multilayer perceptron).An illustration of the perceptron is given in figure 2.3.

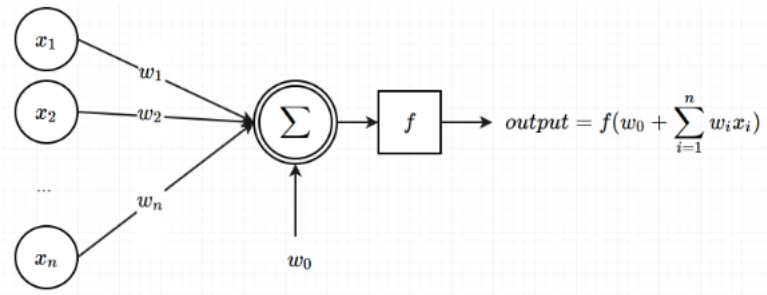


Figure 2.3: Representation of a perceptron, figure from [14]

The output of the perceptron is calculated as

$$o = f(w_0 + \sum_{i=1}^{n} w_i x_i)$$

During the training phase, weights are initialized at zero or randomly to some small value. Then thanks to a learning algorithm, weights are optimised in order to minimize a loss function (which is a function quantifying the error between the prediction of the algorithm and the actual ground truth, for instance, the mean squared error). One way to achieve this goal is to apply the *gradient descent algorithm* to find optimal weights.

A Multilayer Perceptron (MLP) is a specific combination of perceptrons allowing to build more complex classifiers. Fig 2.4 illustrates its typical architecture. The input is propagated from the left to the right and each unit (perceptron) of a layer is connected to units of the previous layer. Here, each neuron of a layer is

[3]In practice, the Gaussian assumption stays a fairly realistic assumption.

connected to every neurons of the previous layer, this architecture is thus called a fully connected network. Two main parameters control the multilayer perceptron model : the number of layers and the number of neurons per layer. In the figure 2.4, we can observe three types of layer :

- **Input Layer**: This layer makes the intermediate between the input data and the first hidden layer. The output of this layer is just the input data itself in the traditional model.
- **Hidden layers**: these layers allow to introduce non linearity in the model to fit non-linear separable datasets. According to the complexity of the targeted problem, the number of hidden layers and the number of neurons have to be adjusted. Using too much neurons could lead to the overfitting[4] of the model, where not enough neurons could fail to create a good model for the dataset. The choice of this parameter is thus essential.
- **Output layer**: It is the last layer of the network. Outputs of its neurons map directly to labels which have to be predicted. Here these labels are intermediate values targeted by the side-channel attack.

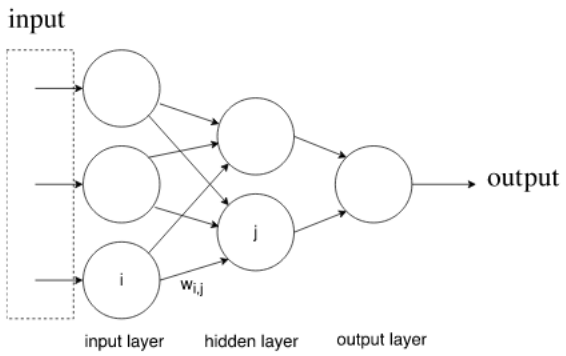


Figure 2.4: Multilayer perceptron architecture, figure from [14]

The training of the multilayer perceptron (corresponding to the profiling phase of the SCA), consists in the learning of weights minimising the loss function for each layer. For this purpose, the back-propagation algorithm is used. Weights are modified by computing the derivative of the loss function with respect to each weight one layer after another and by applying the following modification :

$$\Delta w_{ij} = -\frac{\partial E}{\partial w_{i,j}}$$

[4]Overfitting occurs when a model fits too perfectly with the training dataset and fails to generalise to other data due to too specific weights.

where E is the loss function and $w_{i,j}$ is the weight associated to the connection between neurons of indice i and j.

### 2.7.2 Decision tree and random Forest

A decision tree [19] is a classification model using binary rules to classify the data. They are structured as a diagram (tree) made of directed edges and three types of nodes : root, internal nodes and leaf. In side channel attack context, each leaf is associated to a target intermediate value to be recovered (its label) and each edge is associated to a test on the value of the leakage.

In the profiling phase, training data is used to build the decision tree. First, dataset is presented to the root and split based on a test on a leakage sample that most effectively discriminates sets associated to different target values. Each new subset created is associated to one of the child node of the root and the process is repeated on each new subset in a recursive way until the child node contains only sample associated to the same target value or the gain to split again the set become less than some threshold. Finally, it assigns a target intermediate value to each leaf.

When a new leakage sample is sent to the decision tree during the attack phase, it is first presented to the root and forwarded to one of its child node depending on the result of the edge's test. The process is repeated until a leaf is finally reached. Illustration of decision tree is presented on figure 2.5.

Random forest (RF) is constituted of many decision trees, each of them trained with a different subset of the training dataset. The output of the random forest is computed through a majority vote among its classification trees outputs.

### 2.7.3 Notes on probability and multiclass classification

To implement ML-based attack, *scikit learn* library [2] have been used in this work. In its implementation, *scikit* allows native multiclass classification (i.e classification task with more than two classes). In literature (see for instance [18]), two kinds of multiclass classification have been proposed for ML-based attack: either the model outputs directly the target intermediate value ( = 256 possible classes) or the model outputs its hamming weight (= 9 possible classes). Hamming weight classes lead to easier training but the key recovering is harder as classes are less discriminative. That is why focus will be made on classification with 256 classes. The output probabilities of the model are directly given by the function *predict_proba* implemented in *scikit* for each ML-method used in this paper. These

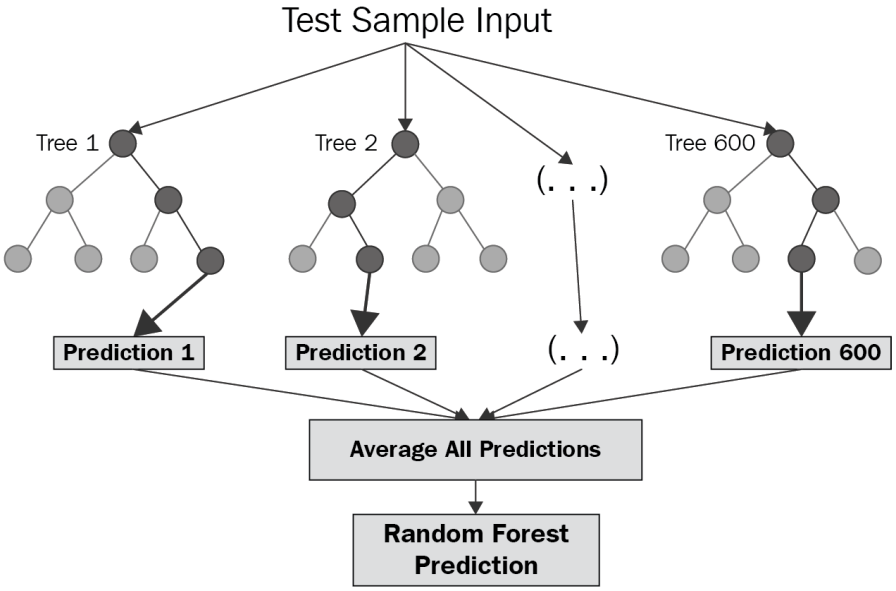


Figure 2.5: Illustration of Random forest, image from [3]

probabilities represent the probability of being part of a class according to the constructed model. For RF, *"the predicted class probabilities of an input sample are computed as the mean predicted class probabilities of the trees in the forest. The class probability of a single tree is the fraction of samples of the same class in a leaf".* see API of [2] for more details. For MLP, the output layer is constituted of 256 neurons corresponding to 256 possible target intermediate values. In this way, a model is directly built for all intermediate values at once.

## 2.8 Leakage detection methods

To find and evaluate to what extent sensitive data leaked from a crypto implementation, different methods have been proposed. These leakage detection methods can be used to evaluate the security of a device but also to discover points-of-interest (POIs) of traces recorded (i.e points leaking a lot of information in the trace). For example, in [8], Durvaux and Standaert introduced a leakage detection method based on the CPA distinguisher, the $\rho - test$.
For this test, a set $\mathcal{L}$ of $N$ traces corresponding to random plaintexts and a fixed key is recorded. To apply a K-fold cross-validation, $\mathcal{L}$ is splitted into K sets $\mathcal{L}^{(j)}$, $1 \leq j \leq k$, of equal size. The profiling set is defined as : $\mathcal{L}_p^j = \bigcup_{i=1,i\neq j}^{K} \mathcal{L}^i$ and the test set as $\mathcal{L}_t^j = \mathcal{L} \setminus \mathcal{L}_p^j = \mathcal{L}^j$. The number of traces in $\mathcal{L}_p^j$ is denoted as $n_p$ and in $\mathcal{L}_t^j$ as $n_t$ such that $n_p + n_t = N$. With theses notations introduced, let's focus on

a specific plaintext byte X (*e.g* one of 16 bytes of the plaintext). For each value of X, a model is built from the profiling set: $\hat{\text{model}}^j_{(\tau)}(X) \leftarrow \mathcal{L}^j_p$. Different choices can be made for the model. For a s bits plaintext byte, it is the mean value of the power traces for every timestep $\tau$ and for every possible value of X (from 0 to 255) that is often used. The correlation between this model and leakages samples in the test set $\mathcal{L}^j_t$ is computed as follows :

$$\hat{r}^j(\tau) = \hat{\rho}(\hat{\text{model}}^j_{(\tau)}(X), \mathcal{L}^j_X(\tau))$$

where $\hat{\rho}$ is the estimated Pearson's correlation coefficient. K cross-validation results $\hat{r}^j(\tau)$ can then be averaged in order to get a single unbiased result $\hat{r}(\tau)$ obtained from the full measurement set $\mathcal{L}$.

$$\hat{r}(\tau) = \frac{1}{K}\sum_{j=1}^{K}\hat{r}^j(\tau)$$

By applying Fisher's z-transformation and normalising value obtained with the standard deviation $\frac{1}{\sqrt{N-3}}$, a sample that can be (approximately) interpreted according to a normal distribution $\mathcal{N}(0,1)$ is obtained.

$$\hat{r}_z(\tau) = \frac{1}{2} \times \ln(\frac{1+\hat{r}(\tau)}{1-\hat{r}(\tau)}) \times \sqrt{N-3}$$

According to [8], [4] and the model used, a leakage is considered at every timestep $\tau$ where $\hat{r}_z(\tau) \leq -5$ or $\hat{r}_z(\tau) \geq 5$. An advantage of this leakage detection method is that the model is built according to a plaintext byte rather than a key-dependent intermediate value. In this way, this method can be implemented in a black box manner, without knowing the key used.

# Chapter 3

# Methods

In this chapter, all parameters, metrics and models of different experiments effected are presented. Then, the proposed countermeasure against ML side-channel attacks is introduced and discussed. The chapter ends by the choice of hyper-parameters for ML-methods.

## 3.1 General settings

**Profile of the attacker**. Attack considered being profiled attack, attacker is assumed to have full control of a training device used for the profiling phase by recording power consumption during execution of AES. During the attack phase, attacker aims at recovering an unknown secret key by measuring power consumption of a target device similar to the one used for profiling. In this way, profiling set and attack set are different.

**Evaluation Metric**. For each attack presented in this document, 20 independent sets of traces have been used. Size of these sets will depend on the experiment and will be exposed in each section. After having use these 20 independent sets for an attack, average rank of the correct key among all the key hypotheses is computed. This average correspond to the guessing entropy metric (see [21] for details). For reminder, after an attack, rank of a key $k^*$ is defined as :

$$rank(k^*) = |\{k \in \mathcal{K} | d[k] > d[k^*]\}|$$

where $d[k]$ denotes the score (here the likelihood) given to the key $k$.

## 3.2 Leakage models

### 3.2.1 Unprotected implementation of AES

In all simulations, the chosen target intermediate value is the output of the first SBox of the first round. It's a common choice in SCA due to non-linearity of the SBox. Its high level of confusion implies that similar keys will produce really different target intermediate value and so leakage[1] The following model will be considered to create leakage for a key k encrypting a plaintext p :

$$L(k, p) = HW(Sbox(k \oplus p)) + \mathcal{N}(0, \sigma^2)$$

where $\mathcal{N}(0, \sigma^2)$ is a normal zero mean distribution with variance $\sigma^2$. In this work, this leakage will be called first order leakage as the information is hidden in the first statistical moment (i.e the mean).

The noise variance allows to control the Signal-to-Noise ratio (SNR) of the leakage (i.e the quantity of information contained in the leakage compared to the noise) For remembering, SNR associate to an intermediate value z for a set of N leakages $(L^i)_{0<i\leq N}$ is defined as :

$$SNR = \frac{var_z(E_i(L_z^i))}{E_z(var_i(L_z^i))}$$

See [15] for more information about SNR in SCA context. In this document, experiences which imply only first order leakage use a noise variance of 20 or 10 corresponding to a SNR between 0.1 and 0.2.

### 3.2.2 Masked implementation of AES

Now, let's suppose a device where countermeasure against SCA have been put in place : a masking scheme with two shares. If shares are considered to be manipulated in parallel (which is the case in hardware implementation), the following model could be used to model the leakage :

$$L(k, p, s) = HW(Sbox(k \oplus p) \oplus s) + HW(s) + \mathcal{N}(0, \sigma^2)$$

where $s$, a random mask, and $s \oplus x$ are the two shares of the target intermediate value $x = Sbox(k \oplus p)$. With masking, intermediate value could not be retrieve by just observing the mean value of the leakage. In this work, this leakage will be

[1] For example, targeting the entry of the SBox is less efficient because similar key (i.e key with similar binary representation implying similar hamming weight) will produce similar intermediate value.

called second order leakage as the information is hidden in the second statistical moment (i.e the variance).

For a same SNR, the number of traces required to attack a masked implementation grows exponentially with the number of masks used in comparison to the unmasked version. In order to reduce the time and resources of simulations, the noise variance considered for the masked implementation in this work is 10.

## 3.3 Proposed countermeasure against ML

The objective of the countermeasure is to find how to change the leakage of the encryption device in a way it makes ML-based techniques construct an incorrect model leading to a wrong key guess. As the profiling set is generally different from the attack set, the advantage of this difference is exploited. The fact that a key recovering is still possible with adapted methods is also investigated.

The setup consists of two encryption process running in parallel (see figure 3.1) on the same device. The first one uses $k \oplus \delta$ as key where the delta $\delta$ is comprised between 0 and 255 and is device specific. The second one uses k as key and a masking scheme with 2 shares. The final leakage obtained at the output of the SBox is :

$$L(k, p, s, \delta) = HW(Sbox(k \oplus \delta \oplus p)) + HW(Sbox(k \oplus p) \oplus s) + HW(s) + \mathcal{N}(0, \sigma^2)$$

Within these leakages, there are information contained into the mean and the variance. The interest is on which information does ML-techniques rely to construct their model. Both information could be used or ML-methods could advantage one information over an other. As in previous models, $\mathcal{N}(0, \sigma^2)$ is a normal zero mean distribution with variance $\sigma^2$. In experiments, this noise variance is set to 10 in order to be coherent with the masked AES model used. Finally, it has to be noticed that two encryption scheme running in parallel requires synchronisation and has some cost in term of resources.

Delta $\delta$ introduced in this setup is device specific. Thus a real life situation in which an attacker has in its possession one (or more) device(s), each with a different delta taken randomly between 0 and 255 could be imagined . With this device, the profiling phase is executed by different methods seen earlier. However, no guarantee could be made that the attacked device posses the same delta as the profiling one.

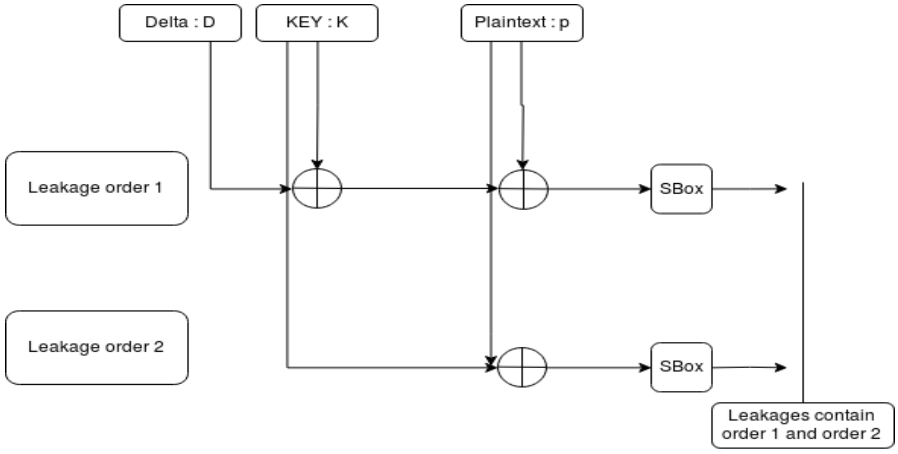


Figure 3.1: Setup studied in this document

According to the model built by ML-methods, information extracted from the mean could have a preponderant role as it's "easier" to use it to recover the key. The number of traces required to exploit a first order leakage is inferior than to exploit a second order leakage. Simulations of next chapter will confirm this fact. **However, if the delta changes between profiling and attack, an error is introduced in the first order leakage**. Question is how does ML-based methods react to this change and to which key will they converge.

Moreover, even if the delta is different between the profiling and the attack phase, information contained in the second order leakage is not affected by this change. Another interesting point is how does ML-methods take this other information into account to build their models and recover the correct key.

This idea to create and exploit a difference between the profiling set and the attack set mixes ideas of two known methods to fool ML techniques : Data poisoning (i.e here, a model which could be manipulated is injected in training data) and Adversarial inputs (i.e here, inputs of the attack use a different delta). More information on these two methods are available in [1] and [9].

In the next chapter, SCA are investigated with this setup in various cases of profiling. Objectives are to observe how information is handled by ML-methods and which key is selected by them according to the available information.

## 3.4 Choice of hyper-parameters for ML-methods

The presented ML-based methods have different parameters which needs to be tuned in order to construct effective models. In this section, analysis of the parameters

of ML methods in the context of SCA is studied and the ones leading to best performances will be kept for the rest of this work. The fact that 20 independent attack sets are used to assess performances and that they are different from the one used from the training ensures to avoid the overfitting of parameters.

All analysis for hyper-parameters are effected on unprotected implementation of AES and have been tested a posteriori on masked implementation to check these parameters were sufficient. Thus, the assumption that optimal parameters for non-masked and masked implementation are similar is done.

### 3.4.1 MLP

As explained in the background section, two main parameters[2] constitutes the MLP architecture : the number of layers and the number of units for each layer.

Let's first analyse the influence of the number of layers. For this purpose, the number of nodes per layer will be fixed to 20 (as proposed in [14], 2000 traces per intermediate value will be used for the profiling phase and $\sigma^2$ will be set to 20 to have an SNR of nearly 0.1. Good performance for two and three hidden layers could be observed in 3.2. To avoid a huge number of parameters to train and keep good performances, two layers will be used.

This first parameter having been set, the influence of the number of neurons had been studied. Not enough neurons will lead to too simple models that are not powerful enough when too much neurons will lead to much complex model difficult to train and susceptible of overfitting. From result exposed in figure 3.2, 40 neurons seems to offer best performance.

Finally, the final architecture used in this paper will be :

- Input layer: number of neurons = number of leakages considered in the attack trace.
- Hidden layer : 40 neurons
- Hidden layer : 40 neurons
- Output layer : 256 neurons corresponding to 256 possible target intermediate values.

[2]Analysis of the activation function will be omit and reLU activation function (i.e $f(x) = max(0, x)$) will be kept. Analysis of use of different activation functions in SCA context is available in [14].

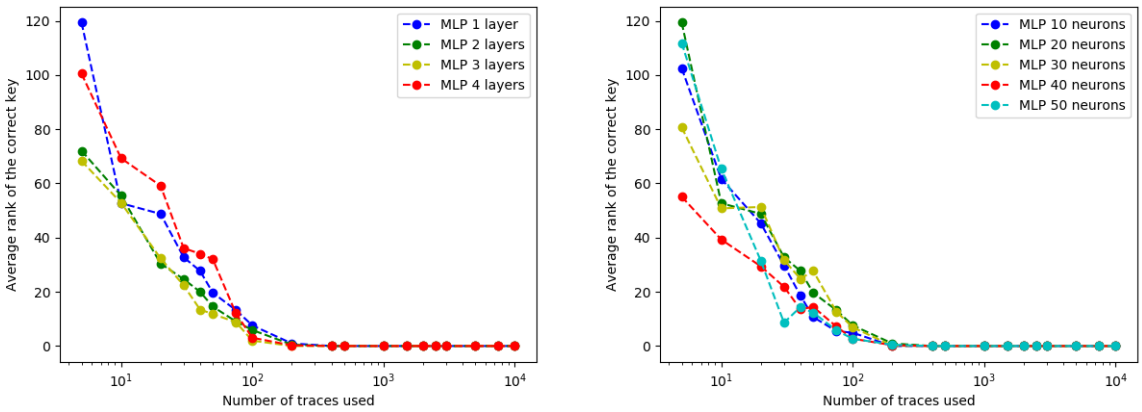


Figure 3.2: Average rank of the correct key according to the number of traces used. Noise variance of 20 and 2000 traces per intermediate value.

### 3.4.2 Random Forest

The main parameter of RF is the number of trees used for the majority vote. More trees imply best performance and lower variance of the result (A "bad" tree would have less weight in the majority vote if there are a lot of trees). However, RF training takes a lot of memory due to all the structure which need to be kept[3]. That's why this parameter could not be set to a high arbitrary value. For this test, attack will be computed with 2000 traces per intermediate value and a SNR of 0.1. Result could be find in figure 3.3.

As expected, presence of more trees implies more accuracy. Because of a memory tradeoff, 200 trees will be used as they seems to offer good performance and a lower variance than 100 trees.

[3]To avoid memory overhead, others parameters such as the maximum depth of the tree have been limited (maximum depth of the tree is set to 15)

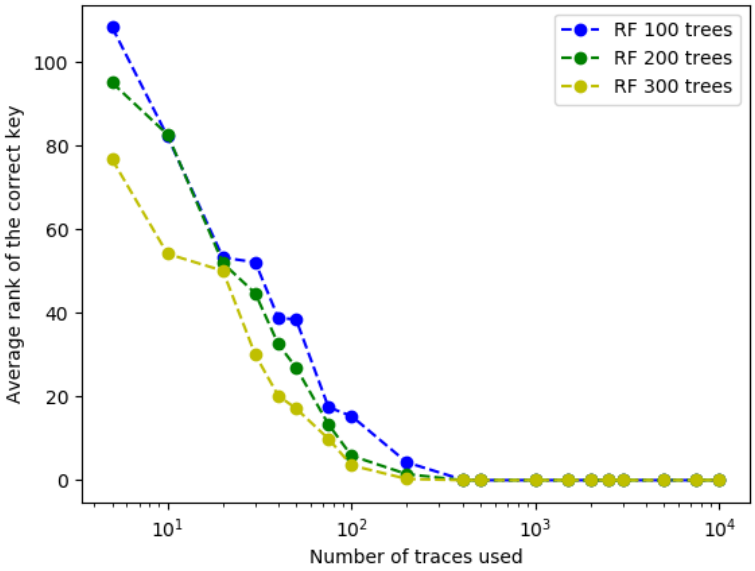


Figure 3.3: Rank of the correct key according to the number of traces used for different sizes of random forest. Noise variance of 20 and 2000 traces per intermediate value.

# Chapter 4

# Experimental results

All methods of experimentation introduced, this chapter presents experimental results obtained. First, a quick comparison between studied methods without ML-countermeasure is exposed. Then, results with the countermeasure are presented. The chapter ends with a parametric study of the countermeasure in various cases of profiling.

## 4.1 Results without ML-countermeasure

### 4.1.1 Non-masked implementation

Parameters for ML-based methods having been set, all methods could now be compared in efficiency in terms of guessing entropy. Results for MLP, RF, template and MCP-DPA targeting the mean can be seen in figure 4.1. Experiences were run with a noise variance of 20 and 40.

Template attack and MCP-DPA seems to slightly outperform ML-based attack in both cases. It could be explained by the fact that these methods use a profiling model which fits perfectly the leakage where ML-methods needs to learn this model by themselves.

For the noise variance of 20, 100 traces seem sufficient to recover the key for all methods where 400 traces are necessary for a variance of 40.

### 4.1.2 Masked implementation

For a same Signal-to-Noise ratio, the number of traces required to attack a masked implementation grows exponentially with the number of mask used in comparison

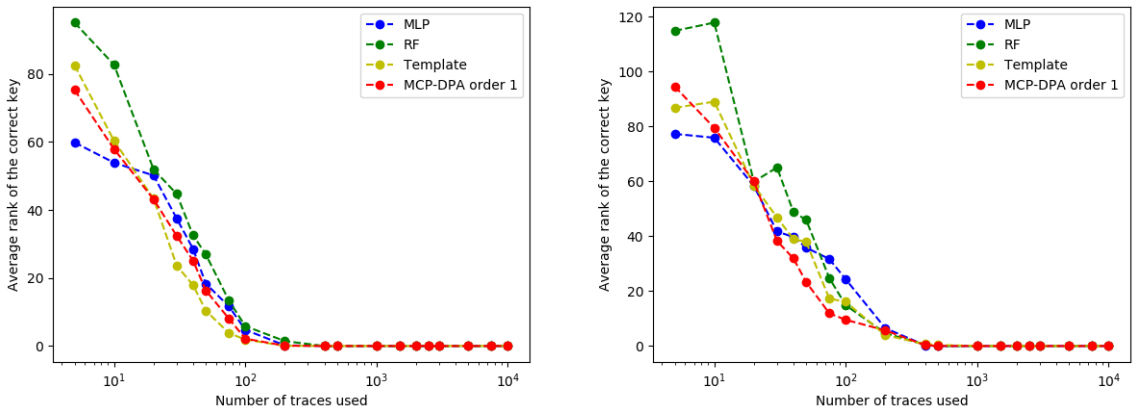


Figure 4.1: Average rank of the key according to number of traces used for all the methods, 2000 traces per intermediate value and noise variance of 20 (left figure) and 40 (right figure)

to the unmasked version. Results for MLP, RF, template and MCP-DPA second order can be observed in figure 4.2. As MCP-DPA on order 1 ignores information contained in the variance, this method is replaced by MCP-DPA on order 2.

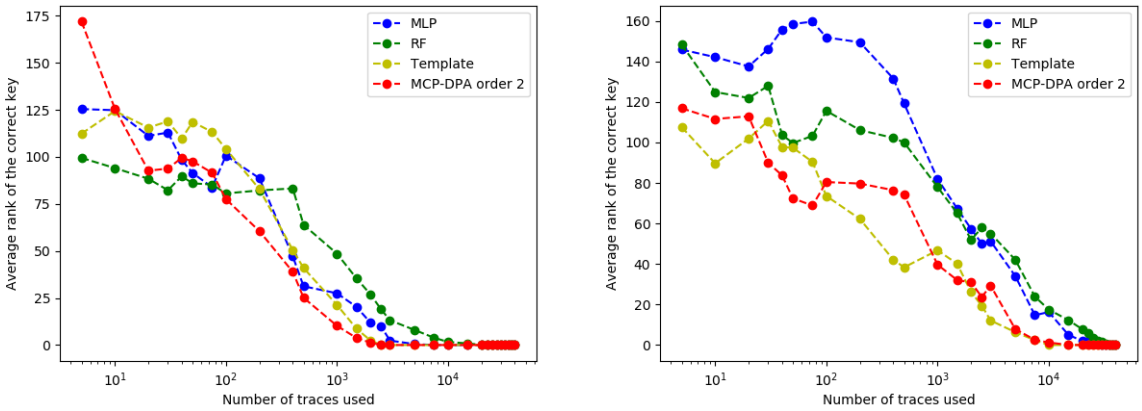


Figure 4.2: Average rank of the key according to number of traces used for all methods, 2000 traces per intermediate value, noise variance of 10 (left figure) and 20 (right figure).

Most methods success in recovering the correct key after nearly 3000 traces for a noise variance of 10 and 15000 traces for a variance of 20. As 100 traces

were sufficient for the unmasked case. This confirms what was expected about the number of traces required. MCP-DPA outperforms all the others methods as it targets directly the variance of the leakage. Template takes also the variance into account as well as the mean, that is why their performances are nearly similar. MLP offers performances close to the template's ones. Finally, RF requires more traces than the others : the key is only recovered after 10 000 and 30 000 traces for a noise variance of 10 and 20 respectively.

## 4.2 Results with ML countermeasure

### 4.2.1 Ideal case : same deltas

Before investigating cases of profiling with a delta different between the profile device and the attack device, let's consider the case of a same delta for both devices and check the key recovering process. In this case, ML-methods are expected to recover the correct key directly by constructing a model taking the delta into account. Indeed, results confirmed this fact and could be observe on figure 4.3.

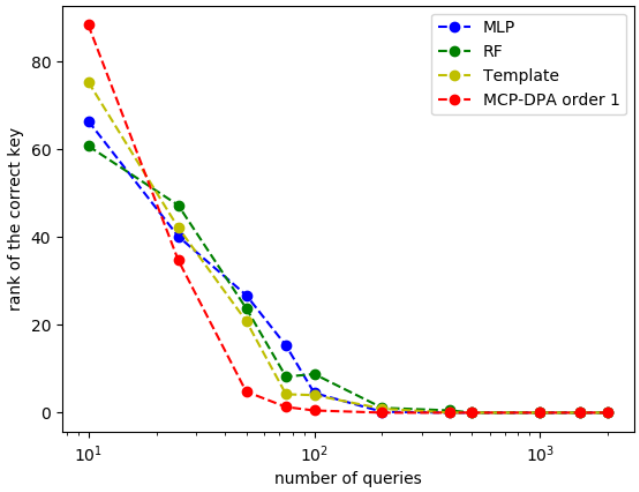


Figure 4.3: Simulation for an attacked device using the same delta as the profiling device. Noise variance of 20.

The performance of the attack seems comparable to those exposed in the previous chapter with no delta. As expected, taking the delta into the model does not seem to change difficulty of the learning and in an ideal setting the with same delta between profiling and attack, SCA could be perform as usual.

### 4.2.2 Change of delta

When a device is targeted by an attacker, he has, a priori, no control on the delta of the device. Scenario in which the delta for the profiling and the attack is different would be quite common since there exist 256 possible deltas. If the model built by ML-based methods is mainly based on the first order leakage, which is delta dependent, the key recovering is expected to fail and converges to another key. Intuitively, this key is expected to be linked to both deltas as the model built compensates effect of the $\delta_1$ (which is not present anymore) and didn't correct effect of the $\delta_2$. That's why attacks are expected to converge to the key $k \oplus \delta_1 \oplus \delta_2$ if ML-based methods didn't exploit correctly information hidden in the variance.

In figure 4.4, plots show rank of $k$,the correct key and $k \oplus \delta_1 \oplus \delta_2$, the false key. Two different deltas were used for profiling and attack.

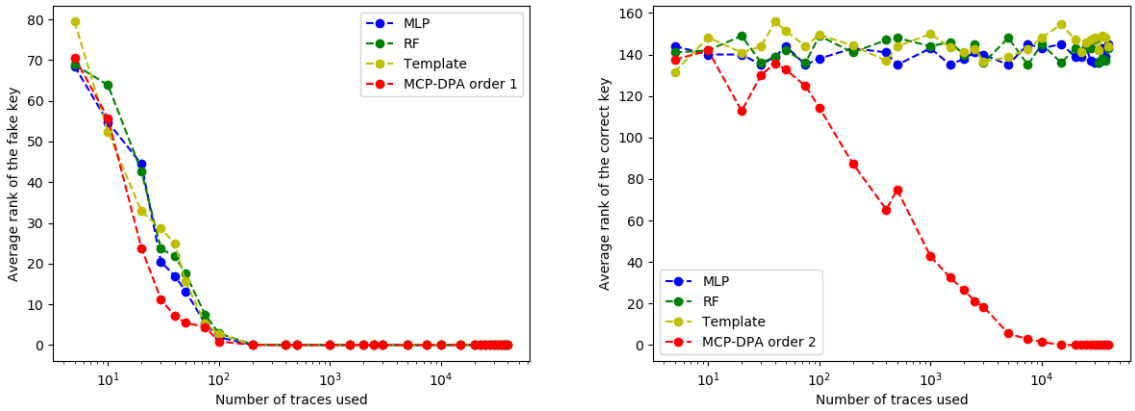


Figure 4.4: Rank of the false key (left) and the correct key (right), noise variance of 10, 2000 traces per intermediate value

According to these results, ML-based techniques converge to the fake key within 100 traces as template attack does. Models built by these methods seem to favour information of the first order leakage, the information which leads easily to a key recovering (as seen in the previous section, very few traces allow to recover the key). By exploiting this fact, ML-methods have been conducted to build a model that could be mislead thanks to the delta change.

On the figure 4.4, MCP-DPA has been used on the mean and the variance ( MCP-DPA of order 1 has been used in left plot and order 2 in right plot). This method could target any statistical moment. In this case, by targeting the variance,

the correct key is recovered within 12 000 traces when ML-based methods and template attacks give it a high average rank.

This first test demonstrates the necessity to understand which model are used during an attack. Machine learning is a great tool but could not be used without paying attention to the specificity of the problem. The fact that models are built to minimise some error criterion seems to lead them to favour some information over another. As seen in this example, this fact could be exploited.

For exhaustivity study, all these methods have been tested in a multivariate setting by creating several leakages with different noise or from SBox of different AES rounds. All methods show the same behaviour as in this univariate setting (i.e methods converge to the fake key). That's why these results will not be detailed further.

## 4.3 Parametric study

Basic cases having been presented, different parameters of the setup are studied in this section to see their impact on the key recovering process. For instance, an attacker could posses various devices to execute its profiling phase. Number of devices used and their respective deltas could affect the outcome of the attack. The presence of a device with the "right" delta among profiling devices could also affect this outcome.

### 4.3.1 Impact of the number of deltas for profiling

If several devices are used, each with its own delta, there's not anymore only one fake key but as many as the number of profiling device. If ML-methods continue to build their models only using information of the mean leakage, fake keys are expected to be keys found by the attack. However, as the number of profiling devices (and so the number of deltas) increases, the leakage part of order 1 contains less information and become progressively random. To study the impact of the number of deltas, three cases were simulated : simultaneously 10, 50 and 256 deltas taken randomly for the profiling. Figures 4.5 and 4.6 illustrate these different cases.

Even with 10 or 50 deltas, leakages of second order seem to be not properly exploited by ML-methods and the first order seems to be privileged. All keys found by the attack correspond to one of the fake key ($k^* \oplus \delta_{profil} \oplus \delta_{attack}$). However, as the number of delta increases, rank of the correct key decreases. For 10 deltas, rank of the correct key is around 70 and for 50 deltas the correct key is ranked

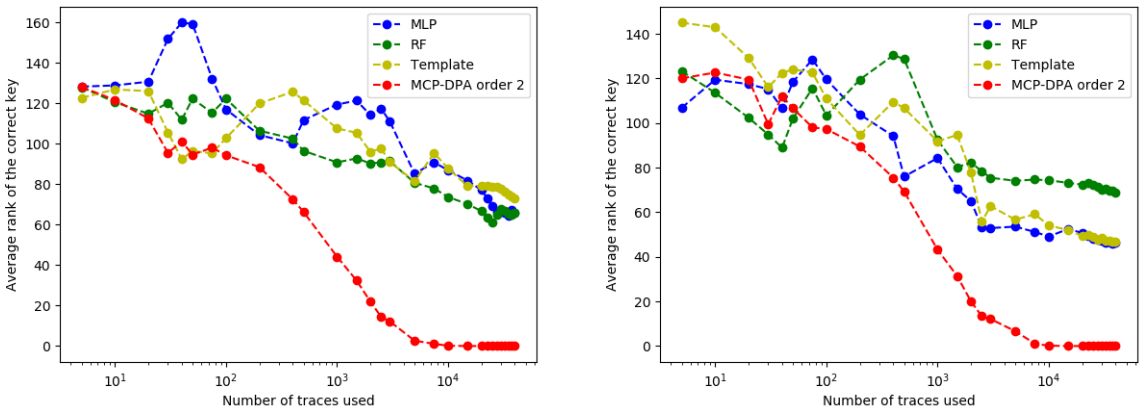


Figure 4.5: Rank of the correct key. Noise variance of 10, 2000 traces per intermediate value, 10 deltas (left figure) and 50 deltas (right figure) in the profiling set.

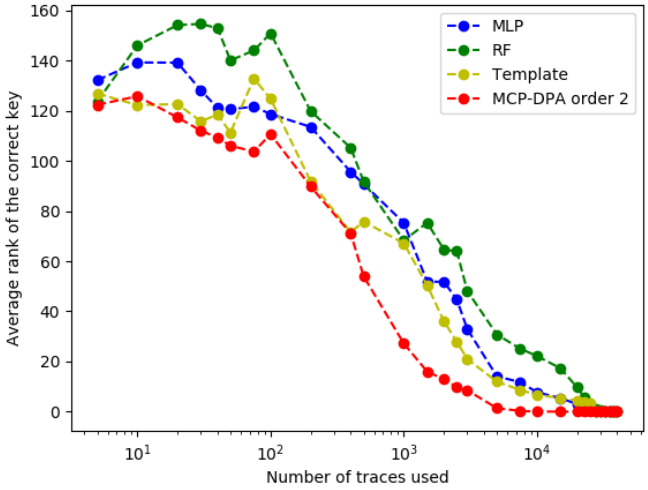


Figure 4.6: Rank of the correct key. Noise variance of 10, 2000 traces per intermediate value and 256 deltas taken randomly for the profiling set.

among the 50 first keys. Finally, for 256 deltas, the correct key is recovered within approximately 12 000 traces. These facts confirm the intuition built earlier : when the information in first order leakage decreases, ML-methods begin to use information in second order to build its model. Template attack exhibits the same tendency : when the first order leakage loses its discriminative power, the variance becomes more important for the model. Note : After further tests, it appears that

less than 100 deltas in the profiling set were enough to recover the key. Results being really similar to figure 4.6, the case of 256 deltas was kept to highlight the absence of information in the first order leakage in this case.

### 4.3.2 Influence of the presence of the correct delta in profiling set

If an attacker posses multiple devices for profiling, he could use one with the same delta as the attacked device. In this case, a part of profiling data matches leakages of the target device. The combination of these "correct" traces for first order leakage and information in the second order leakage could lead ML-methods to recover the correct key. Simulations were run with 10 deltas including the correct one, results could be observe on figure 4.7.

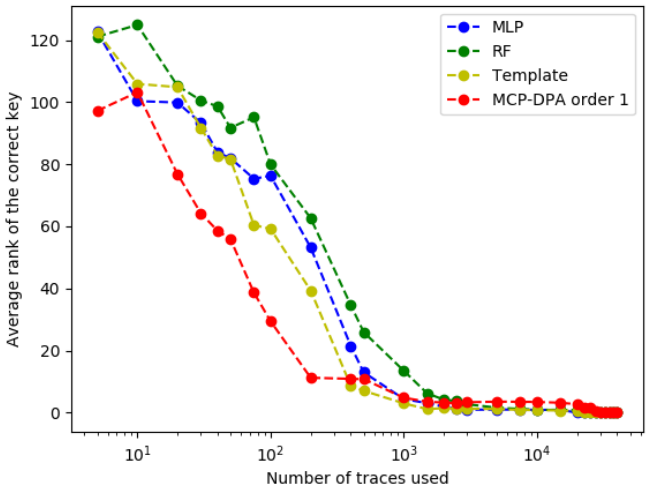


Figure 4.7: Rank of the correct key. Noise variance of 10, 2000 traces per intermediate value, 10 deltas in profiling set including the "correct" one.

The presence of the right delta in a small set of profiling device allows to recover the correct key within 1000 traces. Intuitively, ML-methods must consider the subset of data from the device with right delta as reliable and data from other devices as noisy. If the number of deltas used is increased[1], this subset becomes smaller and information contained in the first order leakage decreases as in previous experiment. A correct key guess becomes again a harder task for ML-methods and

[1]This case have been tested and results obtained are similar to those of the previous section.

he fake key could be top ranked with same chance as the correct key. The number of delta in profiling set becomes again the determinant parameter to influence which key is recovered as seen in previous section (i.e when many deltas are used, the second order leakage is again favoured and the correct key is again recovered).

### 4.3.3 Variant : the case of serial leakage

The countermeasure setup requires two encryption schemes synchronised and running in parallel to leak information together. Considering that the synchronisation between the two encryptions could not be perfect, the case of two leakages of first and second order recorded in two different time have also been investigated. The obtained leakage is the following :

$$L(k, p, s, \delta) = (H(Sbox(k \oplus \delta \oplus p)) + \mathcal{N}(0, \sigma^2), HW(Sbox(k \oplus p) \oplus s) + HW(s) + \mathcal{N}(0, \sigma^2))$$

Separating these two leakages before giving them to studied methods for a multivariate attack, could also be considered as a "pre-processing step"[2] effected to help methods recovering the correct key. Results for a delta different between the profiling and the attack could be observe on figure 4.8. MCP-DPA was not used as it would directly target one of the two leakages and so be similar to simulations seen earlier.

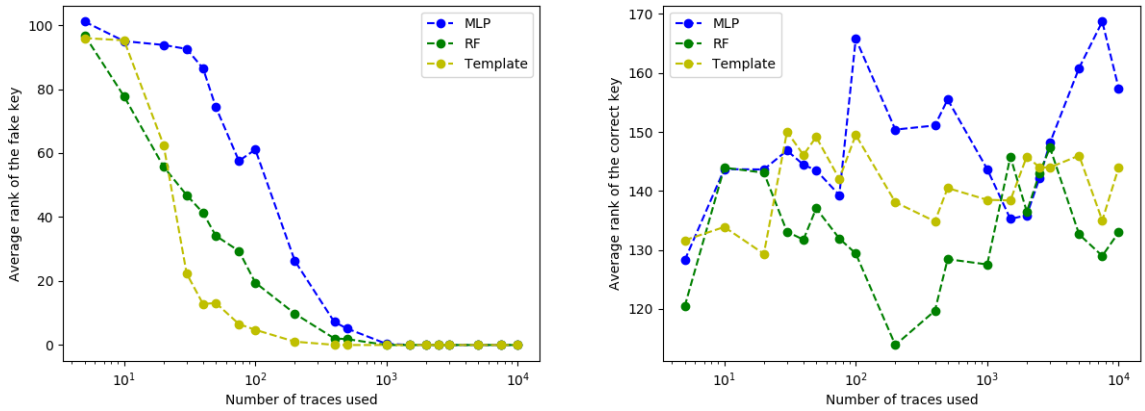


Figure 4.8: Rank of the false key (left) and correct key (right). Noise variance of 20 for first order leakage and 10 for the second order leakage, 2000 traces per intermediate value

[2]Separate these leakage could be compared to choose some features (variance, mean,...) of the leakage and give them to the ML-methods

As the rank of the fake key demonstrates, ML-methods as well as template continues to exploit mainly the first order leakage and less the second order even if they are given separately to methods. De-synchronisation of the two encryption process doesn't seem to influence which key is recovered and the fake key is recovered within 2000 traces.

**Influence of the number of points with second order leakages**

Now let's consider another case of serial leakage : one first order leakage and different second order leakages corresponding to first output byte of the SBox of each round. Each one of these leakages being recorded independently. This case could occur when the two encryption process are not synchronised and multiple points are chosen for a multivariate attack. In this case, leakages have the following form :

$$L(k,p,s,\delta) = (H(Sbox(k\oplus\delta\oplus p))+\mathcal{N}(0,\sigma^2), HW(Sbox(k\oplus p)\oplus s_1)+HW(s_1)+\mathcal{N}(0,\sigma^2)) \\ , ..., HW(Sbox(...(Sbox(k\oplus p))..)\oplus s_n)+HW(s_n)+\mathcal{N}(0,\sigma^2)) \quad (4.1)$$

where $s_i$ is a mask used at the ith round of AES for the first byte. Here there are more information in second order leakages than in the previous cases as leakages of different rounds are exploited. ML-methods could either converge to the fake key by choosing the simple model or to the correct key by choosing the model with the highest information available. Simulations have been made with different number of second order leakage, results could be seen on figure 4.9 for 5 points of a second order leakage.

For MLP, both models are taken into account. The fake key is found really fast and stays the best key but the correct key is ranked as the second best key within 10 000 traces. By increasing the number of points with only second order leakages, MLP chooses to take these information when building its model even if the model based on the first order leakage stays predominant.

RF and multivariate template converge easily to the wrong key but multivariate template doesn't success to exploit second order leakages. RF takes second order leakages into account but not as efficiently as MLP.

The conclusion of this experiment is that ML-methods favour the simple model but increasing the information relative to another model increases the probability to take this model into account.

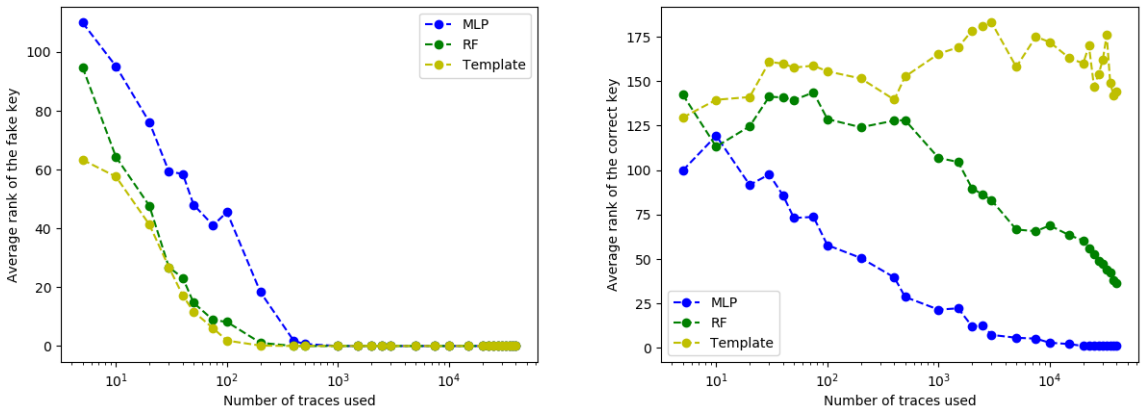


Figure 4.9: Rank of the false key (left) and correct key (right). Noise variance of 20 for first order leakage and 10 for the second order leakage, 2000 traces per intermediate value and five points for second order leakages

## 4.4 Brief discussion of obtained results

In this chapter, the countermeasure presented was demonstrated efficient to mislead ML-techniques. By taking advantage of a difference introduced between the profiling set and the attack set, the correct key could be hidden by a fake key which deliberately leaks lot information.

ML-methods seem to favour information of first order leakage, as seen in all experiments, even if information linked to second order leakages are available. The last experience confirms this trend : many second order leakages were given to methods but even in this case the fake key was recovered first followed by the correct key. On the other hand, in all experiments, MCP-DPA was shown efficient to recover the correct key as it directly targets the useful information. However, the knowledge of where to find this useful information is necessary on the opposite of Machine Learning which finds this information by itself. It is thus its strength but also it's witness.

By using many devices, it's possible for ML-techniques to recover the correct key : either a device with the same delta as the attacked device is used in the profiling set or enough devices are used to make the first order leakage useless. However, it would require a lot of resources. Adversary could also find a way to first discover the delta of the attacked device and uses this information for the key recovering.

Finally, it should be noticed that this countermeasure has some costs : it

requires running two encryption process in parallel and synchronised them (no synchronisation could lead the correct key to be ranked as the second best key). Moreover, the model used for the leakage is quite simple. However, the concept could be extend to the other models or other areas. It highlights the need to be in similar conditions between training and testing.

# Chapter 5

# Results on FPGA

Theoretical approaches and simulations of the setup having been presented, a real hardware implementation of AES on FPGA is now used to confirm previous experimental results.

## 5.1 Implementation

Implementation considered is *Domain oriented masking* (see [10]), a generic masking scheme that leads to hardware designs which can be synthesised for arbitrary protection order on FPGA.

On the FPGA, two encryption schemes were run in parallel and synchronised in term of clock cycle : one is the encryption with our key xored with a delta ( which creates first order leakage) and the other is the encryption with our key masked by mean of two shares (a random seed is used to generate the mask).

During the profiling phase, 2000 leakages per intermediate value were recorded during the full AES encryption. Plaintexts and keys were chosen randomly for each encryption where delta was fixed to 0. Then, 200 000 attack traces were recorded with a fixed key ($[0, 1, 2, 3, 4, 5, ...15]$), random plaintexts and a delta of 254.

## 5.2 Extraction of POI

As all previous simulations were performed on a single leakage and to avoid manipulating huge dimensional data, points-of-interest were extracted from traces. A $\rho$-test was effected with 400 000 traces and a fixed key. Result of this test when target the first plaintext byte could be observe on figure **??**.

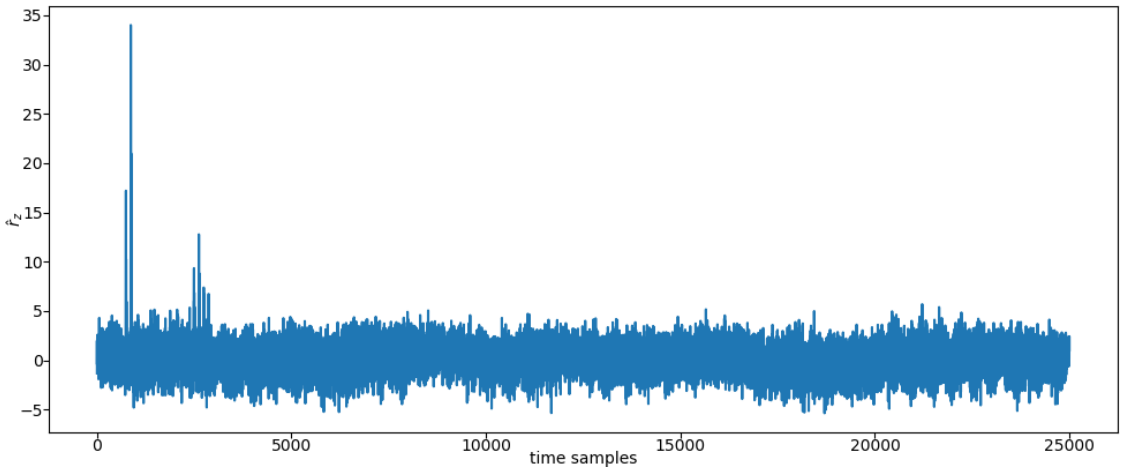


Figure 5.1: Results of $\rho$-test run with 200 000 traces, X axis : time sample, Y axis $\hat{r}(\tau)$ (normalised correlation after Fischer transform)

All samples with a value of $\hat{r}(\tau)$ higher than 5 (see background section for more information) could be chosen as POI. With the $\rho$-test, these POIs are quite localised on samples that directly depend on the chosen plaintext byte. As the key is fixed, peaks observed occur when the plaintext is directly manipulated (which leak no information about the key) as well as when the target intermediate value is manipulated. That is why the chosen POI have to be properly selected among points with high correlation. It had to correspond to manipulation of an intermediate value and thus first points with high correlation will not be used as they certainly correspond to plaintext manipulation. Different of these POIs were used for univariate attacks.

## 5.3 Univariate attack

Hereafter, rank of the true key and the fooling key recovered is plotted according to the number of attack traces. For ML-based attack, a pre-processing step of data normalisation[1] have been effected before using traces. Results of different methods could be seen on figure 5.2 for a chosen point-of-interest.

In general, around 50 000 traces are required to retrieve the wrong key for MLP, RF and template and 7500 traces for MCP-DPA on first order. An important difference could be observed from theoretical results : MLP give a better rank to the correct key in this real setup (around 40). This phenomena only occurs for

[1]Normalisation is known to help the training of ML-algorithm like MLP.

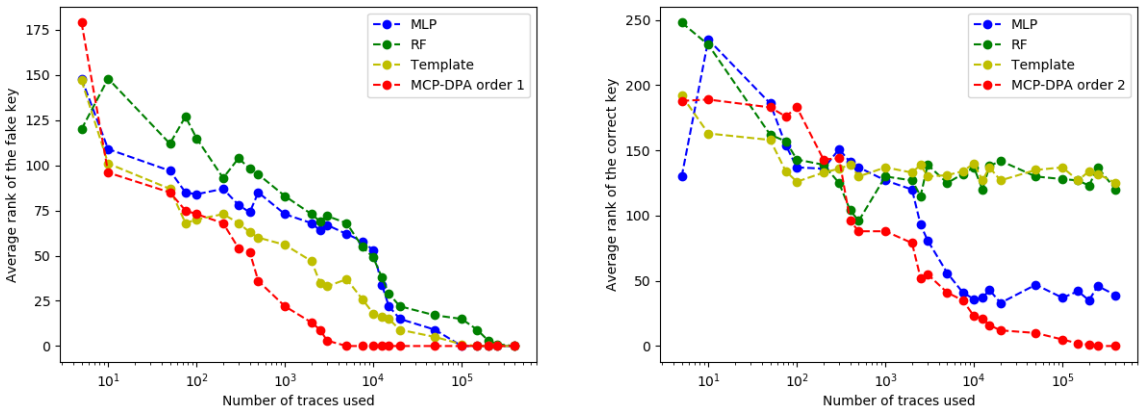


Figure 5.2: Rank of the false key (left) and correct key (right) for an univariate attack on real implementation with a chosen POI.

some POIs. The better rank obtained by MLP for the correct key is around 20 for a specific POI but most POIs give a rank to the correct key similar to the figure 5.2 or to simulated results. Moreover, in all cases the wrong key $k \oplus \delta_1 \oplus \delta_2$ is ranked first as for RF and template attack.

With real measurements, models constructed by ML-methods are more complex than the one studied earlier. It seems that depending on the POI used (and by extension the information), MLP could exploit first order as well as second order leakage. Finally, one could also observe that MCP-DPA based on second order always converge to the correct key after nearly 100 000 traces when MCP-DPA on first order converges fast to the wrong key. Other methods take more traces to find the wrong key. Perhaps because, more information being available, their models take other characteristics into account or they could be influenced by second order leakage.

## 5.4 Multivariate attack

$\rho$-test showed that many points contain information about the intermediate value manipulated. To complete this study, multiple points will be used to feed ML-techniques (multivariate template attack will also be tested). Five POIs were selected, results could be observe on figure 5.3.

All methods have similar behaviour than in the univariate case and converge to the wrong key after nearly 50 000 traces. The second order leakage does not seem to be exploited properly. Even by selecting the five "best" POIs found in the

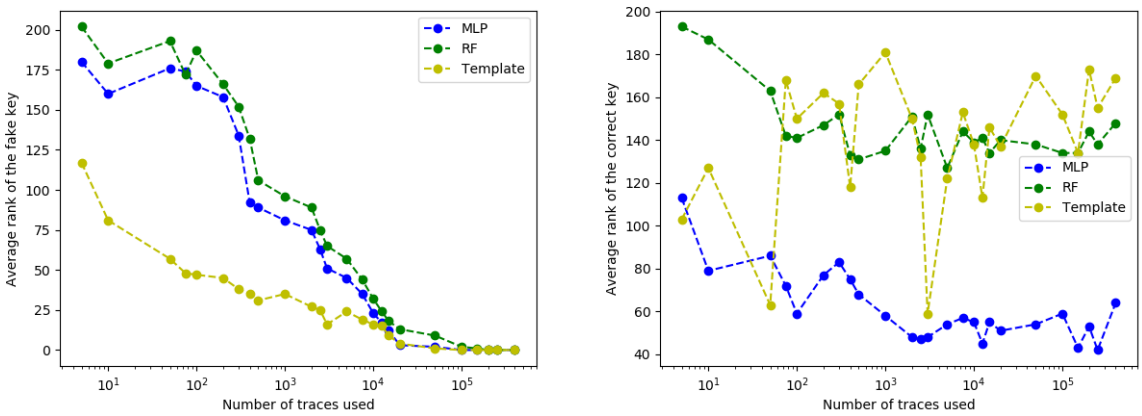


Figure 5.3: Rank of the false key (right) and the correct key (left) for a multivariate attack on real implementation on FPGA.

univariate case, MLP does not find the correct key and recover the fake one. It does not improve but even obtains less good results (rank of the correct key around 60) like if more dimension leads this methods to simplify its model.

## 5.5 Comments on results obtained

Practical results confirm the functioning of the proposed countermeasure but mitigate a little its power. When the fake key continues to be ranked first by all the methods, MLP constructs a model which seems to take some features of second order leakage into account. This phenomenon occurs only for some POIs, thus it seems to depend on the information available. However, when using multiple leakage, correct key is not retrieve and even badly ranked than before for MLP. As if information of the first order leakage was again predominant compared to the second order.

In further studies, different parameters could be varied to see their influence on the key recovering. It could be interesting to use several deltas in profiling set, to run several fake encryptions in parallel (to increase power signal relative to these encryptions) or to tune again MLP parameters as they have been tuned only with the simulate setting.

# Conclusion and perspectives

Objective of this master thesis was to demonstrate, in a side-channel context, that Machine Learning methods could be mislead. Models these methods construct from data could be manipulated to lead to a wrong prediction. By injecting a first order leakage with parameter $\delta_1$ in training samples, legitimate information of a second order leakage were nearly ignored. During the attack phase, a slight change in this first order leakage by replacing $\delta_1$ by $\delta_2$ in data led ML methods to retrieve a wrong key. Experiments on real device were also conducted and results confirm this trend. However, on real devices, models built being more complex, some POIs allow MLP to rank the correct key better than in simulations. An idea to improve the protection could be to run more fake encryptions in parallel, leading to a correct key more difficult to retrieve and implying better convergence to the wrong key.

The trick proposed to fool ML-techniques is based on the difference between profiling device using $\delta_1$ and attacked device using $\delta_2$ and highlight the need to be in similar conditions between training and testing. This problem could be avoided by ensuring enough diversity in the data used to feed Machine Learning. However, in this SCA attack context, acquiring enough diversity in traces would imply important resources (i.e using several devices to reach enough diversity in $\delta$). Exploiting this lack of diversity could be an interesting track to defeat ML techniques in other areas. It could be see as a mix between the idea of adversarial examples and the data poisoning.

In further studies, this approach to fool ML techniques could be extended to other domains, setups or parameters. Characteristics which lead to favour a model over another (i.e the "complexity") will also require more investigation to better understand the way ML-techniques handle information. Finally, more ML techniques could also be tested as adaptive methods which change their models at the time they run or even deep learning which could manage high dimensional data.

## List of abbreviations

| | |
|---|---|
| **AES** | Advanced Encryption Standard |
| **CMOS** | Complementary Metal Oxide Semiconductor |
| **FPGA** | Field Programmable Gate Array |
| **HW** | Hamming weight |
| **MCP-DPA** | Moments-Correlating Profiled Differential Power Analysis |
| **ML** | Machine Learning |
| **MLP** | Multilayer Perceptron |
| **POI** | Point of Interest |
| **RF** | Random Forest |
| **SCA** | Side-channel attack |
| **SNR** | Signal-to-Noise Ratio |

# Bibliography


[1] Overview on attack against machine learning. `https://elie.net/blog/ai/attacks-against-machine-learning-an-overview/`.

[2] Scikit-learn library. `http://scikit-learn.org/stable/`.

[3] Et al Armando Fandango, Amita Kapoor. *TensorFlow Machine Learning Projects*. Packt, 2018.

[4] Josep Balasch, Benedikt Gierlichs, Vincent Grosso, Oscar Reparaz, and François-Xavier Standaert. On the cost of lazy engineering for masked software implementations. In *International Conference on Smart Card Research and Advanced Applications*, pages 64–81. Springer, 2014.

[5] Davide Bellizia, Simone Bongiovanni, Pietro Monsurrò, Giuseppe Scotti, and Alessandro Trifiletti. Univariate power analysis attacks exploiting static dissipation of nanometer cmos vlsi circuits for cryptographic applications. *IEEE Transactions on Emerging Topics in Computing*, 5(3):329–339, 2016.

[6] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ascad database. *ANSSI, France & CEA, LETI, MINATEC Campus, France. Online verfügbar unter https://eprint. iacr. org/2018/053. pdf, zuletzt geprüft am*, 22:2018, 2018.

[7] Joan Daemen and Vincent Rijmen. Advanced encryption standard (aes)(fips 197). *Technical report, Katholijke Universiteit Leuven/ESAT (2001)*, 2001.

[8] François Durvaux and François-Xavier Standaert. From improved leakage detection to the detection of points of interests in leakage traces. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 240–262. Springer, 2016.

[9] Ian Goodfellow, Patrick McDaniel, and Nicolas Papernot. Making machine learning robust against adversarial inputs. *Communications of the ACM*, 61(7):56–66, 2018.

[10] Hannes Groß, Stefan Mangard, and Thomas Korak. Domain-oriented masking: Compact masked hardware implementations with arbitrary protection order. In *TIS@ CCS*, page 3, 2016.

[11] Jonathan Katz and Yehuda Lindell. *Introduction to modern cryptography*. CRC press, 2014.

[12] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Annual International Cryptology Conference*, pages 388–397. Springer, 1999.

[13] 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.

[14] 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.

[15] Stefan Mangard. Hardware countermeasures against dpa–a statistical analysis of their effectiveness. In *Cryptographers' Track at the RSA Conference*, pages 222–235. Springer, 2004.

[16] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. *Power analysis attacks: Revealing the secrets of smart cards*, volume 31. Springer Science & Business Media, 2008.

[17] Amir Moradi and François-Xavier Standaert. Moments-correlating dpa. In *Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, pages 5–15. ACM, 2016.

[18] 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)*, pages 4095–4102. IEEE, 2017.

[19] Lior Rokach and Oded Z Maimon. *Data mining with decision trees: theory and applications*, volume 69. World scientific, 2008.

[20] François-Xavier Standaert. Introduction to side-channel attacks. In *Secure Integrated Circuits and Systems*, pages 27–42. Springer, 2010.

[21] François-Xavier Standaert, Tal G Malkin, and Moti Yung. A unified framework for the analysis of side-channel key recovery attacks. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 443–461. Springer, 2009.

**UNIVERSITÉ CATHOLIQUE DE LOUVAIN**
**École polytechnique de Louvain**
Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve, Belgique | **www.uclouvain.be/epl**