UCLouvain

École polytechnique de Louvain

Side-Channel Attacks Against a Bitcoin Wallet

Authors: **Tristan Bodart, Alexandre Ghos**
Supervisor: **François-Xavier Standaert**
Readers: **Baudoin Collard, Jean-Charles Delvenne, François Koeune, Balazs Udvarhelyi**
Academic year 2020–2021
Master [120] in Electro-mechanical Engineering
Master [120] in Mathematical Engineering

Abstract

Nowadays, the cryptocoins and especially the Bitcoin have become very popular among the general public. This trend led to the commercialization of hardware wallets to store them. This master thesis aims to attack the open source hardware wallet developed and commercialized by Satochip thanks to side-channel attacks. The different potential attack points of the wallet will first be described. Then, a key derivation algorithm based on HMAC SHA-512 and used by the wallet will be chosen to be the subject of the attack developed in this work. A measurement setup relying on current consumption will be mounted and then, some metrics will be computed to evaluate the level of information available in the traces. Some limitations due to the card and to the communication protocol have prevented to perform a great alignment of the traces, and the result is a weak SNR and PI value. After that, a simulated SASCA will be performed against the HMAC SHA-512 for different noise levels. Finally, the results will show that the attack would be feasible with a sufficient level of information in the traces. Unfortunately, the success rate is low for the noise level observed in the real traces, so the attack will be unlikely to be successful in a real case with this card. However, since the metrics values are usually higher on this type of smart card, the security of another device against this attack could clearly be questioned.

Acknowledgements

First, we would like to thank Pr. François-Xavier Standaert, our supervisor at UCLouvain, for his support and the sharing of his experience throughout the year. It is thanks to his feedback and advice that we were able to complete this work. He also gave us the opportunity to discover the world of cryptographic research and we are grateful to him.

We thank PhD. Baudoin Collard, our external supervisor, for his availability and for the information and the advice provided during this thesis. He provided us the cards and helped us to understand and use the Satochip Hardware Wallet so that we could work in the best conditions.

Then, we would like to thank Balazs Udvarhelyi, who has been always available and involved in our work by providing advice and suggesting ideas when we encountered some problems. He also helped us to make the experimental set up and some manipulations in the laboratory. Without his support and help, it would have been very difficult to carry out this master thesis.

We also thank Pr. François Koeune for having accepted to be part of the jury and for having followed us throughout the year, while giving us feedback at key stages of the work.

Next, we thank Olivier Bronchain for his advice and the access to his toolbox. He also took the time to make some improvements and modifications on it which helped us a lot.

We thank Pr. Jean-Charles Delvenne, who also accepted to take part in this master thesis as a jury member.

After that, we want to thank Godefroy Devys for his proofreading that allowed us to improve the writing of this work.

Finally, we thank our families, friends and roommates who provide us their support and motivation during this year and who have helped us to complete these 5 great years of study at EPL.

Contents

Chapter 1

Introduction

The number of people using smart cards is increasing nowadays. Indeed, these are everywhere: ID cards, banks cards, SIM cards, ... Another emerging sector is the sector of cryptocurrencies. Currently, the total market capitalization of the cryptocurrencies reaches more than 2 trillion dollars. This rise led to the development of commercial activities related to this field. And the possibility to pay with cryptocoins started to develop with the possibility to store them on smart cards such as real money, which makes them even more accessible. Companies started to develop and sell hardware wallets on smart cards with their own implementations of cryptographic algorithms which can be quite poorly protected in order to be competitive with market prices. The robustness of the Bitcoin protocol could therefore be useless if some flaws are present in the wallets of the users.

Side-channel attacks have been the subject of many research during the past few years, it uses the physical leakage of cryptographic devices in order to recover its secrets. This kind of attack is then perfectly applicable to those hardware wallets, so their security can be seriously questioned.
In this work, a hardware wallet commercialised by Satochip, will be attacked with side-channels and more particularly, the attack will focus on a software implementation of a HMAC SHA-512.

Attacking cryptocurrencies hardware wallet and break their security has already been studied in the literature. As an example, San Perdo et al. in [1] attacked a hardware wallet using two paths. The first one recovered the user's PIN code by exploiting the verification function, while the second one recovered a private signing key from the scalar multiplication of ECDSA using a single signature.

It is not the first time either that a HMAC function will be targeted by a side-channel attack. The first theoretical DPA attack against HMAC, according to our knowledge, was made in 2004 by Lemke et al. [2]. This attack was aimed at the hash functions RIPEMD-160 and SHA-1. Then Okeya et al. published three papers

in 2006, 2007 and 2008[3] [4] [5] which were based on the evaluation of the security of HMAC, and more precisely SHA-1 hash functions, against DPA. McEvoy et al. also evaluated the security of HMAC with the Hamming distance leakage model but with SHA-2 hash functions this time[6]. They had to make strong assumptions on the target implementation to make it work. They showed that DPA attacks were practicable in reality and they also developed some countermeasures. In 2009, Fouque et al. developed a template attack against HMAC SHA-1 in [7], they succeeded to recover the key of the HMAC with only one iteration. Zhang et al. showed in 2011[8] that HMAC with whirlpool hash algorithm was vulnerable to CPA and DPA. Zohner et al. evaluated the security of HMAC with SHA-3 hash and more particularly the BLAKE, Grøstl, JH, Keccak, and Skein algorithm[9]. In 2012, Bertoni et al. also studied DPA against HMAC with Keccak hash function in [10]. Belaid et al. performed DPA attacks against HMAC SHA-2 as McEvoy et al. did but with weaker hypothesis[11]. More recently, Collin attacked HMAC SHA-256 with CPA and Template Attack[12].

Even if attacking HMAC seems to be something quite common and feasible given the number of works available on this subject, our approach of the attack has something new. Indeed, all the attacks already performed against HMAC in the literature aimed to recover the key of the HMAC, because it is usually the important data when the HMAC is used in its chosen field. In the context of the Bitcoin environment, the HMAC is used for a key derivation and this requires a different approach. The key of the HMAC being already known since it is made of public values, it is some bytes of the HMAC's message that have to be recovered because they contain the secret data. This way of attacking HMAC has never been tried, which is therefore the contribution of our work to all the others already done on the subject.

This work is constituted as follows: The Chapter 2 contains the theoretical background which regroups the cryptographic algorithms and tools used in this work and the leakage sources and side-channel attacks that are needed to have a good understanding of this master thesis. Then, in Chapter 3, an overview of the Bitcoin protocol will be made, and followed by a presentation of the Satochip hardware wallet in Chapter 4. This will allow to understand some difficulties encountered while performing side-channel attacks on this smart card. After that, in Chapter 5, the different attack vectors in the wallet will be described and analysed which will lead to the choice and the design of the attack. The measurement setup will be detailed in Chapter 6 and the experimental results are exposed in Chapter 7. Finally in Chapter 8, a conclusion of the work done and the results obtained will close this master thesis.

Chapter 2

Theoretical background

In this section, the theoretical knowledge needed for a good understanding of this master thesis will be described. First with a description of the cryptographic primitives and algorithms, then with the tools used to find the Points Of Interest (POI) in the side-channel leakages, the channels that had been used during the experiments and finally with a description of the side channel attacks that will be discussed.

2.1 Cryptographic primitives/algorithms

A cryptographic primitive is a low-level cryptographic function, from which different algorithms can be built. In this section, the different primitives used in this work are detailed.

2.1.1 SHA-512

Hash functions are compression functions currently used in many domains such as the Bitcoin protocol, data compression, message authentication codes, ...
There are different hash functions families currently used in information security applications: MD, SHA, RIPEMD,...
The first SHA function was created in 1993 and a lot of other functions have been developed over the years to improve resistance to attacks and correct the weaknesses.
SHA-512 is a Secure Hash Algorithm belonging to the SHA-2 family. It has been created in 2001 by the National Security Agency of the USA. A good hash function, such as the SHA-512 function is appreciated thanks to 3 properties. They are described in [13]:

- Preimage resistance: for essentially all pre-specified outputs, it is computationally infeasible to find any input which hashes to that output, i.e., to find any preimage x' such that H(x') = y when given any y for which a corresponding input is not known.
- Second preimage resistance: it is computationally infeasible to find any second input which has the same output as any specified input, i.e., given x, to find a 2nd-preimage x' $\neq$ x such that X(x) = H(x').
- Collision resistance: it is computationally infeasible to find any two distinct inputs x, x' which hash to the same output, i.e., such that H(x) = H(x').

The SHA-512 function takes as input a message of arbitrary length and outputs a digest message of 512 bits length. The algorithm operates in 64 bits words and includes 80 rounds, detailed in Algorithm 1. [14]

In Algorithm 1, the input vector (I) contains the message cut in 16 words of 64 bits and the initial vector (V) contains initial values independent of the message. This algorithm uses the functions $SIG_0, SIG_1, CH, \Sigma_0, \Sigma_1, MAJ$ that are defined as follows:

- $\text{SIG}_1(x) = \text{ROTR}^{19}(x) + \text{ROTR}^{61}(x) + \text{SHR}^{6}(x)$
- $\text{SIG}_0(x) = \text{ROTR}^{1}(x) + \text{ROTR}^{8}(x) + \text{SHR}^{7}(x)$
- $\text{CH}(x, y, z) = (x\&y) \wedge (\sim x\&z)$
- $\Sigma_1(x) = \text{ROTR}^{14}(x) + \text{ROTR}^{18}(x) + \text{ROTR}^{41}(x)$
- $\Sigma_0(x) = \text{ROTR}^{28}(x) + \text{ROTR}^{34}(x) + \text{ROTR}^{39}(x)$
- $\text{MAJ}(x, y, z) = (x\&y) \wedge (x\&z) \wedge (y\&z)$

In which,

- "$ROTR^n(x)$" =circular right shift (rotation) of the 64-bit argument x by n bits,
- "$SHR^n(x)$" =left shift of the 64-bit argument x by n bits with padding by zeros on the right,
- "+"= addition modulo 2^{64}.

Algorithm 1 SHA-512 compression function

Input: the 16 words of the block $I = (I_1, ...I_{16})$, the 8 words of the initial vector $V = (V_1, .., V_8)$

Output: the digest $D = (D_1, ...D_8)$

1: $(M_1, ...M_{16}) \leftarrow (I_1, ...I_{16})$
2: **for** i=17 to 80 **do**
3: $\quad M_i \leftarrow SIG_1(M_{i-2}) + M_{i-7} + SIG_0(M_{i-15}) + M_{i-16}$
4: **end for**
5: $(A, B, C, D, E, F, G, H) \leftarrow (V_1, ...V_8)$
6: **for** i=1 to 80 **do**
7: $\quad T_1 = H + CH(E, F, G) + \Sigma_1 E + M_i + K_i$
8: $\quad T_2 = \Sigma_0 A + MAJ(A, B, C)$
9: $\quad H = G$
10: $\quad G = F$
11: $\quad F = E$
12: $\quad E = D + T_1$
13: $\quad D = C$
14: $\quad C = B$
15: $\quad B = A$
16: $\quad A = T_1 + T_2$
17: **end for**
18: **return** $D = (D_1 \leftarrow A + V_1, ..., D_8 \leftarrow H + V_8)$

2.1.2 HMAC

To understand the HMAC function, it is necessary to know the definition of a Message Authentication Code (MAC). In [15], the MAC is defined as a tag message, that is made of a message and a key. This key is secret and has previously been shared by the sender and the receiver. When the sender wants to send the message m to the receiver, he computes a tag t thanks to a tag-generation algorithm and sends the message m and the tag t to the receiver. Then, the receiver is able to verify that the tag is valid and corresponds to the message that has been sent. The goal is to prevent an adversary from modifying a message sent by the sender without the parties detecting that a modification has been made.
Now that the MAC has been defined, the HMAC can be explained. It is a keyed-hash message authentication code, which is a type of MAC often used to verify the integrity and the authenticity of a message, but it is also used for the key derivation in Bitcoin applications.
The Algorithm 2 is inspired of the description given in [15].

Algorithm 2 HMAC

Inputs: the key K, the message to be authenticated M and the cryptographic hash function H.

Output: the $HMAC^K(M)$

1: **return** $HMAC^K(M) = H((K \wedge opad)||H((K \wedge ipad)||M))$

In this algorithm, opad is the byte "0x36" repeated a certain number of times to have the same size as the key K, and ipad follows the same method with the byte "0x5C". So it is important to note that these are constant values.
The sign || represents a concatenation of two values. The security of the HMAC depends on the security of the hash function used, the size of the output and the size of the secret key.
In Figure 2.1, which comes from [11], the structure of the HMAC can be seen, with F the hash function and k the key. This structure is made of two separate hashes, the inner hash and the outer hash. It can be seen that the number of compression functions F of the inner hash depends on the length of the message m.

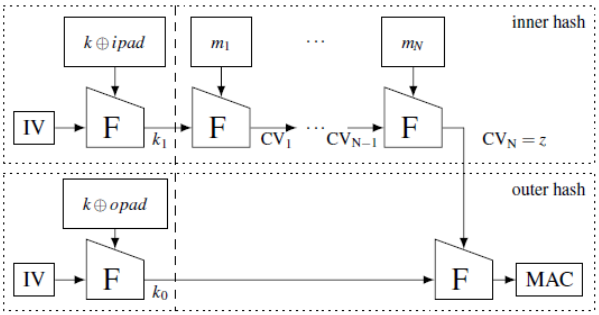

Figure 2.1: HMAC scheme with hash function F and key k

Thanks to its properties, the SHA-512 hash function guarantees the security for the HMAC and the two of them form the HMAC SHA-512 structure that will be used later in this thesis.

2.1.3 Public key cryptography

The Bitcoin ecosystem is based on public key cryptography, also called asymmetric cryptography. It is an encryption method that uses two different keys, the public one which will be used to encrypt messages and the private one to decrypt them. These two keys therefore form a pair, their generations can be made using various cryptographic algorithms. It is different from the symmetric cryptography where there is only one same key used to encrypt and decrypt a message.
The advantage of this asymmetric cryptography is that the public key is computed from the private one using one-way functions and then it's very difficult and unlikely to recover this private key with the public one. So the public keys can be shared with anybody without the risk that the private key can be recovered.
In these systems, if Alice wants to send a message to Bob, Alice can use Bob's public key to encrypt the message and he will be the only one able to decrypt it using his private key. Such a process is illustrated in Figure 2.2:

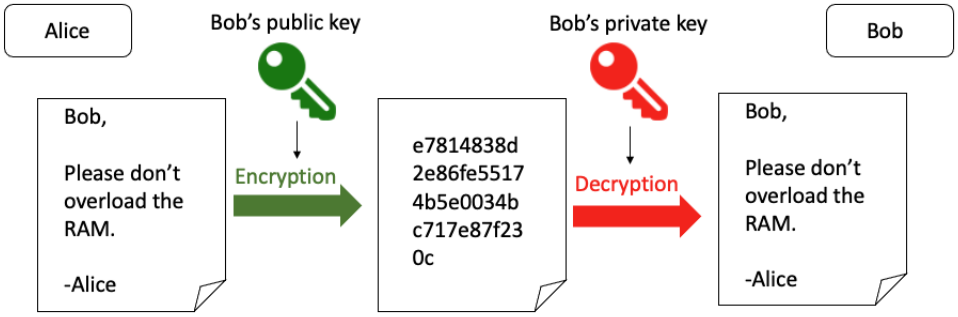

Figure 2.2: Illustration of public key cryptography

For Bitcoin application, the private keys are used for authentication purpose by signing transaction and the public keys are used to verify the transaction authenticity and to receive a transaction. All the transaction protocol will be explained more in details later in this paper. Two algorithms using public key cryptography that are used in the context of this master thesis are presented in the next sections.

2.1.4 Elliptic curves (EC)

Elliptic curves are an important primitive for the public key cryptography and especially regarding Bitcoin environment. This section presents an overview of elliptic curves and operations that can be effectuated with them. Those curves are presented in an affine coordinate system since it is easier to follow but it is important to note that some applications present EC computations in projective coordinates. This section is inspired from [16].

The conversion between these two coordinates systems can be made easily. As stated in [17], an elliptic curve E over a field $\mathcal{K}$ is given by the Weierstrass equation :

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \tag{2.1}$$

with coefficients $a_i \in \mathcal{K}$, $i = 1..6$ such that coordinates in $\mathcal{K}$ for which 2.1 is satisfied don't vanish simultaneously (smoothness of the curve, no singular point) the partial derivatives :

$$2y_1 + a_1x_1 + a_3 \text{ and } 3x_1^2 + 2a_2x_1 + a_4 - a_1y_1, \tag{2.2}$$

In the cryptography context, elliptic curves are often defined over the prime field containing p elements $\mathcal{F}_p$. Assuming that p is large ($p > 3$), one can take for $\mathcal{E}$ an equation of the form cf. [17]:

$$\mathcal{E} : y^2 = x^3 + a_4x + a_6 \ (mod\ p) \tag{2.3}$$

together with the point $\mathcal{O}$ called point at infinity. With this definition, one can define some operations on that curve :

1. the opposite of a point (x, y) :

$$-(x, y) = (x, -y) \forall (x, y) \in E(\mathcal{F}_p)$$

2. Adding point of infinity to another point :

$$(x, y) + \mathcal{O} = \mathcal{O} + (x, y) = (x, y) \forall (x, y) \in E(\mathcal{F}_p)$$

3. Adding two points with same x-coordinates when the points are distinct or have y-coordinate=0:

$$(x, y) + (x, y + x) = \mathcal{O} \forall (x, y) \in E(\mathcal{F}_p)$$

4. Adding two points with different x-coordinates : Let $(x_1, y_1), (x_2, y_2) \in E(\mathcal{F}_p)$ with $x_1 \neq x_2$. Then :

$$(x_1, y_1) + (x_2, y_2) = (x_3, y_3)$$

with :

$$x_3 \equiv \lambda^2 - x_1 - x_2 (\mathrm{mod} p),$$
$$y_3 \equiv \lambda (x_1 - x_3) - y_1 (\mathrm{mod} p),$$
$$\lambda \equiv \frac{y_2 - y_1}{x_2 - x_1} (\mathrm{mod} p).$$

5. Adding a point to itself : Let $(x_1, y_1) \in E(\mathcal{F}_p)$ with $y_1 \neq 0$:

$$(x_1, y_1) + (x_1, y_1) = (x_3, y_3)$$

with :

$$x_3 \equiv \lambda^2 - 2x_1 (\mathrm{mod} p),$$
$$y_3 \equiv \lambda (x_1 - x_3) - y_1 (\mathrm{mod} p),$$
$$\lambda \equiv \frac{3x_1^2 + a}{2y_1} (\mathrm{mod} p).$$

Once the addition rule is totally defined, one can notice that the set of points on $E(\mathcal{F}_p)$ forms an abelian group under this addition rule.
Another important operation while working with elliptic curve cryptography, and then with Bitcoin, is the scalar multiplication of a point of the curve by a scalar integer i. A simple and obvious way to compute this multiplication is to add the

point $P \in E(\mathcal{F}_p)$ to itself i times following the rule described above. The result is therefore iP. A common and efficient way to perform this multiplication is the double-and-add algorithm as presented in [18].

More specifically for the Bitcoin application, the parameters of the elliptic curve used are well detailed in [19].

Now that the operations with elliptic curves have been detailed, two algorithms using these operations will be presented in the next section. Those algorithms are called the EDCSA and the ECDH algorithm and they are used in the protocol set up by Satochip, so they are essential for a good understanding of this protocol.

2.1.5 ECDSA

The Elliptic Curve Digital Signature Algorithm (ECDSA) is developed in Algorithm 3 and is explained in detail in [20]. It is the elliptic curve equivalent of the Digital Signature Algorithm, but the interest of using the elliptic curves is that the strength-per-key-bit is better than without elliptic curves.
The goal of this algorithm is to provide a signature of a message. More precisely, it uses a hash function so the signature is made of the hash of the message and not directly of the message itself. This signature is dependent on the secret key of the signer but also on the message that is signed. It has to be verifiable (following Algorithm 4) by a third party without accessing to the private key, so the signer can't repudiate his own signature and the message can't be changed after it has been signed.
This algorithm is part of asymmetric cryptography, as defined above, because the verification of the signature is made thanks to a public key, computed from the private one used to sign, which is also provided by the signer.

Signature generation

Algorithm 3 Elliptic curve digital signature generation

Input: The parameters of the curve (q, a, b, G, n)
Output: the pair (r, s)

1: Select a random or pseudorandom integer $k, k \in [1, n-1]$
2: Compute point $P = (x, y) = k * G$ and $r = x \bmod n$
3: **if** r=0 **then**
4: Go to line 1
5: **else**
6: Compute $e = H(m)$ $\rightarrow$ H=(SHA-1 (160 bits))
7: Compute $s = k^{-1}(e + dr) \bmod n$
8: **if** s=0 **then**
9: Go to line 1
10: **end if**
11: **end if**
12: **return** (r, s)

Signature verification

Algorithm 4 Elliptic curve digital signature verification

Input: ECDSA signature (m, r, s) Public Key Q
Output: Signature Accept or Reject

1: Compute $w = s^{-1}(mod\, n)$
2: Compute $u = h(m) * w(mod\, n)$
3: Compute $v = r * w(mod\, n)$
4: Calculate value of R.y to retrieve point R
through square root method from received r value
5: Compute $R = uP + vQ \in E(\mathcal{F}_p)$
6: **return** $x(R) == r\ (mod\, n)$ (with $x(R)$ the x-coordinate of R)

2.2 Detection tools

The goal of the Point Of Interests detection is to locate where the sensible information is located in a trace. The tools further developed can also give information on the quantity of information available in the side-channel leakage. The ones that will be used in this thesis are the Welch's t-test, the Signal-to-Noise Ratio (SNR) and the Perceived Information (PI).

2.2.1 Welch's t-test

As explained in [21], the Welch's t-test is an adaptation of the Student's t-test [22]. It is used to highlight the differences between two populations thanks to a statistic test involving the means and variances of the two populations. This test is the following,

$$t = \frac{\hat{\mu}_1 - \hat{\mu}_2}{\sqrt{\frac{\hat{\sigma}_1^2}{N_1} + \frac{\hat{\sigma}_2^2}{N_2}}}, \tag{2.4}$$

where σ_i is the standard deviation of the population i and N_i is the number of samples. If $|t|$ is larger than 4.5, one can conclude that there is a significant probability that there is a difference between the two populations (with a p-value smaller than 10^{-5}). This test is really helpful when the side-channel leakages are quite long and noisy. During this work, the encountered noise level is high so the number of traces needed to compute the SNR is quite huge (more than 10^6), this implies that the SNR cannot be computed directly because of the too high computational cost. That explains the interest to compute a t-test first.
The inconvenience of the Welch's t-test is that it can only distinguish two populations while the SNR can deal with the 2^{16} populations of a 16-bits variable. But as the computational cost of the t-test is lower than the SNR's, it can provide the window in which the SNR will be applied.

Methodology

The methodology used in a side-channel measurement context is called Test Vector Leakage Assessment (TVLA), and is well explained in [23]. Suppose that n measurements of the leakage are recorded for a device that has a constant secret key, with n different plaintexts (denoted by $P_{i\in\{1,...,n\}}$). Each measurement (denoted by $T_{i\in\{1,...,n\}}$) contains m samples (denoted by $\{t_i^{(1)}, ..., t_i^{(m)}\}$). Then, the test will be performed for each of the m samples. Since it is an evaluation, the "secret" key is known, so all the intermediate values can be computed for each leakage measurement. Then those measures will be separated in 2 sets Q_0 and Q_1, with $Q_0 = \{T_i|\text{target bit}(P_i)=0\}$ and $Q_1 = \{T_i|\text{target bit}(P_i)=1\}$. After that, one can compute the t-test and see if the two sets are distinguishable from each other at some samples.

2.2.2 Signal-to-Noise Ratio

The SNR [21] is used to evaluate the available signal about an intermediate variable into the leakage measurements. For a cryptographic application, the

SNR for one time sample is computed for a targeted intermediate value X and is estimated as,

$$SNR = \frac{Var[\mu_x]}{E[{\sigma_x}^2]} \tag{2.5}$$

where μ_x and ${\sigma_x}^2$ are respectively the mean and the variance of a time sample for each possible value of the intermediate value X. The SNR is calculated for each time sample and will point the samples of the leakage traces that are linked to X, which are called the POIs.

2.2.3 Perceived Information

The Perceived Information (PI), defined in [24], is the lower bound of the Mutual Information (MI) which is contained in the leakage of a device. The concept of MI is well explained in [25]. This kind of measurement is inspired from the logarithmic measures of information detailed in the mathematical theory of communication introduced by Shannon in 1948 [26]. This PI is computed thanks to estimated models for the leakage, so it will then depend on the quality of those estimated models. It is also a metric that will give information on the Gaussian noise in the leakage measurements, but compared to the SNR, the main advantage of this tool is that it can be based on multivariate models.
The computation is made in two phases: the first one is a profiling phase. As the true distributions $p(k|\mathbf{l})$ (where $\mathbf{l}$ are the leakage values and k the intermediate value) are unknown, Gaussian templates will be built as estimated models of leakage (this process will be detailed in the Template Attack subsection) and they will correspond to the maximum likelihood estimates of the conditional density function f$[L|x]$.
During the second phase, the attacker will use those templates to compute the concrete success probability of a maximum likelihood attack. If the model is too different of the true distribution, the PI could be negative and will underestimate the information leaked by the device.
In the discrete case, the PI is defined as follows:

$$\widehat{PI}(K;L) = H(K) + \sum_{k \in \mathcal{K}} p(k) \cdot \sum_{i=1}^{n_t(k)} \frac{1}{n_t(k)} \cdot log2(\tilde{m}(k|l_k(i))) \tag{2.6}$$

In which, $n_t(k)$ is the number of traces used for the value k, $\tilde{m}(k|l_k(i))$ is the estimated model of the conditional distribution and $H(K) = log2(|\mathcal{K}|)$.

2.3 Leakage sources

Before getting to the heart of the matter of side channel attacks, it's important to define a side channel. To illustrate that, let's explore different viewpoints regarding the implementation of the cryptographic primitives previously presented: they can be seen as simple black boxes performing mathematical operations, a computer program running on a given processor in a given environment or even a specific hardware circuit performing some operations. The two last descriptions differ from the first one since they exploit the physical point of view of performing the operations. And when looking at this layer of the operations, one can imagine that he could extract some information about the secret from potential leakages smartly looking at some channels. The purpose of this section is to present some channels that could be looked at when a device is performing cryptographic operations. There are obviously way more channels that could be presented but the purpose of this thesis is not to cover all of them but only those encountered during this work.

2.3.1 Power consumption

As mentioned above, a cryptographic primitive and its implementation algorithm are running on a specific processor in a given environment if they are software implemented and are related to a particular hardware circuit if they are hardware implemented. In both of these cases, the process works using electric signals, performs some computations on the input and then returns the output, consuming some possibly data-dependent amount of power. This amount of power obviously varies with all the activities occurring on the cryptographic device (transistors,...). In [27], Standaert states that the origin of the data-dependent power consumption of a digital circuit containing CMOS gates (a large part of them) occurs when the capacitors of this gate are charging providing side-channel information leakages. This consideration highlights the fact that power consumption forms a good channel to look at. Furthermore, power analysis is an easy task for any person having an oscilloscope and probes.

2.3.2 Electromagnetic radiation

As suggested in [27], using the Biot-Savart law:

$$\hat{dB} = \frac{\mu_0 I \hat{dl} \times \hat{r}}{4\pi r^2},$$

where dB is the elementary magnetic field, μ_0 represents the magnetic permeability, dl the infinitesimal length of the conductor carrying an electric current I, is the unit vector to specify the direction of the distance vector r starting from the

current to the field point. If the power consumption is data-dependent, then the electromagnetic radiations should lead to information leakages due to its current dependency and to the fact that the field orientation depends on the current direction. This fact makes electromagnetic radiation also a good channel to look at. For some devices it is preferred to power consumption since it is as easy to implement (a probe and an oscilloscope) and for some devices (such as smart card), the internal circuits are not easily accessible so electromagnetic probes make that task easier.

2.4 Side channel attacks

Taking advantage of the side-channel information to perform an attack and then trying to recover the secret of a cryptographic device is what is called a side-channel attack (SCA).
This section will therefore present an overview of the different SCAs encountered during this work, it will help to understand the context in which they can be used respectively, in order to understand the final choice of the attack.

2.4.1 Differential Power Analysis

The Differential Power Analysis (DPA) is a kind of side-channel attack that was first presented by Kocher et al. in [28]. In this type of attack, the attacker uses multiple inputs (also called plaintexts) for a targeted cryptographic operation whose key remains unknown, and analyzes the power consumption. It is important to notice that it is therefore only possible if the attacker controls the input of the cryptographic operation. Then, the goal is to predict the value of one key-dependent bit computed by the device, by trying different guess keys. This will be made possible thanks to the leakage which is dependent on the true value of this targeted bit.
After that, in order to find the correct key among all the guesses, a particular distinguisher that depends on the kind of DPA is used. This distinguisher is an algorithm which allows to class the different guess keys thanks to probabilities computed from gaussian templates (e.g. for a Template Attack) or correlation rate (e.g. for a CPA), assuming that it would be maximized with the correct guess key. In the case of the Kocher's DPA, the distinguisher uses the differential in order to recover the correct key. This distinguisher is the following:

$$\Delta_D[j] = \frac{\sum_{i=1}^{m} D(P_i, b, K_s)T_i[j]}{\sum_{i=1}^{m} D(P_i, b, K_s)} - \frac{\sum_{i=1}^{m}(1 - D(P_i, b, K_s)T_i[j])}{\sum_{i=1}^{m}(1 - D(P_i, b, K_s))} \tag{2.7}$$

where the function $D(P_i, b, K_s)$ corresponds to the computing value of the bit b with the key K_s for the plaintext P_i.

2.4.2 Simple Power Analysis

The Simple Power Analysis (SPA) is the most basic side-channel attack and consists of measuring the power consumption of a device [28]. This power consumption is used to deduce the operations that are performed. The main limitation of this attack is that the observation of the leakage is always made with the same inputs for the targeted algorithm, so it is used when the attacker doesn't control the inputs, thus when DPA is not applicable. So as mentioned in [29], the SPA typically targets variable instruction flow, which is a big limitation if one wants to recover information that are data-dependent.
One has to notice that the fact that the input is constant does not prevent to make profiled SPAs, using a multitude of traces for the same inputs. This kind of approach is notably used by the Soft Analytical Side-Channel Attack developed below.

2.4.3 Template Attack

The Template Attack (TA) is a kind of DPA. In this attack, we assume that one has an available device similar than the one we want to attack and we will run two phases: the profiling phase, which is done on the device of the attacker and then the online attack, which is done on the real device.

1. **The profiling phase** consists in building a model which characterizes the leakages for a certain targeted byte and then we will use this model in order to retrieve the secret during the attacking phase. The model can be chosen by the attacker but a typical model is the normal:

$$\mathcal{N}(l \mid \mu_i, \sigma_i) = \frac{1}{\sigma_i \sqrt{2\pi}} e^{-\frac{(l-\mu_i)^2}{2\sigma_i^2}}$$

   where μ_i is the average leakage value corresponding to the target value i, and σ_i its standard deviation. The goal of the profiling phase is therefore to characterize with known plaintexts and keys the different parameters of the models (means and variances for a Gaussian) corresponding to every possible output byte at a specific time sample (i.e. one that gives us a good information about the target value).

2. **The online attack** phase works as follows: The templates can be used to estimate the probability of each subkey byte. For each power trace measured (with a plaintext byte x), this probability is characterized by:

$$\Pr[x = i] = \frac{\mathcal{N}(l \mid \mu_i, \sigma_i)}{\sum_j \mathcal{N}(l \mid \mu_j, \sigma_j)}$$

The probabilities corresponding to different traces can then be combined in order to detect the correct subkey byte. The distinguisher is thus the combination of the probabilities, we take the logarithm of these obtained for each trace and sum them in order to obtain the likelihood for each key guess. Proceeding that way, we can then find a key candidate which will have the maximum log-likelihood.

These attacks aim to optimally extract information from the leakage if the model built during the profiling phase is accurate [30].

2.4.4 Soft Analytical Side channel Attack

The Soft Analytical Side-channel Attacks (SASCA) were introduced in [31] from which this section is widely inspired. They work by operating directly on the models for several intermediate values computed during a profiling phase using leakage traces to obtain conditional probabilities on the side-channel leakage (see previous section). This attack works in three parts:

1. Construction phase. The first step is to build a "factor graph". This is a directed bipartite graph with two types of nodes:

   - **Variable nodes** representing intermediates variables in the targeted operations. One can note a first particularity of this attack, different intermediate variables are targeted in the same time.
   - **Function nodes** representing the apriori knowledge on variables and the operations linking these ones which are usually atomic operations.

   The edges of the graph can carry two types of messages through the graph, messages from variables to function and messages from function to variables.

2. Information extraction phase. By performing a TA, the probabilities of all the intermediate variables of the implementation can be computed and added as function nodes to the factor graph.

3. Decoding phase. This last step consists in decoding the factor graph using the Belief Propagation (BP) algorithm. This algorithm aims to propagate the information about variable nodes.

The Belief Propagation algorithm

The BP algorithms facilitates the global marginalization computing problem, exponentially complex depending on the number of variables, by replacing it with

several local marginalization and message passing techniques.
In the case of the factor graph presented above, the algorithms will compute marginal probability distributions of each node which is informed, through the edges, by its neighbours of their marginals states and then itself propagates its own marginal along edges of the graph. This process has to be performed until convergence of the marginals.
In order to formalize it a bit, let's denote α_i be the i^{th} intermediate value $i \in 1, ..., N$, f_i be i^{th} function node $i \in 1, ..., M$ and define $\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^{N}$. One can define the function P^* of α as :

$$P^*(\boldsymbol{\alpha}) = \prod_{m=1}^{M} f_m(\boldsymbol{\alpha}_m)$$

where factor $f_m(\boldsymbol{\alpha}_m)$ represents a function of a subset $\boldsymbol{\alpha}_m$ of $\boldsymbol{\alpha}$. This function is depicted using the factor graph described above. In our case, we are interested in the marginalization of this function, so we aim to compute the following function :

$$Z_n(\alpha_n) = \sum_{\boldsymbol{\alpha}, \boldsymbol{\alpha}_n = \alpha_n} P^*(\boldsymbol{\alpha})$$

and its normalized version $P_n(\alpha_n) = Z_n(\alpha_n)/Z$, where

$$Z = \sum_{x} \prod_{m=1}^{M} f_m(\boldsymbol{\alpha}_m)$$

.

As said above, the BP algorithms works by sending messages between variables. These messages represent scores for different possible values for the variable nodes. The algorithms works as follows for updating scores:

0. Initialization of the algorithm :
   The initialization of the algorithm, all the messages of type q are initialized with no information on the variable:

   $$\forall n, m, \alpha_n : q_{v_n \to f_m}(\alpha_i) = 1$$

1. Messages from variable node to factor node:

   $$p_{v_n \to f_m}(\alpha_n) = \prod_{m' \in \mathcal{M}(v_n) \setminus m} r_{f_{m'} \to v_n}(\alpha_n)$$

   In this formula, $\mathcal{M}(v_n)$ denotes the adjacent nodes of the variable node v_n. In this equation, the variable node sends to f_m the product message about α_n received from all its neighbours excluding f_m.

2. Messages from factor node to variable node:

$$r_{f_m \to v_n}(\alpha_n) = \sum_{\boldsymbol{\alpha}_m \setminus \alpha_n} \left(f_m(\boldsymbol{\alpha}_m, \alpha_n) \prod_{n' \in \mathcal{N}(f_m) \setminus n} p_{v_n, \to f_m}(\alpha_{n'}) \right)$$

This formula represents the function node f_m sending to the variable node v_n the sum over all the possible values (α_m) of its neighbours ($\mathcal{N}(v_n)$), except v_n fixed to α_n. The elements in the sum are the products over the message received from all the neighbours of the current node except v_n about their value in α_m.

The BP algorithm works by iterating these rules on the whole graph. It converges if the factor graph is tree-shaped but if it contains cycles, there is no guarantee that the algorithm will converge and will lead to the correct solution. But [31] mentions that it usually still gives a result good enough for many applications. The normalized marginal function of a variable α_n can be recovered, once the network has converged, by multiplying all the incoming messages at its node:

$$P_n(\alpha_n) = \frac{1}{Z} \prod_{m \in \mathcal{M}(n)} r_{m \to n}(\alpha_n).$$

2.5 Rank estimation and key enumeration

For the attacks which consist in recovering separate parts of a secret, one can define a metric which will give the computational effort that the attacker has to provide in order to retrieve the entire secret knowing the computational effort needed to recover the subkeys independently. That's the purpose of key enumeration and rank estimation. These two algorithms are well detailed by Poussier in [32].
The key enumeration will be used in the context where an attack would fail at entirely recovering the secret, and it consists in trying exhaustively but smartly the most likely keys.
The rank estimation will be used to compute an estimation of the rank of the entire key using those of the subkeys. This will allow the attacker to estimate if recovering the entire key is accessible with its computational power.

Chapter 3

Bitcoin overview

The aim of this chapter is to present an overview of the Bitcoin, introduced and developed by Satoshi Nakamoto in [33]. The main purpose of this section is therefore not to provide all the aspects of the Bitcoin rigorously but just to enable the reader to get familiar with its functioning and thus have a better idea of the usefulness of the hardware wallets studied by this work. **Thus, this section contains simplifications**. For more details, see the excellent book of Antonopoulos [19] from which this section is inspired.
The Bitcoin regroups all the concepts and technologies which allow the entire ecosystem to be operational. The currency stored and exchanged in this Bitcoin ecosystem is also called the bitcoin, but it only represents the first application of the Nakamoto's invention which consists of 4 parts:

- *The Bitcoin Network:* A decentralized peer-to-peer network.
- *The blockchain:* A public transaction database.
- *The consensus rules:* A set of rules for decentralized and independent validation of transactions and currency creation.
- *A Proof-of-Work algorithm:* A mechanism helping to reach decentralization.

The motivation behind these innovations is to allow secured financial transactions that will be sent directly by the payer to the payee without going through a financial institution, i.e. "A Peer-to-Peer Electronic Cash System".

3.1 Blockchain

The blockchain in the Bitcoin environment is a kind of public database containing the history of all the bitcoin transactions. More precisely, its structure is an ordered

linked list of blocks containing transactions. An illustration of the blockchain structure with simplified blocks contents can be found in the Figure 3.1 below:

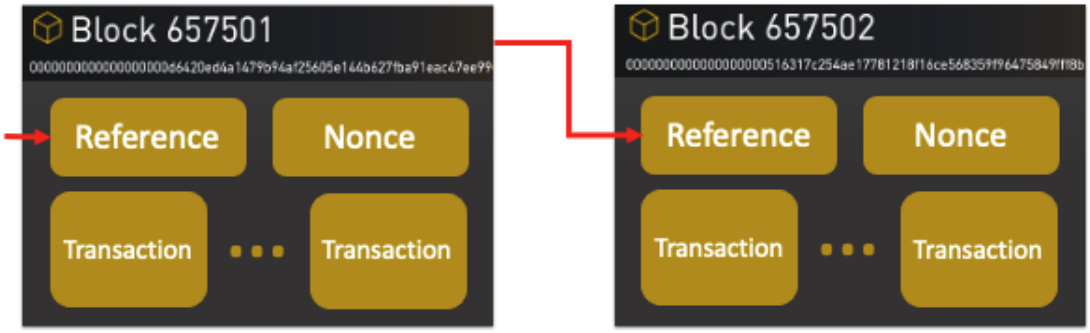

Figure 3.1: Blockchain structure

As it can be seen on the illustration, a block contains an identifier (built by hashing its content concatenated with the hash of its predecessor), a reference to its predecessor, several transactions and finally a nonce, whose utility will be presented later.
The main purpose of the blockchain's particular structure is to reach the decentralization, but it will be explained in a next section.

3.2 Bitcoin transaction

As the goal of Bitcoin is initially to enable a peer-to-peer electronic cash system, transactions are obviously an important part of the protocol. The purpose of this section is therefore to explain the creation of a valid transaction. There are different types of transactions but in this thesis, only the "Pay-to-Public-Key-Hash" transactions (P2PKH) will be developed. In this kind of transaction, the payer makes the transaction to a public key of the payee.
As the blockchain is public, everyone may pretend to write something on it. The network has therefore to ensure that what is written on the blockchain corresponds to valid transactions. The transaction is valid if and only if 2 things are provided by the payer:

1. *Proof of ownership: Does the payer has the bitcoins he wants to spend?*
   A bitcoin is defined as a chain of electronic signatures. If one wants to make a transaction using one of his private keys linked to his bitcoins, he needs to make a reference to a previous transaction in the blockchain, which was intended to one of his public keys. Such a previous unused transaction

includes the address (the hash of the public key) of the payee. Therefore, as one can only spend transactions that were destined to him, the system can verify that these transactions exist and then certify that the payer has the bitcoins he wants to spend. This avoid the double spending since once a previous transaction is used, it can't be reused later.

2. *Authentication: Is the person writing the transaction on the blockchain the owner of the bitcoins he wants to spend?*
   The proof of ownership will be provided by the payer's private key via a digital signature. Indeed, if the payer signs a previous transaction destined to him with his private key, anyone can verify that the address contained in this previous transaction matches with the signature. Thus, the user is authenticated and will not be able to refute the transaction.

This process is illustrated on the Figure 3.2. The verification process will be explained in the next section.

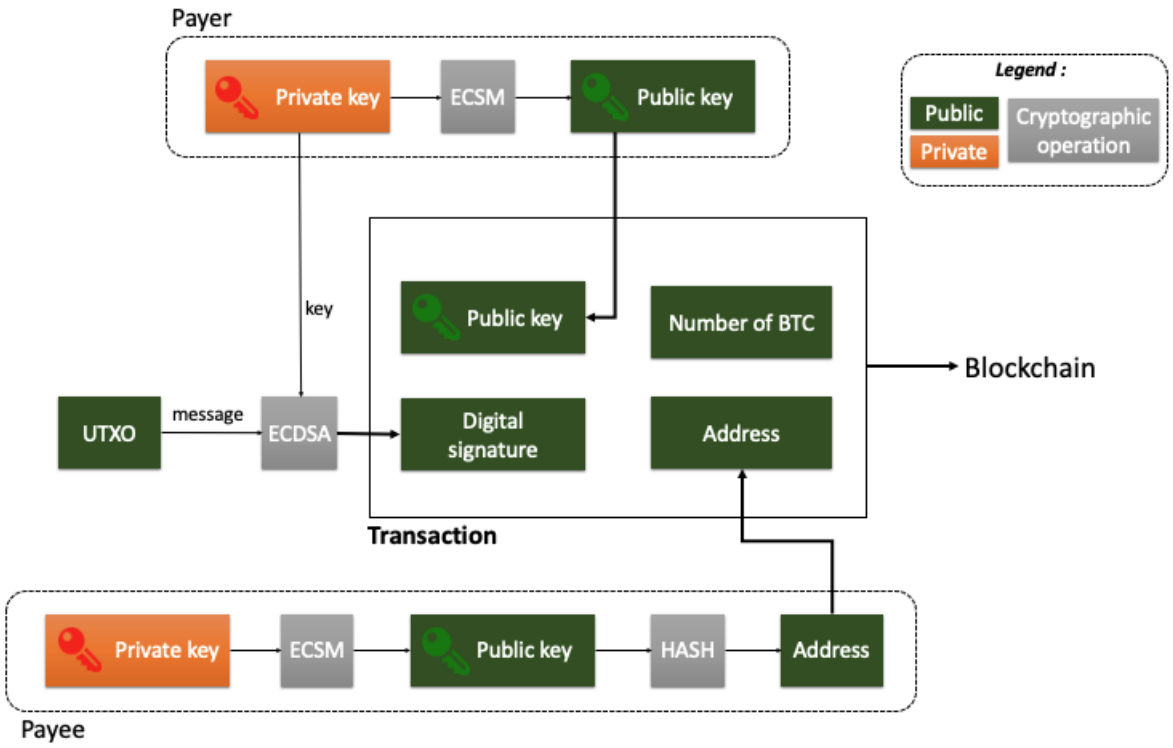

Figure 3.2: P2PKH Transaction

3.3 Mining

Once the transaction has been sent to the network, it is still not written into the blockchain until it is verified by the process called mining. This is a two-fold process, the main purpose is to support the decentralization objective of the bitcoin network by validating and clearing transactions to put them on the blockchain. Also, the mining process has the effect of increasing the money supply in circulation, as it creates bitcoins.
The first part of the process will be to verify if one block size (1MB) of transactions correspond to the consensus, and check that the block is correctly formed, which includes many verifications: check the block size, check each transaction signature, look for double-spending,...

The second one is a proof-of-work mechanism. It consists of trying exhaustively to find the nonce, a number of fixed length added to the current block, for which the hash of that block will be smaller or equal than a fixed given value. This problem can only be solved thanks to brute force methods because of the preimage resistance of the hash function and it requires a huge power of calculation. The first miner who successfully retrieves the nonce and publishes a block is rewarded with brand new bitcoins that are paid to him and therefore enter the network.
Once a new block has been published, all the miners have to check if its content respects the consensus before trying to mine a new one. If the published block's content doesn't respect the consensus, the miners will still work on this previous block until a new one is published and accepted by the majority of the network.
This mechanism therefore guarantees the reliability and decentralization of the Bitcoin network.
Although innovative, this system is very energy-consuming. As an indicator, the whole Bitcoin network performs approximately 11^{20} hashes per second [34] consuming approximately 145 Twh of electricity each year, generating 7180 tons of e-waste and 70 Mt of CO_2.

Chapter 4

Satochip Hardware Wallet

This thesis is made in partnership with Satochip (Secure Anonymous Trustless and Open Chip) [1]. It is an open-source wallet running on hardware devices such as smart cards or USB keys and supporting many cryptocoins (Bitcoin, Ethereum, Litecoin,...). The wallet in the Bitcoin environment is used to control access to the user's coins, managing the keys and creating/signing transactions. It represents the primary user interface with the Bitcoin environment.
The commercial version of the applet is provided on smart cards from NXP running on a specific operating system, Java Card Open Platform (JCOP). This commercial version therefore relies on the security of the NXP chips. Such a system is widely used in the industry as in banking, healthcare and public transportation sectors.
The Satochip software is open source and can be found in [35]. And the Satochip version under investigation in this thesis is the v0.6-0.3.
In this chapter, a presentation of the smart card studied in this work will be done, and more particularly its components, operating system, functioning and the limitations that it involves. Secondly, the structure of the bitcoin wallet and the different key derivation mechanisms will be described. Finally, the mechanism of signature for a bitcoin transaction in the hardware wallet will be presented.

4.1 Smart Card NXP JCOP3H145

This section presents an overview of the NXP smart card J3H145. The functioning and the features of the card are developed in detail as they are required to understand all the opportunities, limitations, difficulties and hypothesis that this kind of devices imply for a side-channel attacker. More particularly the communication protocol and the memory management have a big impact on the recording of the traces because they will respectively impose the way of triggering during the

[1] https://satochip.io

recording of the traces and ensure sufficient speed of execution of operations. On the other hand, the elements present on the card will be interesting when choosing the path of attack. It is therefore an important section to understand the choices that will be made in the rest of this work.

4.1.1 Structure and components of the chip

The functional diagram of the card, available in the data sheet [36], is shown in Figure 4.1.

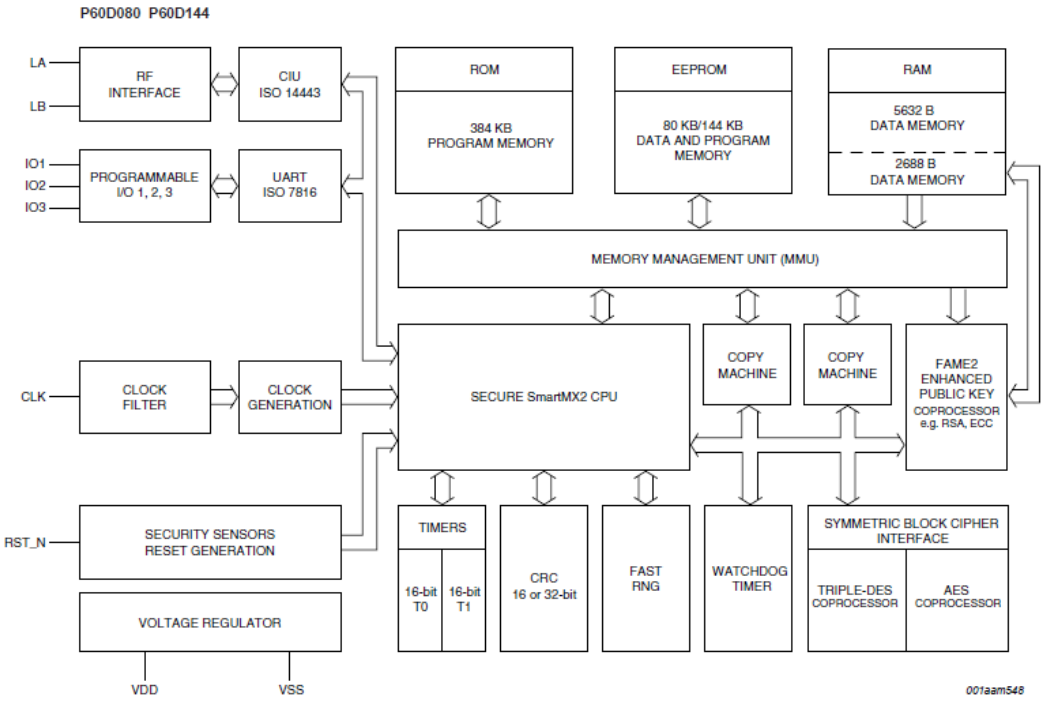

Figure 4.1: Functional diagram of the NXP P60D080 chip

One can see that the CPU on the card is the SmartMX2 P60 family. This 16-bits processor introduced the IntegralSecurity™ architecture. This architecture includes more than 100 security features and works with powerful co-processors allowing high level performance for cryptographic operations. Notice that 3 co-processors are present on the diagram: the first one is a new generation Fame2 crypto coprocessor dedicated to ECC and RSA, the second carries triple-DES and the last one supports AES.
The communication with the card is made through to the 3 pins I/O with the

ISO7816 UART protocol [37]. In practice, the communication used during this work is some abstraction levels above UART, in the applicative layer, because it is limited by the Java Card API that doesn't allow all the functionalities of UART. Those limitations will be developed in a following point.
There is also a pin to probe the clock signal on the card. This one is useful for a side-channel attacker as it allows to verify if there is jitter, which could lead to a misalignment of the different leakage measurements.
Another important point to notice is that the card uses different types of memories: ROM, EEPROM and RAM.
The other elements visible on the diagram are not presented here because they do not serve the direct interests of the thesis.

4.1.2 Operating system and Java Card

The operating system of the card is Java Card Open Platform 3 (JCOP3), also developed by NXP, and it is an implementation of the Java Card 3.0.4 classic edition. The Java Card technology combines a subset of Java programming language and a runtime environment (JCRE) optimized for smart cards.
In Figure 4.2, one can find a complete Java Card application and its components.

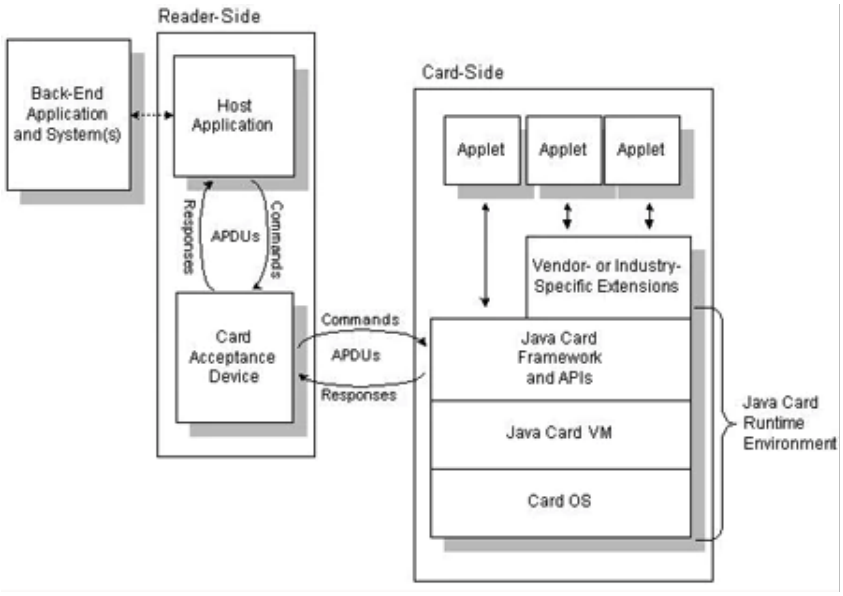

Figure 4.2: Architecture of a Java Card Application, by Oracle.com

In this figure, the host application represents for example a PC, an electronic payment terminal, ...
The Card Acceptance Device (CAD) is typically a card reader forwarding messages

exchanged between the host application and the card.
On the card side, there may be one or several applets, a native OS, a Java Card Framework, APIs and a Java Card Virtual Machine which defines a subset of Java compatible for smart cards. That's this part which will interpret byte code, manage objects, ...

4.1.3 Java Card Applet

The different type of communication between the JCRE and the applet are available in Figure 4.3.

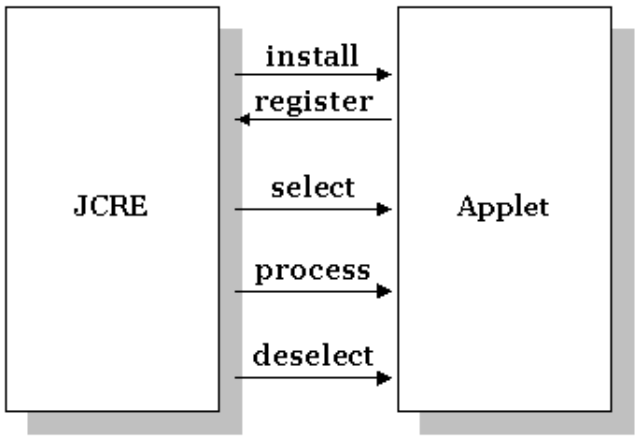

Figure 4.3: The Java Card Applet Methods, by `Oracle.com`

The methods in the figure are the ones which must be implemented in an applet.

- The `install()` method will instantiate the applet, it will call the `register()` method to complete the installation.
- The `select()` method is used to identify the applet that must process the APDU received with the `process()` method (see next section).
- The `deselect()` method doesn't have to be implemented but, for example, it can reset the PIN.
- The `process()` method will therefore process the APDU received and redirect to a specific method of the applet depending on the content of the APDU.

4.1.4 Communication protocol

As mentioned above, the communication protocol supported by the smart card is the Universal Asynchronous Receiver Transmitter (UART) protocol. It meets the international standard ISO7816 described in detail in [37].
The message-passing model for Java Card applications uses Application Protocol Data Unit (APDU), that's a data packet exchanged between the CAD and the framework. Its structure is shown in Figure 4.4.

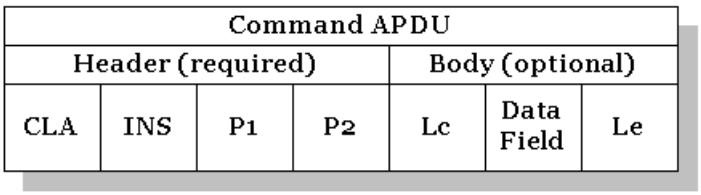

Figure 4.4: Command APDU, by `Oracle.com`

The CLA specifies the norm used for the communication, INS indicates a specific instruction such as communication channel management (it means that that's the host application which chooses the communication channel). P1 and P2 can be used for other INS or for data. Lc is the length of the data in the APDU and Le the expected length of the response data.

Once the APDU is received by the card, this one will treat it thanks to the `process()` method that redirects it towards the appropriate function in the applet. Then, once the function has been executed, the applet sends a response to the computer under another format presented in Figure 4.5

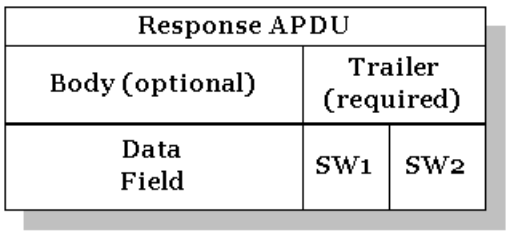

Figure 4.5: Response APDU, by `Oracle.com`

Were SW1 and SW2 are status response characterising if the process went well or if it failed. In this last-mentioned case, this is the type of error that is sent.

4.1.5 Difficulties based on the card

Memory management

In Java Card there is no garbage collector, unlike in Java, in order to free automatically the memory linked to temporary declared variables. It makes the memory management more complicated for the programmer and a bad management in the targeted device software can, at some point, complicate the task of the side-channel attacker. As mentioned above, there are 3 types of memory on the card: RAM, ROM and EEPROM. The first one is a volatile storage while the two others are non-volatile. The management of the memory in Java Card is well detailed by Zhiqun in [38].
The first difficulty occurs if there are some allocations of objects non-specifically stored in the RAM. Indeed, every occurrence of the `new` Java keyword will store the object in EEPROM by default. This process is 1000 times slower than writing on RAM and can wear this persistent memory [38]. Then, for the attacker, it will slow down the trace acquisition and wear out faster the smart card.
The second difficulty appears when objects are stored in the transient memory by the developer. This kind of memory can only be cleared by deselecting the applet or disconnecting the card. So, when performing many processes in a row, the memory gets filled quickly since the same object is declared in a new part of the memory at every process. Then, when the card's RAM is full, it will throw an error in order to free its memory. This would not especially be a problem for a daily usage since the card is frequently disconnected of the CAD. But for an attacker it could slow down the trace acquisition because freeing the memory is made slowly by the card ($\approx 1s$). As an example, let's say there's only an array of size 32 declared in transient memory during the targeted process, the attacker would therefore lose 1 second every 20 traces taken. Knowing that, with our setup recording 20 traces every second, the bad memory management would decrease the number of traces taken by a factor 2 in this case.
So, if the targeted device's software has some allocations of objects in transient memory during the process, it will cause troubles to the attacker by slowing down its traces acquisition.
Fortunately, there is a common solution to these problems: the declaration of the objects used by the applet in the targeted process during the trace acquisition has to be done in the transient memory and during the creation of the applet. It will therefore be declared only once for all the traces. By doing it, the attacker uses the quickness of the RAM without risking to fulfill it too quickly. By proceeding that way, the card won't allocate new space in memory for the same objects given it is not declared in each process anymore but it will rather overwrite the previous one. Then, there will not have memory problems or speed problems during the

trace acquisition phase.

Triggering

The second difficulty while working with Java Card is triggering, or trying to frame operations of interest in the side-channel leakage during the trace acquisition phase.
Indeed, as the operating system of the card is JCOP, the card therefore only supports Java card natively and the user is limited to the methods available in the API. Working with APDUs to communicate implies that there are limitations due to this communication protocol. In the literature, this problem has been discussed by Mangard et al. in [39] and Mateos et al. in [40].
In an optimal way, the attacker would like to trigger just before the operation of interest in order to have a good alignment of the traces and to store only information of interest. It is typically made by sending an indicator, a signal recognizable by the Picoscope, over a channel just before the operation of interest. Furthermore, by doing it, the attacker isn't constrained to capture the processing, receiving and sending of the APDU in its traces.
However, the task is going to be more difficult with the functioning of Java Card. Indeed, when sending an APDU, that's the host application (see Figure 4.2 and Figure 4.4) that chooses a channel of communication (the one in which it sends the APDU). After sending the APDU, the host waits for one unique response over this channel. Furthermore, once the APDU has been processed, the applet has not the possibility to send messages over another channel that the one indicated by the host application initially. This is not allowed because of the limitations of the Java Card API. Knowing it, the card can't send a signal over the channel just before the operation of interests since it can only send one message at the end of its operations (the response APDU Figure 4.5).
Then, the method used for triggering in this master thesis, considering these difficulties, will be described in a following chapter.

Co-processors and hidden implementations

The last difficulty developed in this chapter is that some of the algorithms are implemented by the platform and the implementation details are not available. Furthermore, as there are cryptographic co-processors, some of the cryptographic operations are implemented in hardware and NXP states that their chip implements countermeasures against side-channel attacks [36]. Attacking this kind of implementations could therefore lead to potential huge reverse engineering efforts.

4.2 Hierarchical Deterministic wallets

It is important to understand that a bitcoin wallet will not contain bitcoin, but keys linked to bitcoins. Managing these keys and signing transaction requests submitted by the host are the only concerns of the wallet.
As presented in the previous chapter, in order to sign bitcoin transactions, the users have to use a key. The particularity of the bitcoin environment is that a key will only be useful either to derive other keys or to perform transactions. Moreover, a common practice is to perform **only one** transaction for each key even if nothing in the wallet prevents a key from being reused. By working that way, the number of keys to store may quickly become too important for the small memory of the card. That's why Hierarchical Deterministic (HD) wallets defined by the BIP32 standard have been introduced [41]. In this standard, the keys are "derived" in a tree shape (see Figure 4.7), which means that each key is computed based on the previous one, which is therefore at the upper level in the tree. During a derivation process, the new key that has been derived is called "child key" and the key from which the new has been derived is called "parent key". The main interest of this standard is that it allows to recover all the keys from the top of the tree, so there is only a seed, made of bits of entropy (between 128 and 512 bits), that must be stored in the card. When a signature must be effectuated, the key needed is derived from the seed with one of the two derivation modes detailed below.
It is also important to notice that it is common practice to use the leaves of the tree in order to sign transactions even if it is not always imposed by the wallet. The importance of the keys belonging to the intermediate levels, between the seed and the leaves, is detailed further.

4.2.1 Master key, extended keys and chain code

As said in the introduction of this section, a HD wallet only stores a seed from which it derives keys. The first key that must be derived is called the master key of the wallet and this derivation process is shown in Figure 4.6.

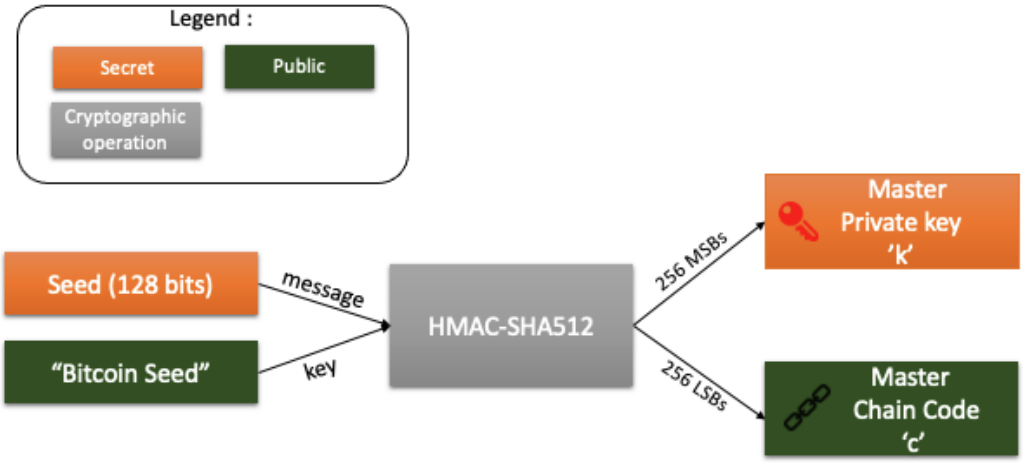

Figure 4.6: Master key derivation

As it can be seen in the scheme, the output of the HMAC SHA-512 block is split in two, the 256 left bits (the Most Significant Bits, MSBs) constitute the private key and the right ones (the Least Significant Bits, LSBs) are called the "chain code".
At this point, it is important to define to notion of extended keys. To increase the security of the derivation at the lower levels, 256 bits of entropy are added to the pair private-public key and these bits are the chain code. The chain code and the key will form an extended key, represented as a pair (k, c) with k the key and c the chain code. When child keys will be derived, it will therefore not only depend on the parent key but also on the chain code, which will increase the security of the derivation.

4.2.2 Index

Each key has an index, going from 0 to 2^{32}. There are two types of keys depending on their derivation process: the hardened keys and non-hardened keys. A non-hardened key has an index going from 0 to $2^{31} - 1$ and an hardened key has an index going from 2^{31} to 2^{32}. The difference in the derivation mode between these two types of key will be defined later.
An example of tree formed by the seed and the keys can be seen in Figure 4.7, which comes from [41]. The different levels of this tree are determined by the BIP44 standard [42]. The indexes of the keys at each level, from the top of the tree to the key to derive, form together the path of derivation. This path is needed when a derivation is performed, as it mentions all the intermediate keys to be derived from the master key in order to reach the desired one.

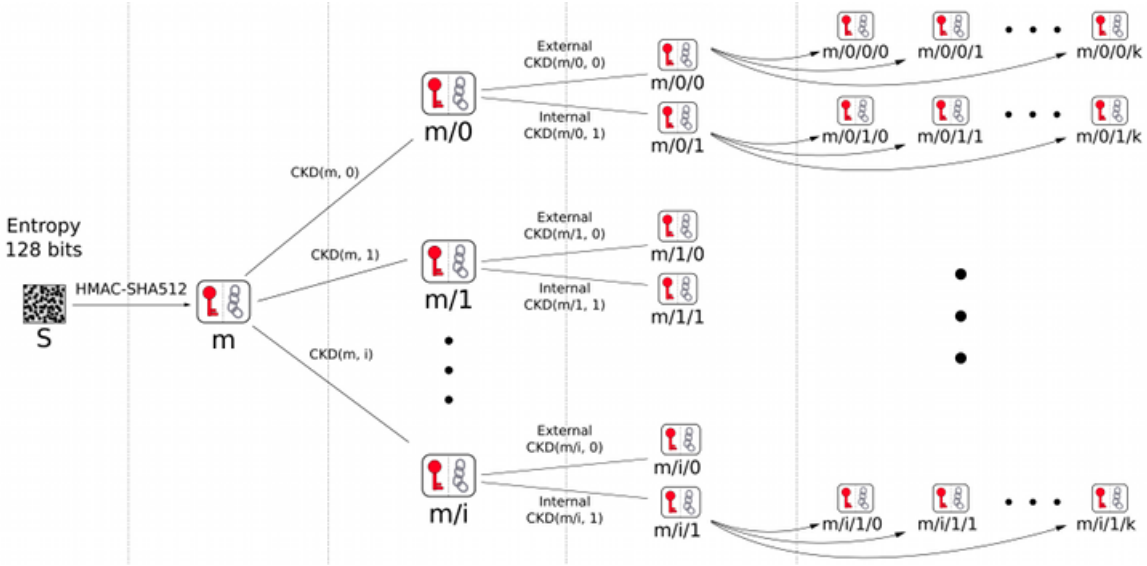

Figure 4.7: Wallet structure

4.2.3 Path levels

The five levels of the tree are shown in Figure 4.8. The levels with an apostrophe are the ones made of hardened keys. Each level has a meaning:

- **m** This is the level of the master key, the top of the tree.
- **purpose** It is a constant set to 44, which indicates that the BIP44 protocol is used for the next levels of the tree.
- **coin_type** The BIP32 wallet structure can be used for many cryptocoins as Bitcoin, Litecoin, Dogecoin, etc... At this level, sub trees are created for each type of coin. Coin_type is a constant set for each type of coin. The constant for the Bitcoin is `0x80000000`.
- **account** This level splits the wallet into several accounts, whose indexes start from 0 and are increasing.
- **change** There are only two values for the index at this level. The value 0 is used for the external chain, which contains all the keys that are meant to be visible from outside the wallet in order to receive payments. The value 1 is used for the internal chain, which contains the keys used to return transaction change.

- **address_index** This is the last level of the tree, these are the keys used for signing transaction. The indexes are starting from 0 and are increasing at each transaction.

```
m / purpose' / coin_type' / account' / change / address_index
```

Figure 4.8: Path levels with BIP44 protocol

4.2.4 Child Key derivation

As the wallet uses the chain derivation, child extended keys need to be derived from a parent extended key (key + chain code) and the index i of the key to derive. The derivation algorithm is different if the child key is private or public and if it is hardened or not. The three different derivation modes are explained below.

Non-hardened derivations

The first type of non-hardened derivation is the public child key derivation. This kind of derivation is particularly useful because it allows to derive a child public key from a parent public key, so the parent private key linked to the parent public key doesn't have to be known to derive a child key (Figure 4.9). This is interesting if you have someone in charge to make sales for you (e.g. a website) and you want the customers to pay on your Bitcoin account. The seller can derive as many keys as needed without knowing any of your secret keys.

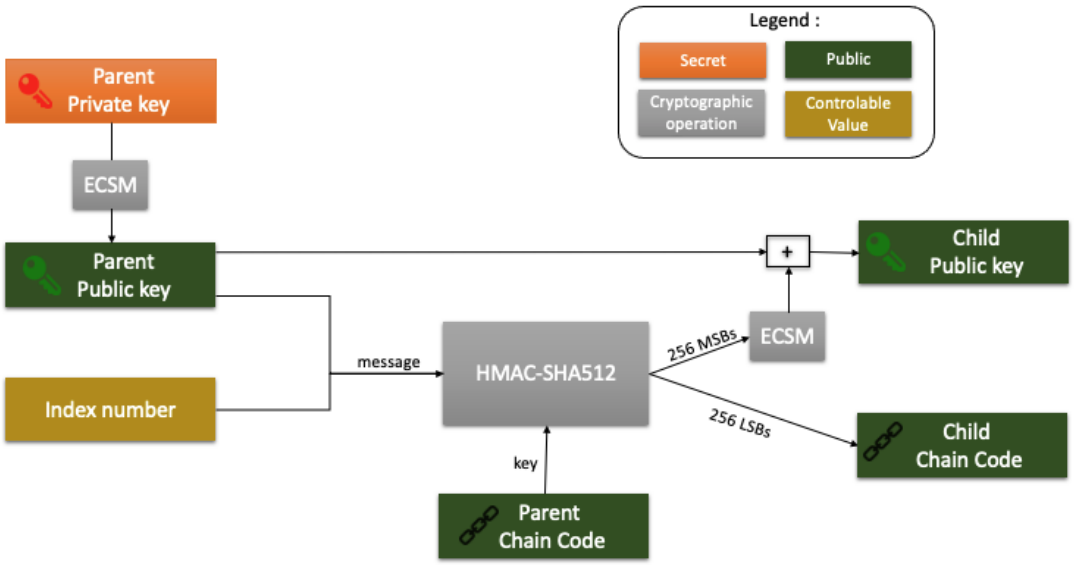

Figure 4.9: Non-hardened public derivation

Figure 4.9 shows the derivation mechanism, which consists in a HMAC-SHA512, whose key is the parent chain code and the message is the concatenation of the parent public key and the index of the child key. Just as for the master key, the output is divided in two blocks of 256 bits each. The block containing the LSBs is the chain code linked to the child extended key, and the block containing the MSBs is used in an elliptic curve scalar multiplication (ECSM, see 2.1.4). The output of this multiplication and the parent public key undergo a point-wise addition (see 2.1.4) whose result gives the child public key.

Moreover, the non-hardened derivation also allows to derive child private keys. This derivation is shown in Figure 4.10. This derivation mode is also based on a HMAC and has the same inputs as the public non-hardened. The difference lies in the computation of the child key. The 256 MSBs of the HMAC's output are added (in modular addition with the order of the elliptic curve N) with the parent private key. The 256 LSBs of the HMAC's output give the child chain code of the extended key. Besides, it is important to notice that the child chain code is common to the public and private key.

It is important to note that this derivation system has a flaw. Indeed, the parent public extended key and the index being public, all the inputs of the HMAC SHA-512 are known by a potential attacker. If this one was able to recover only one child private key derived from this parent public key, he would be able to recover the corresponding parent private key by reversing the modular addition.

This explains why this derivation system is not used at every level of the tree and that's why a hardened derivation system which doesn't have this flaw has been implemented. This hardened derivation method is the subject of the next section.

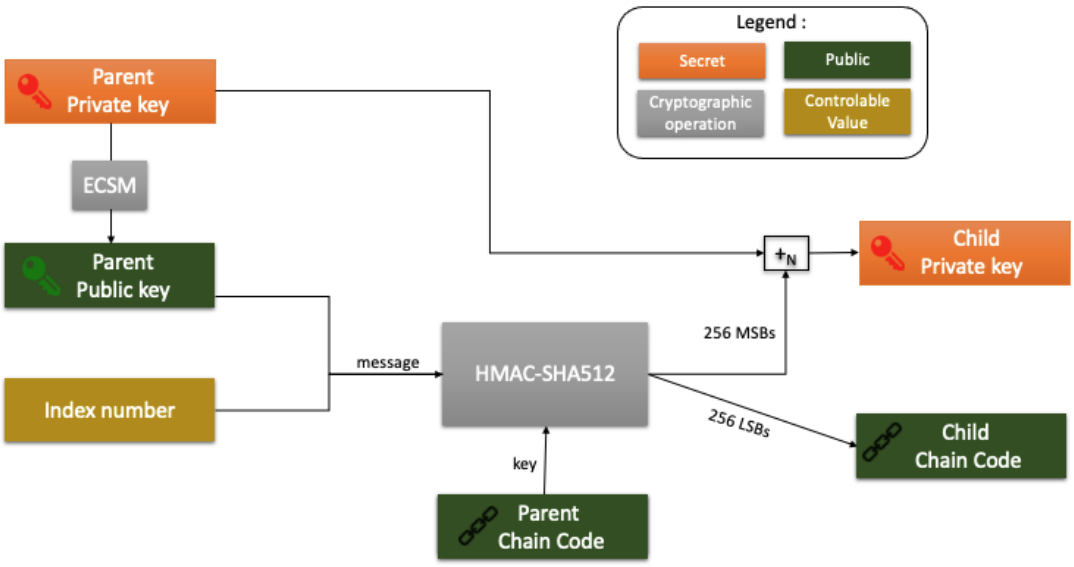

Figure 4.10: Non-hardened private derivation

Now that the two non-hardened derivation modes have been presented, it can be shown that the public key computed with the first derivation mode corresponds to the one computed from the private key which comes from the second derivation mode.
First, let's define a few variables that will be used for the demonstration:

- P_0, the parent public key
- P_1, the child public key
- S_0, the parent private key
- S_1, the child private key
- $EC[x]$, the scalar multiplication $x * G$ with G the starting point of the curve
- $+_p$, the point-wise addition
- $+_N$, the modulo N addition

Then, it is important to remind the link between a public and a private key. Starting with a private key S_0, the corresponding public key P_0 is computed as: $P_0 = EC[S_0]$
For the derivation of the child public key P_1 in Figure 4.9, one has:

$$P_1 = EC[HMAC_L(P_0, i, C_0)] +_p P_0$$

And for the derivation of the child private key S_1 in Figure 4.10:

$$S_1 = HMAC_L(P_0, i, C_0) +_N S_0$$

Which gives,

$$P_1 = EC[HMAC_L(P_0, i, C_0) +_N S_0]$$

So, to verify that the 2 derivation mechanisms give the same public key, the following relation must be verified:

$$EC[HMAC_L(P_0, i, C_0)] +_p P_0 = EC[HMAC_L(P_0, i, C_0) +_N S_0]$$

This equality, demonstrated in [43], belongs to the properties of the elliptic curves. It has therefore been shown that the 2 methods of derivation give the same public key.

Hardened derivation

For the higher levels of the tree, the method of derivation used is called "hardened". The main difference with the non-hardened derivation is that the parent public key is not used in the derivation process as it can be seen in Figure 4.11. The HMAC's inputs are now the parent private key concatenated with the child index number and the key of this HMAC is still the parent chain code. For the outputs, the 256 LSBs are still the child chain code and the 256 MSBs are added modulo N with the parent private key to give the child private key. This prevents a potential attacker of finding the parent private key even if he recovers a child private key. This is then a more secure derivation method than the non-hardened one, but it has not the same advantages. Indeed, in this case it is impossible to derive a public key from another public key.

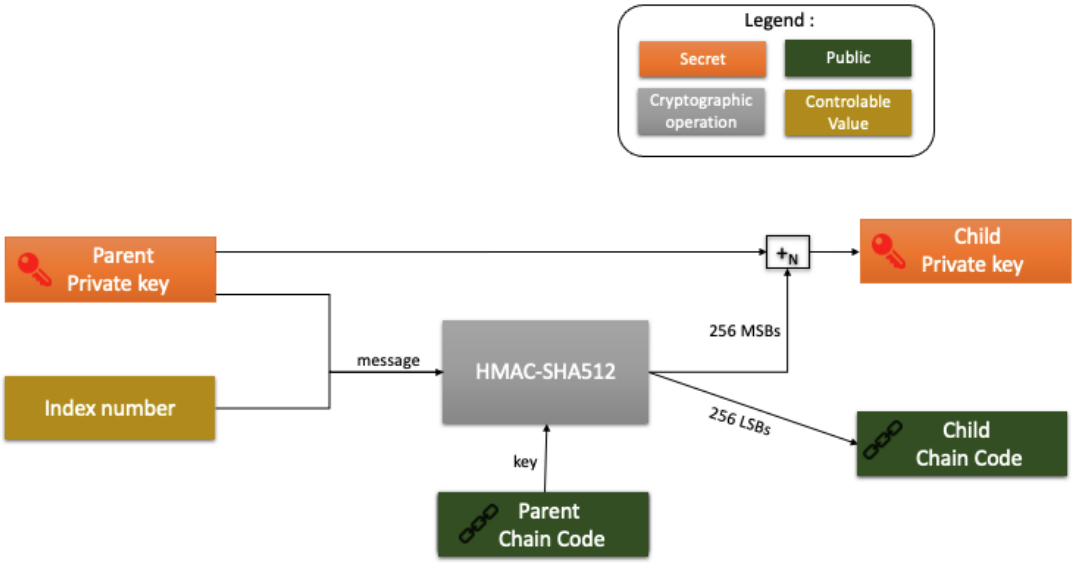

Figure 4.11: Hardened derivation

4.3 Signature of transaction

As it has been said, the Satochip wallet intervenes in the "transaction" part of the Bitcoin protocol and as mentioned in the previous chapter, the transaction mode used by the card during this work is the "Pay to public key hash". For a transaction, 3 main operations are performed on the card (Figure 4.12).

- The key derivation: the private key of the sender that will sign the transaction and the public key that will verify that transaction have to be derived from the seed (see section 4.2.4).
- The hash of the transaction data: The data of the transaction, which is given to the card contains the bitcoin accounts that will send money, the amount of the transaction and the accounts that will receive the money. All those information are hashed directly on the card.
- The signature of the transaction: The hashed data is signed by the private key previously derived and the result of the signature is returned by the card.

As explained in the previous chapter, once the transaction has been signed, this transaction can be verified by the miners thanks to the public key linked to the private one that has just signed it. Then, if the transaction is valid, it can be added to the blockchain.

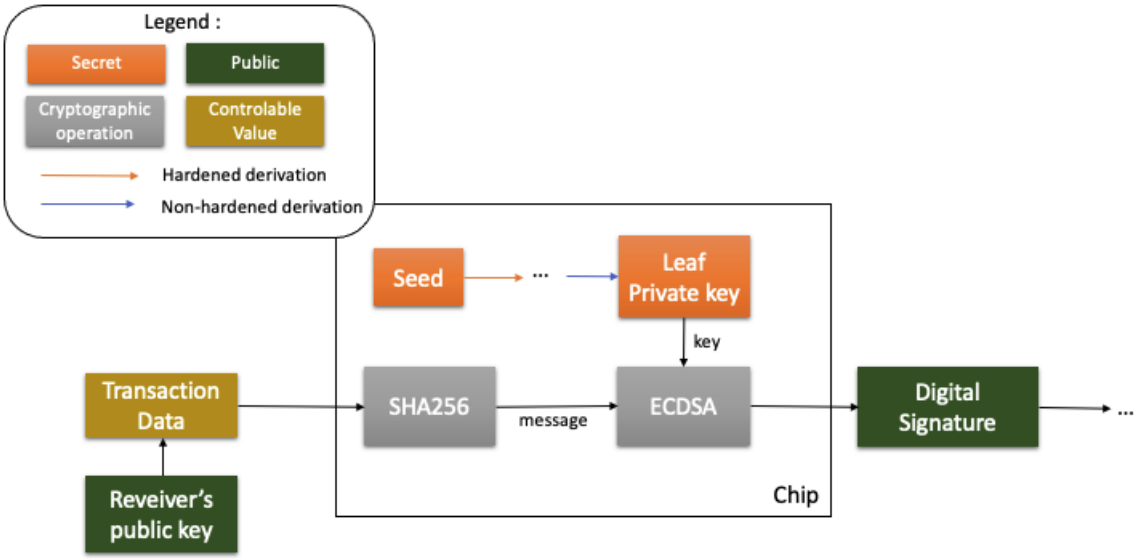

Figure 4.12: Transaction P2PK

Chapter 5

Attack vectors

As the goal of this thesis is to attack the wallet presented in the previous chapter, it is necessary to identify and analyse the possible attack vectors from a side-channel attacker viewpoint. This chapter will first describe the targeted secret of the attacks and evaluate the consequences of recovering it. Then the different ways identified to attack the Satochip hardware wallet will be developed. Finally, based on the difficulty of putting those attacks in place, the choice and the design of the attack that will be studied during this thesis will be argued.
One can note that the PIN code is not part of the attack vectors although it is present on the card. In the framework of this work, it was left aside on purpose as it seemed more interesting and relevant to focus on cryptographic functions. Moreover, the recovery of the PIN code on a smart card with a Template attack has already been done by Bouder et al. in [44] and by San Pedro et al. in [1].

5.1 Targeted secret and impact on the wallet

The goal of the different attacks that will be presented below is always the same: recovering a private key of the wallet.
Indeed, if an attacker retrieves a private key, he can spend all the bitcoins directly linked to it if the key belongs to the leaves of the tree. However, if the key belongs to a higher level of the tree, recovering it could have bigger consequences. (see Figure 4.7). Indeed, when recovering a private key, all the keys that can be derived from it are also accessible (see Section 4.2.4) and then all the bitcoins linked to these lower-level keys can be spent. The impact on the wallet therefore obviously depends on the level of the key retrieved. Higher the level of the key is, greater the impact could be, because it could potentially give access to more keys and then more bitcoins.

5.2 Signature of transaction

The signature of a transaction has already been discussed in a previous section. It is made thanks to the ECDSA which involves a private key of the signer and a message. In the context of this master thesis, the private key is unknown and the message is known and controlled by the attacker as shown in Figure 5.1.

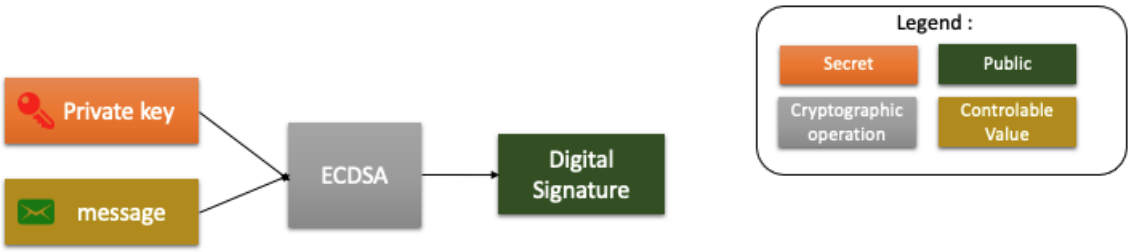

Figure 5.1: ECDSA

Attacks on ECDSA have been developed many times in the literature, and they exploit different flaws, which mainly depend on the implementation of the scalar multiplication on elliptic curves used in the algorithm. For example, Wang and Fan showed in [45] that 85 signatures would be enough to recover the secret key for an implementation using the double-and-add. De Micheli et al. developed an attack on a wNAF implementation for the scalar multiplication in [46], and they were able to recover the secret key with only 3 signatures. But it is important to note that attacking ECDSA is not equivalent to attacking a scalar multiplication on elliptic curves as there is also a random generation to consider in the ECDSA algorithm (see 2.1.5).
The difficulty of our implementation of ECDSA is that the scalar multiplication on the elliptic curves is made in hardware thanks to the FAME2 cryptographic co-processor of the chip. There is therefore not detail on the way it is implemented. This would further complicate the task even if Udvarhelyi has shown in [21] that hiding an implementation is not an efficient countermeasure. Indeed, Roche and Lomné have recently broken the ECDSA of the Google Titan Security Key using 6000 signatures while the implementation was also hidden [47].
Furthermore, our processor is said to be protected against DPA and SPA [36]. In view of those information, even if this attack vector doesn't seem impossible, it is not the easiest to exploit, so one decided to focus on another vector.

5.3 Private to public key conversion

As mentioned in the previous chapter, the Satochip hardware wallet supports conversion of a private key to a public key using elliptic curves. This conversion is performed thanks to a scalar multiplication between a scalar (the private key) and the starting point of the curve, which allows to obtain the corresponding public key. The operation is illustrated in Figure 5.2, in which k is the private key and G the starting point on the curve. The operation to attack here is also the scalar multiplication on elliptic curves and once again, the attack to set up will depend on the implementation of this multiplication. But this time, there is no random generation that comes into play, unlike with the ECDSA. In the literature this kind of attack has already been discussed. Among others, Roche et al. in [48] and Fouque et al. in [49] studied the possibility of attacking a scalar multiplication with a "doubling attack". Nevertheless, this attack vector has not been retained for the same reasons as for the ECDSA.

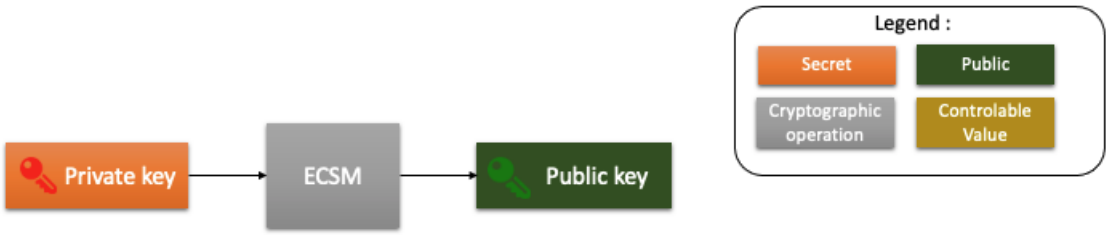

Figure 5.2: Private to public key conversion

From the point of view of the consequences of the attack, this way would have a bigger impact than attacking ECDSA given that it would be applicable to any key, no matter how high it is in the tree of the wallet (see Figure 4.7), unlike ECDSA which only concerns the keys at the lowest level.

5.4 Key derivation

The key derivation algorithm has been developed in detail in Section 4.2.4. The attack studied here will be based on a hardened derivation, which corresponds to the derivations used in the high levels of the tree and is therefore more interesting for an attacker than a non-hardened. The goal of this attack is to recover the secret parent private key from which a child private key is derived. To do so, the HMAC SHA-512 function must be attacked. This has been done several times in the literature as developed in the Introduction.
Moreover, the HMAC SHA-512 on the smart card being software implemented,

we therefore have access to all the implementation which is a benefit compared to the two previous vectors. However, one can note that it is an unusual attack because the targeted secret is present in the message of the HMAC (the second compression function, see Figure 2.1) and not in its key (the first compression function), which is already known. Usually, when a HMAC is attacked, that is its key that is targeted by the attackers, and to our knowledge, the recovery of a secret contained in the HMAC message has never been attempted in the literature. The HMAC with the controlled, known and unknown values is shown in Figure 5.3 below:

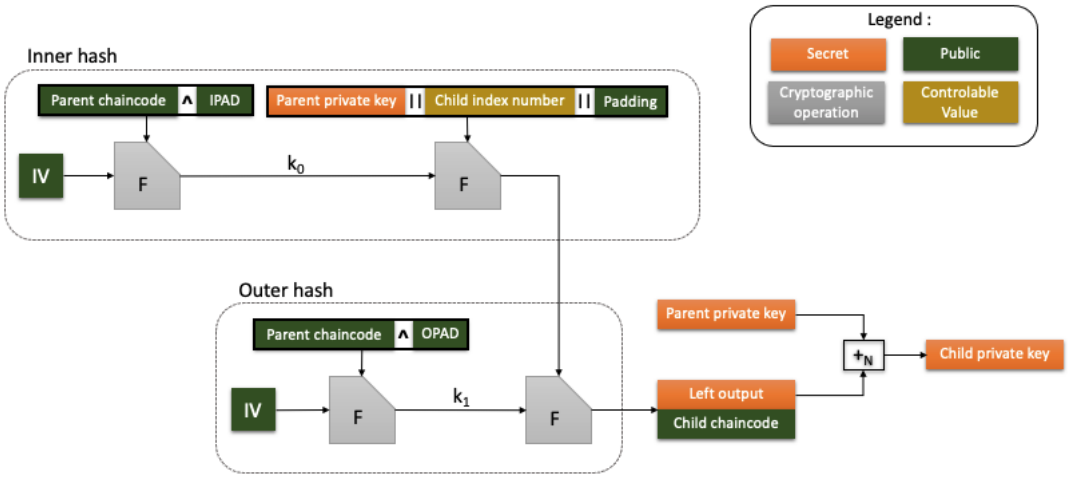

Figure 5.3: HMAC in a hardened derivation

5.5 Choice of the attack on HMAC SHA-512

In the Satochip wallet loaded on the card, the HMAC SHA-512 algorithm is implemented in software while the private to public key conversion and ECDSA are implemented in hardware and protected. The fact that an algorithm is implemented in software is more dangerous for the security of the card. The implementations in software are usually longer than hardware ones, so intermediate values are manipulated for longer and it is more interesting for a potential side-channel attacker. Furthermore, the fact of knowing the implementation prevents the attacker from some reverse engineering efforts. Accessibility and control of the implementation can also help to set up the attack. The problem can be simplified in a first step and then complicated to gradually increase the difficulty, which is not possible with a hidden hardware implementation.
So in the continuation of this work, the objective will be to attack the hardened key derivation algorithm and more precisely the HMAC SHA-512, even if it has

never been done in the same way before. In order to do that, the attack will focus on the inner hash, because that's where the controlled values and the secret values are manipulated (See Figure 5.3).

5.5.1 Setting aside of the DPA

In this part, the feasibility of a DPA will be studied. The possibilities are quite reduced given the little control that is left to the attacker. Indeed, the attacker would only have the control on the 4 bytes of the child index number in the second compression function F. Furthermore, these bytes are concatenated with the 16 bytes of the secret. In the compression function of the HMAC SHA-512 (Figure 1), the values have a length of 64 bits, and the interesting values (the secret key and the controlled values) are distributed in the first 5 rounds as follows:

- $M_1 = 0X00 \mid key\ (7\ bytes)$
- $M_2, ..., M_4 = key\ (8\ bytes\ each)$
- $M_5 = key\ (1\ byte) \mid index\ (7\ bytes) \mid padding\ (3\ bytes)$

So the only round where controlled values are used and where a DPA could be applied is the fifth, and more particularly in the 7th line of the Algorithm 1:

$$T_1 = H + CH(E, F, G) + \Sigma_1 E + M_5 + K_5 \tag{5.1}$$

It is useful to note that in this equation, as the input of the compression function is constant, all the values but M_5 are constant and unknown (except K_5 which is constant and known). This line of the algorithm is decomposed in several operations in the software because as mentioned in section 4.1, the processor has a 16-bits architecture so the variables used have a maximal length of 16 bits. And the only line of code that is interesting among them is the one that uses a part of M_5 with both a secret and a controllable value. This equation is shown in Figure 5.4, in which $S1|V1$ is a part of M_5 and more precisely, $S1$ is the targeted secret and $V1$ is the variable controlled by the attacker. With regard to $IV1|IV2$ and $UK1|UK2$, they are just other intermediate variables used in the code and they don't correspond to any particular variable of the Algorithm 1.

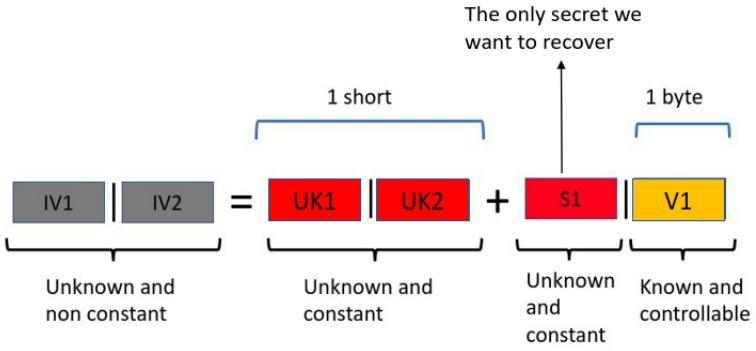

Figure 5.4: The line of the code concerned by the attack

The particularity of this equation is that the secret and the variable are concatenated in a unique short, which will complicate the task of recovering $S1$. Nevertheless, we can decompose this equation between the left and right bytes respectively. For the right part, we have:

$$IV2 = UK2 + V1 \tag{5.2}$$

This equation could allow to recover UK2 by performing a DPA, but this value is not important and could only give information about the presence of a carry for the left bytes.
And for the left bytes, we have:

$$IV1 = UK1 + S1 + carry \tag{5.3}$$

In this equation, the carry is 0 or 1, in function of the value that we give to V1. One could compute probabilities on this carry thanks to those recovered on UK2 with the equation 5.2. Then, we could have probabilities on UK1+S1 and on IV1, but not on S1 alone. That's why one can conclude that we are not dealing with a DPA and therefore a SPA must be used to obtain probabilities on S1 only.

5.5.2 Design and strategy of the attack

As a DPA is not possible to set up, one has to focus on the possibility of performing a SPA on the inner hash. The SASCAs, presented in Section 2.4.4, are a good candidate to perform this attack.
Indeed, while performing an SPA, it is crucial to extract as much information per trace as possible. This task is handled by the SASCAs by modelling the leakages of several intermediates values to compute marginal probability distributions for the

subkeys. Furthermore, the implementation of the HMAC SHA-512 being available, one can therefore construct the factor graph and set up the attack.

Firstly, only the attack against the first word of the HMAC's message (56 MSBs of the key) is developed. Then, the extension of the attack will be discussed at the end of this section. A particularity with the smart card attacked here is that it can only manipulates `shorts` (16 bits) and `bytes` (8 bits). As said before, the processor has a 16-bits architecture.
Let's illustrate the construction of the graph by setting the first operation of interest in a factor graph. This operation is $v0 = s0 + c0$ where $v0$ is an intermediate variable, $s0$ the 8 MSBs (the 8 first bits of $s0$ are zeros) of the secret key and $c0$ a constant:

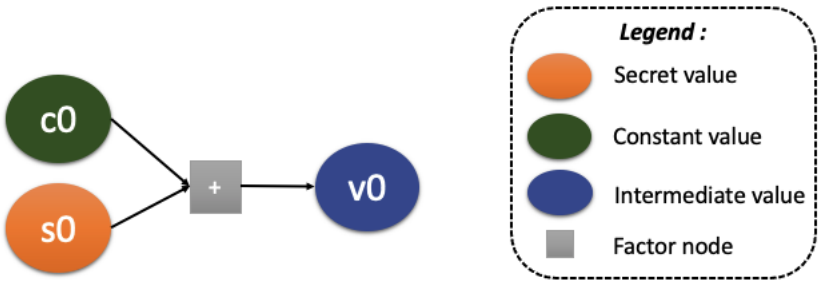

Figure 5.5: Example of a factor graph

Let's recall that in a factor graph, there are two types of nodes: factor nodes and variable nodes. Here the factor node is represented by the "+" and the variables nodes are $v0, s0$ and $c0$. Then, operating that way for all the operations involving targeted secrets, one can build the complete factor graph for the 56 MSBs of the private key manipulated in the first round of the compression function (the first byte is known see Section 5.5.1) . It is available in the Appendix in Figures A.1 and A.2.
Here is the list of the operations used for building the whole factor graph of the compression function:

- " + " : the modular addition.
- " | " : the bitwise OR.
- " $>>$ " : the shift operation.
- " $\sim$" : the bitwise inverse.
- " & " : the bitwise AND.
- " $\hat{}$" : the bitwise XOR.

By looking at these figures, one can notice that the SASCA for the first round of the compression function will therefore contain 69 variable nodes, 56 function

nodes and its first step will therefore be to build 56 leakage models. Once the graph has been built and the models are computed, the BP algorithm can be performed and one will be able to find the best subkeys candidates by looking at the marginals after convergence. In case of success, this attack would allow to find the 56 bits of the secret key knowing only 9 constants.

Extending this attack would be possible in two different ways:

- One could extend the graph through the next rounds and then build a graph with the 5 first rounds of the HMAC-SHA512. The problems with this method are the memory and computational costs which will quickly become impossible to handle by the attacker.

- The second way would exploit what is called an extend and prune strategy. This strategy is used to recover a whole secret by recovering some interdependent parts of it. When facing this configuration, one can't divide his problem in independent subproblems so the way of doing things is to progressively build a solution by solving these subproblems in a smart order.
  Let's illustrate it by an example, if an attacker is faced to an attack in which the secret must be recovered in three different parts. Recovering the third one needs the second one, and recovering the second one needs the first one. In this obvious situation the attacker must begin by recovering the first part, use it to recover second one which will allow to recover the third one. Then, the whole secret is recovered. That's what is called an extend and prune strategy.
  So back to the second way of extending the attack, the procedure would be to consider separately the rounds of the compression function. The factor graphs for the next rounds will have exactly the same structure as the one presented in Figure A.1 and A.2. The difference will be that the constant for the further rounds depend on the secrets of the previous ones. Knowing it, one can use an extend and prune strategy to recover the whole secret. The strategy will therefore be to perform a SASCA on the first round to recover the first part of the key (56 bits) and then using the result to compute the constants of the next round and performing a SASCA on it. By repeating these steps until round 5, the whole secret key could be recovered using 5 successive SASCAs.

Chapter 6

Measurement acquisition setup

In this chapter, the setup designed to measure the leakage of the device will be described. The whole setup can be seen in Figure 6.2. It is an important step for a side-channel attack because the quality of the acquired traces will be determinant for the result of the attack. The method used to launch the operations will also be explained.
The leakage is measured thanks to the current probe Tektronix CT1 and sampled by a PicoScope 5000 Series USB oscilloscope at 10 MHz with a 12 bits resolution (see Figure 6.1). The voltage range of the oscilloscope is set to +-10mV in order to maximize the precision while avoiding the saturation.

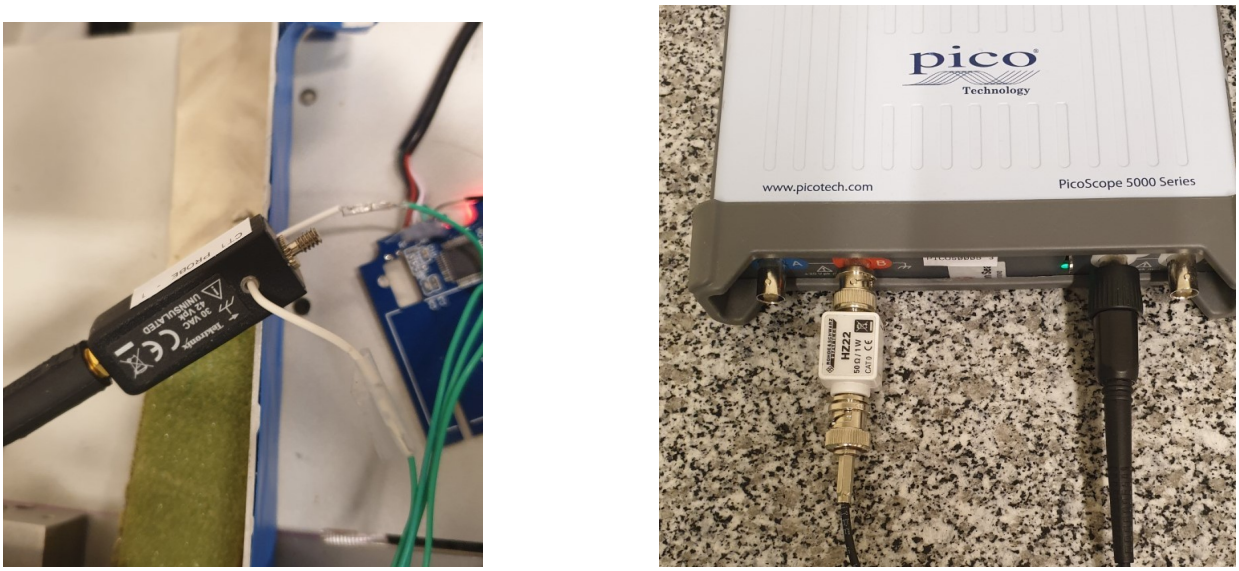

Figure 6.1: On the left: Current probe Tektronix CT1. On the right: PicoScope 5000 Series USB oscilloscope

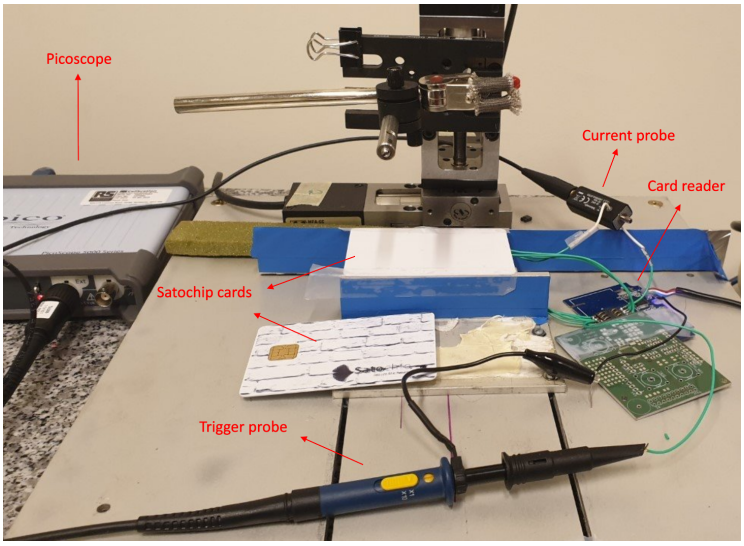

Figure 6.2: Overview of the experimental setup

During this work, we first tried to measure the leakage with an electromagnetic near field probe R&S HZ-15 and a matched preamplifier R&S HZ-16 but as it can be seen on the top of Figure 6.3, the operations were not distinguishable in the traces, there was too much noise. So, after having tried some leakage detection tools that were not conclusive, we decided to switch to the current probe Tektronix CT1 (Figure 6.1), whose measures of the leakage for a HMAC were more distinguishable

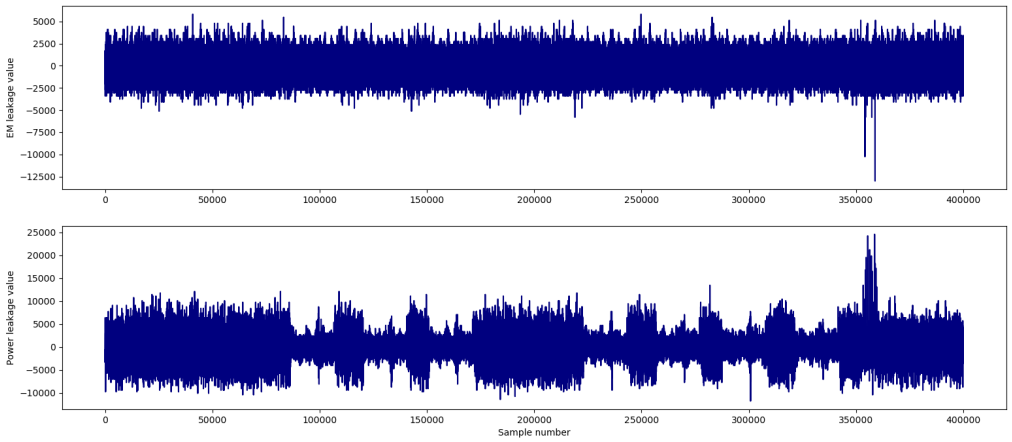

Figure 6.3: On top: trace of a full HMAC with the electromagnetic field probe. On the bottom: trace of a full HMAC with the current probe.

The triggering technique used to record the traces is made thanks to an APDU sent by the computer to the card. Although we wanted a more precise trigger to frame the interest operations to avoid a maximum of jitter, the fact that we use Java Card prevents us from doing so, as detailed in Section 4.1.5. The possibility of triggering with the same technique as Beckers et al. in [50] has been tried but it has been ruled out since it didn't work better than triggering using the APDU. The problem using the APDU for triggering is that the leakage recorded includes the process of the APDU and not only to the operation of interest. This increases the size of the recorded leakage and thus increases the risk of desynchronization of the traces due to jitter.
In practice, a probe is connected to the IO1 pin of the smart card (see Figure 4.1), and each time an operation (e.g. a HMAC) is asked to the card, the computer sends a signal in this pin and the recording of the leakage by the picoscope is started. This APDU protocol was already implemented as the default protocol used to communicate with the card so it didn't necessitate a lot of manipulation to put this system in place.
The stimulation of encryption, as well as the management of collected data, are performed thanks to a Python script [1]. Some simplifications, that will be explained in the next section, allowed us to reach about 20 traces per second.
An important point to highlight is the fact that we observed a very limited lifespan for the card. Indeed, when taking around $3 * 10^6$ traces for the simplified HMAC, the card was not functional anymore. This will be a restricting factor as detailed in the next section.

[1] The metrics have been computed thanks to the Python package SCAlib, see `https://scalib.readthedocs.io/en/latest/index.html#about-us`

Chapter 7

Experimental results

In this chapter, some modifications and simplifications that have been performed to the software for the recording of the traces are first explained. Then, an evaluation of the noise level in the traces will be developed thanks to some metrics, and the intermediate results are successively shown. After that, the implementation of a simulated SASCA will be detailed and then the results obtained for this simulation will be commented.

7.1 Modifications of the software

The first step for computing the useful metrics is the trace acquisition phase. During this phase, it is important for the attacker to reach a good rate of acquisition since some of the metrics as the SNR or the PI can require many traces before reaching convergence. In this direction, some modifications on the Satochip software had to be done. Indeed, with the provided HMAC implementation, the computation time by the chip was around 4 seconds. With the sampling frequency used (10MHz), it represents 40'000'000 samples to store each time a HMAC is computed. Firstly, it would take a lot of time for the acquisition of the traces and secondly, it would be a huge memory cost to store all those samples. Hopefully, all the operations performed in the whole HMAC process are not useful for the attack, so shorten the HMAC execution is an interesting solution. Indeed, given that the attack focus is the inner hash of the HMAC, the outer hash is not useful for the attack so it has been deleted. After that, the hash function SHA-512 has 80 rounds, but as only one round of the 5 targeted will be attacked at a time, only one round is necessary to be kept for each step of the attack. After setting up these simplifications: the number of rounds and the outer hash, the new version of the HMAC is 200'000 samples long, which is only 0.5% of the initial length. The trace acquisition can therefore be performed in good conditions without having lost any

relevant information for the attacker. It is important to note that the modifications detailed here don't have an impact on the feasibility of the attack. The attack could be done without them but it would be longer and it would require a large storage system.

7.2 Evaluation of the noise level

A first step after recording the traces is to evaluate the noise level in the traces, and thus the quantity of information they contain. In order to do that, a t-test followed by an alignment of the traces and then some SNRs and PIs will be computed. While recording traces with the setup, we faced the limitation of the smart card explained previously, which is the limited life span. After a certain number of operations performed, the card was no longer working and we had to replace it by a new one.
The problem with switching the card is that the leakages which come from different cards don't have exactly the same behaviour. We can't therefore just merge two sets of traces coming from two different devices because their respective leakage models are different. This observation was made while computing a SNR from 2 different cards and it has never converged even with 3'000'000 traces. Each time a new card was used, the calibration had to be restarted and the traces from the old card couldn't be used anymore. Thus, it would be interesting to study the question of the portability of the templates built by different cards, but this could be the subject of a future paper and will not be studied here. Moreover, the inter-device profiling portability has already been studied by Bronchain and Standaert in [51] who showed that the portability of templates between two devices induce some information loss on Cortex microcontrollers, which confirms the observation made with the cards .
With the number of traces that can be recovered with one card, we weren't able to compute the metrics for 16-bits variables (size of the words used by the card). So the choice to reduce this size has been made. This allows to limit the number of intermediate values and thus decrease the number of traces needed. Thus, the metrics that are computed in the continuation of this work will be computed on 4 bits.

7.2.1 T-test

The set of traces on which the metrics will be calculated contains 1'500'000 traces of 200'000 samples each. But before computing those metrics, it would be interesting to perform an alignment of the traces in order to take full advantage

of them. To do so, it is important to localise where the useful information is manipulated in the trace. Therefore, a t-test has been performed beforehand on the set of traces in order to localise a window in which the alignment could be performed. The result of this t-test is shown in Figure 7.1, with the mean of the traces as a reference.

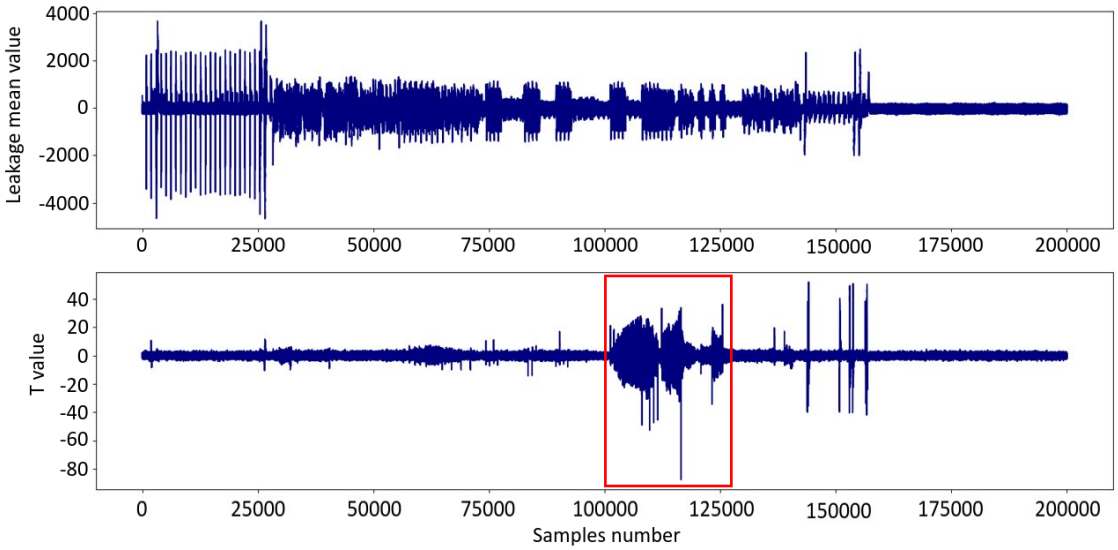

Figure 7.1: On top: Mean of the traces. On the bottom: Welch's t-test

A window of 25'000 samples is highlighted between the 105'000 and 130'000 samples and another between 140'000 and 165'000. The second one is not interesting because it corresponds to the APDU and not to the compression function, so that is on the first window (framed in red) that we will focus during the next steps.

7.2.2 Processing of the traces

Now that the window of interest has been localised, a processing of the traces can be done. It is important because despite the trigger set up, the traces are not perfectly aligned. This means that a particular time sample in one trace does not correspond exactly to the same time same sample in another trace. It can be seen in Figure 7.2 that the misalignment leads to a small amplitude of the mean (in red) compared to the reference trace (in blue).

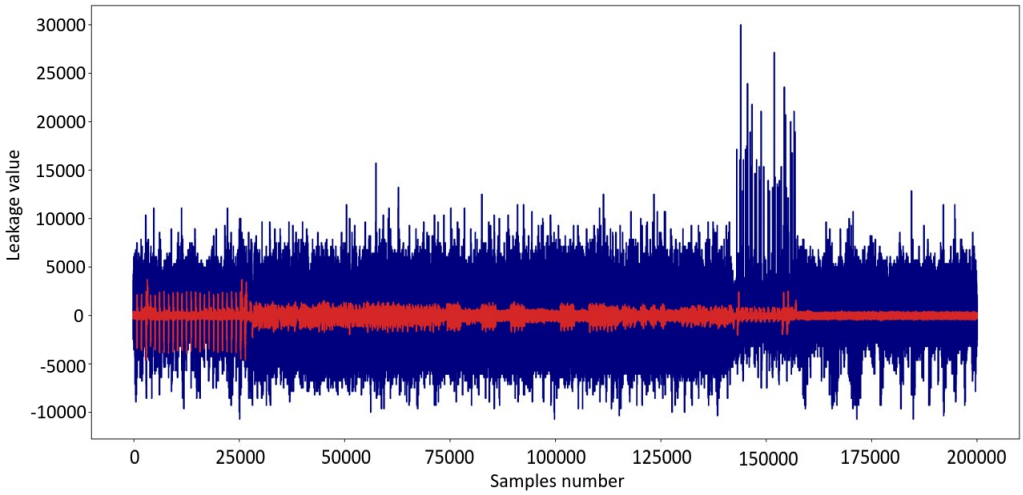

Figure 7.2: Mean of non aligned traces

In order to fix this misalignment problem, an alignment algorithm has been implemented. This algorithm is based on cross-correlation [52] and is detailed in Algorithm 5. This method has been chosen because it gives good results for a low computational cost [53]. It is a static alignment because it only computes the offset of the trace from a reference trace in a certain alignment window. Then it shifts the trace to obtain the best cross-correlation.
We can note that if a minimum correlation is not reached, the trace is discarded so we set this minimum correlation in order to save approximately 80% of the traces after alignment.

Algorithm 5 Cross-correlation alignment

Input: the model trace M, the set of N traces to be aligned T = (T1,..,TN), the alignment window AW, the correlation threshold CT, the interest window IW and the spike noise threshold NT.

Output: the aligned traces set.

for i=1 to N_{traces} **do**
  $C \leftarrow cross\_correlation(M[AW], T_i[AW])$
  if max(C) > CT && max(T_i[IW]) < NT **then**
    $Offset \leftarrow argmax(C) - len(AW)$
    $T_i \leftarrow roll(T_i, Offset)$
  else
    suppress T_i from T
  end if
end for
return T

The result of the alignment on a window of 2000 samples for a trace containing the 25000 samples highlighted with the t-test is shown in Figure 7.3. In this figure, the mean of the traces (in red) in the alignment window is much closer to the amplitude of a simple trace (in blue) than before alignment. This confirms the proper functioning and usefulness of this algorithm.

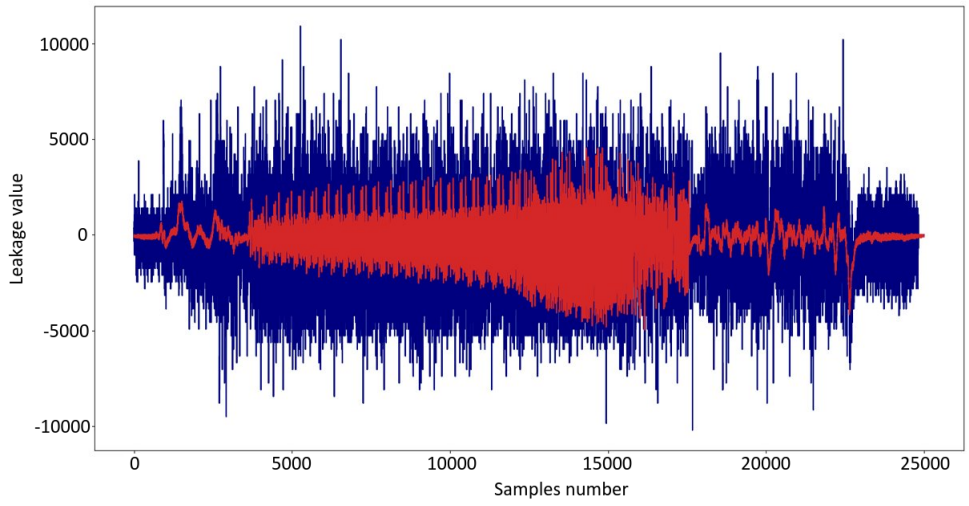

Figure 7.3: Mean of aligned traces

7.2.3 SNR

Once the traces have been aligned, the computation of the different SNRs can be done. We will perform a SNR for each different operation appearing in the compression function because the SNR values obtained could be different depending on the operation. Indeed, it would be possible that one operation leaks more than another.
Firstly, the SNRs were computed on 8 bits, but even with more than 1'500'000 traces, they didn't converge, so as it has been said at the beginning of the section, we decided to simplify the problem and decrease the number of classes for the targeted variables by taking into account only 4 bits. Then, with the same number of traces, it will provide a better accuracy for each of the classes and the SNRs will converge.
After that, the first SNR has been computed on the 4 LSBs of a modular addition operation with random inputs of the compression function of the SHA-512 with 1'500'000 traces. The result can be seen in Figure 7.4. We can see that the largest peak barely reaches a value of 0.00013 which is really small compared to the values usually obtained in the literature for this type of smart card. For a comparison, the smart card used in [54] allowed to obtain a SNR up to 0.1. This difference of 3 orders of magnitude could be explained by the quality of the set up measurement. Indeed, unlike us, they used a smart card tester that allows to communicate with the cards and to tune the carrier field. They mention that having a adjustable field strength is important when taking contact-less power measurements because lowering the field strength to a level where the smart card has just enough power to operate can greatly improve the SNR of the measurements. Moreover, the smart card tester can provide a trigger signal to obtain perfectly aligned measurements. We did not have access to all these features and our trigger was not perfect because of the limitations due to the use of Java Card. This could explain why the SNR value obtained with our device is 3 orders of magnitude lower.

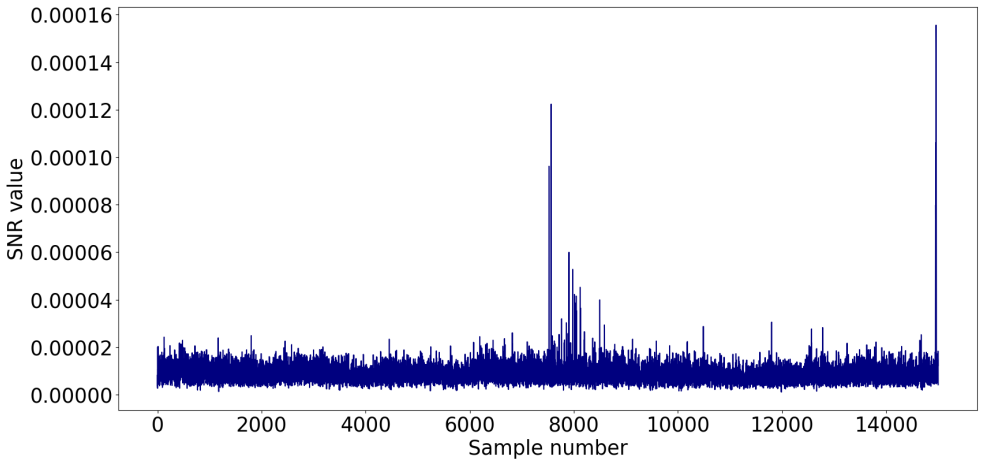

Figure 7.4: SNR value of a modular addition operation on 4 bits with random inputs of the compression function

Then, a second SNR has been performed on a bitwise AND operation with random inputs for the compression function. It can be seen in Figure 7.5 that unlike the SNR performed on a modular addition, this SNR doesn't converge with the same amount of traces. This observation has also been made by Collin in [12], who explained that it was due to the fact that unlike the modular addition, the output of the bitwise AND is not uniform. Indeed, for random inputs of a bitwise AND operation, the probability of obtaining each of the different outputs is not the same. Thus, the accuracy in the estimation of the different classes is different, which explains that more traces are needed in order to decrease the noise level and reveal the POIs using the SNR.
However, when computing a SNR, one can note that the noise level is always decreasing while the number of traces increases. It is shown in Figure 7.6 which shows the evolution of the SNR value of the modular addition's best POI. Once the peaks have decreased to a certain value, they won't go up again. They will continue to decrease or stabilize at their final value as the figure demonstrates.
Thus, despite the non-convergence of the SNR for the AND operation, one can note that the level of the different peaks has the same order of magnitude (10^{-4}) as the SNR performed on the modular addition. So we can still conclude that the final value with 4 bits won't converge to a better value than the one obtained with the modular addition, it will be at best the same value or less.

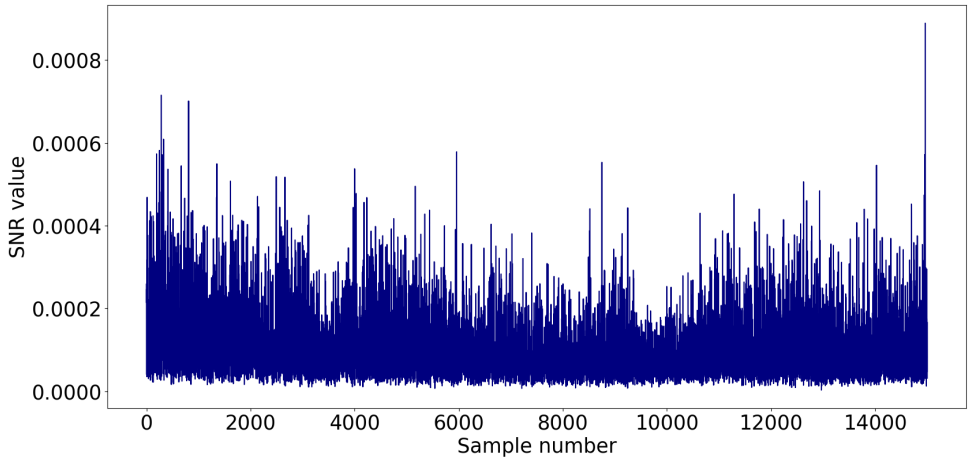

Figure 7.5: SNR value of a bitwise AND operation on 4 bits with random inputs of the compression function

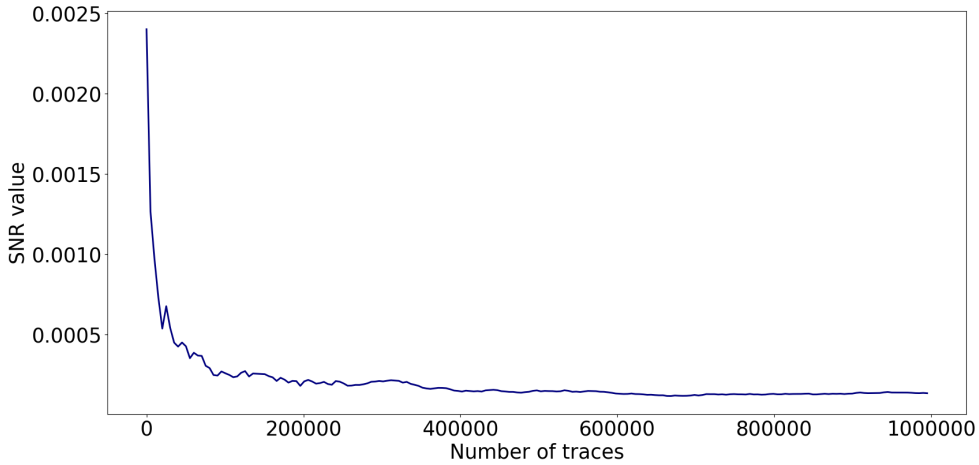

Figure 7.6: Evolution of the SNR value of a modular addition's POI in function of the number of traces

Then, the last operation that has to be observed is the bitwise OR operation. The corresponding SNR, once again computed on 4 bits with random inputs of the compression function, can be seen in Figure 7.7. This one has converged and the higher peak has a value of 0.000085, which is a bit lower than the SNR of the modular addition.

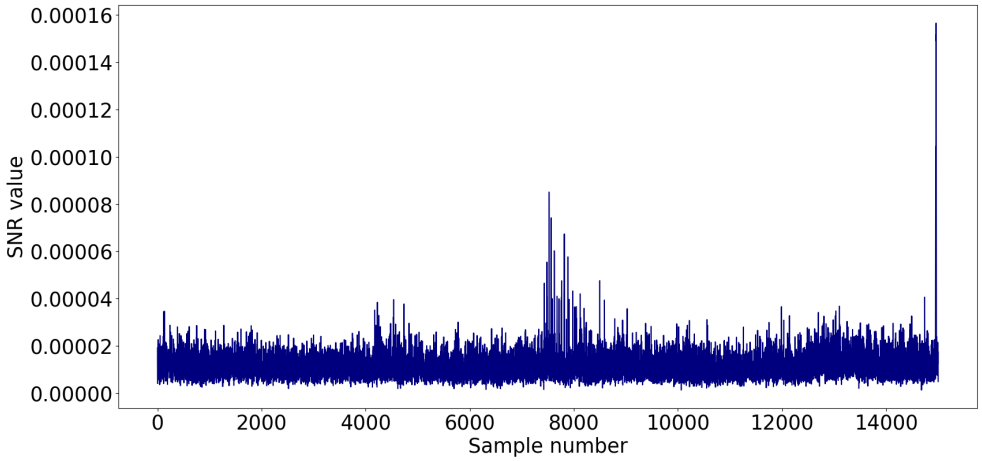

Figure 7.7: SNR value of a bitwise OR operation on 4 bits with random inputs of the compression function

Given the very low value of the different SNR peaks and the number of peaks visible on the different figures, it would be worthwhile to use a metric based on multivariate models. This will allow to see if even more information could be extracted from the traces. Since this is not possible with the SNR, the PI of the traces based on all the samples that correspond to the peaks which emerge from the noise level will be computed. And because the SNR of the bitwise OR and the modular addition are of the same order of magnitude, we decided to focus only on the modular addition by assuming that the results would be similar for the bitwise OR operation.

7.2.4 Perceived Information

They are four different ways from which the PI based on multivariate models could be computed. They depend on the samples that are taken in order to compute it. Indeed, we could even take the window containing all the samples from the first to the last peak revealed by the SNR, or make a selection of a small number of samples by only considering the samples whose SNR value is outside of the noise level. After that, no matter what choice is made, one can use a dimensionality reduction technique to reduce both the memory and timing complexity of the attack even if there is a risk to lose information by doing it. The Linear Discriminant Analysis dimensionality reduction chosen to perform this task is developed by Cagli et al. in [55].
The interest of taking all the window and not only the POIs revealed by the SNR is

to avoid the loss of information due to the jitter. Indeed, if a shift of 2 or 3 samples remains between the traces after the alignment, the POIs will be at different places in each trace. Then, for some of them, the real POI in the trace will fall next to the POI revealed by the SNR and thus the information contained in this specific trace will be lost if all the samples aren't considered for the computation of the PI.

However, even if taking all the samples of the window seems to be the best solution given the explanations previously developed, it is important to mention that the speed of convergence will be different in function of the cases. Indeed, Lerman et al. highlighted that the convergence of the PI is impacted by the number of useless samples taken for the models' computation in [56]. The models built using less useless samples lead to a faster convergence of the PI than those built with more useless samples. Thus, the faster convergence will probably be obtained for the case where a pre-selection of the samples has been made by the SNR, and using a LDA dimensionality reduction. And the slowest will be the one for which all the samples of the window are taken and no LDA has been performed. So depending on the size of the window and the number of traces available, make a pre-selection of the samples thanks to the SNR could be a good solution even if it isn't as good as taking all the window.

PI with the whole window

First, the best solution which is to compute the PI thanks to all the samples of the window has been studied. To do so, the window of 1100 samples highlighted by the SNR visible in Figure 7.10 will be used to compute the PI.

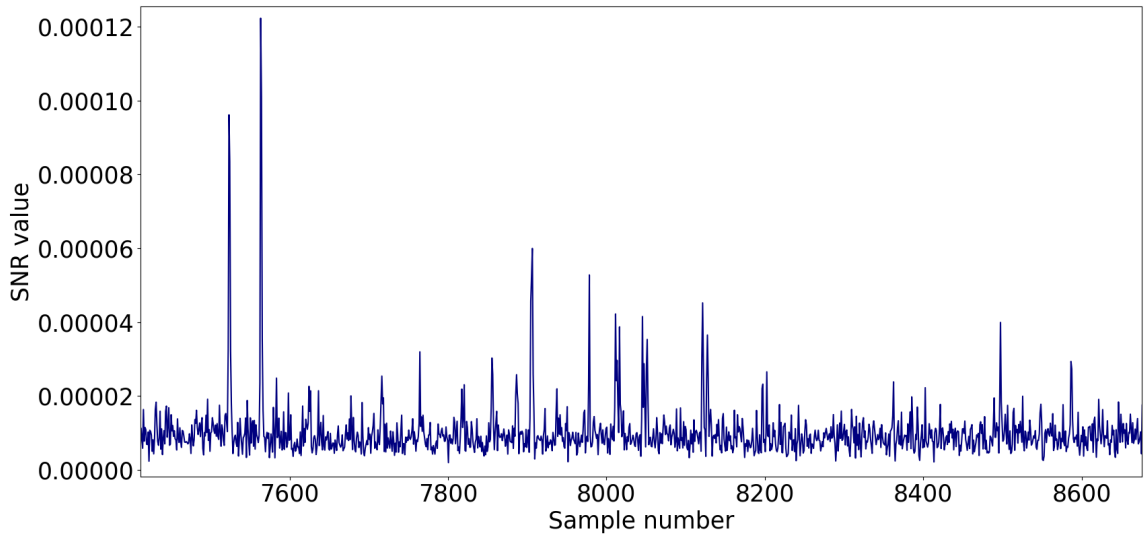

Figure 7.8: Window of samples highlighted by the SNR

When computing the PI based on multivariate models, the number of dimensions has an impact on the speed of convergence. As explained above, using more dimensions leads to a slower convergence. Then, the PI will be first computed with a LDA dimensionality reduction to 15 dimensions in order to see if it has converged. If not, the dimension will be reduced to 1.
The PI computed with LDA dimensionality reduction to 15 dimensions is visible in Figure 7.9. The value obtained for 1'600'000 traces is still negative but the value isn't stabilized yet. As said before, this case has the lowest speed convergence so it would necessitate more traces which is not possible due to the lifespan of the card.

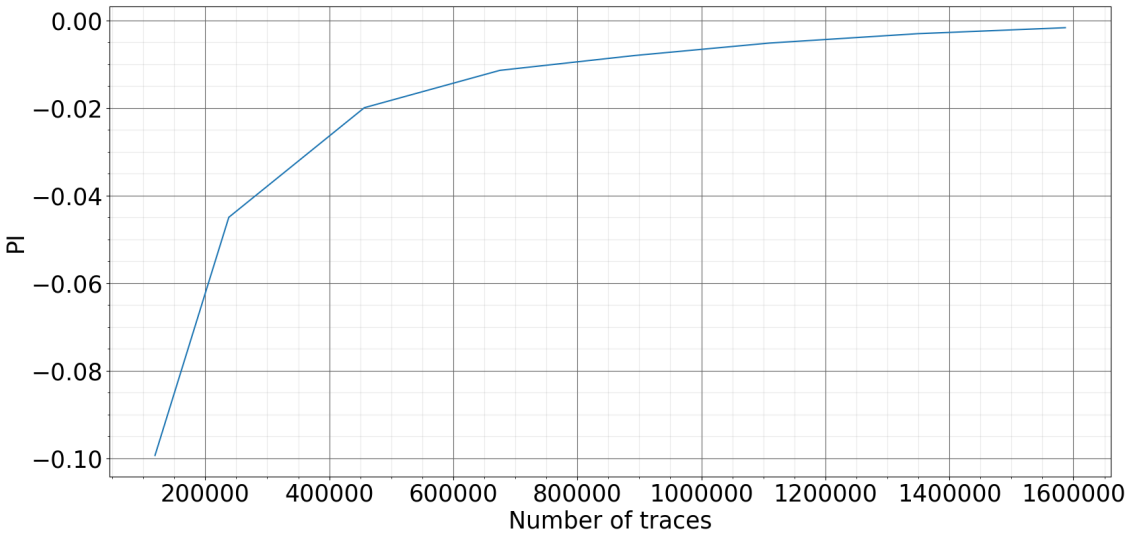

Figure 7.9: PI computed on all the samples of the window with LDA dimensionality reduction to 15 dimensions

Then, we analyzed the evolution of the PI with the same samples but with a LDA dimensionality reduction to 1 dimension. The value has not yet converged either but the PI obtained with the 1'600'000 traces is around 0.002 so it is higher than the previous one and therefore more interesting to us. However, it is not because the value with 1 dimension is higher than the one with 15 dimensions for the current number of traces that it will be true when the values will be both stabilized. Once again, with the current number of traces the 1-dimension value could be closer to its final value than the 15-dimensions value which could obtain a higher value after stabilization.

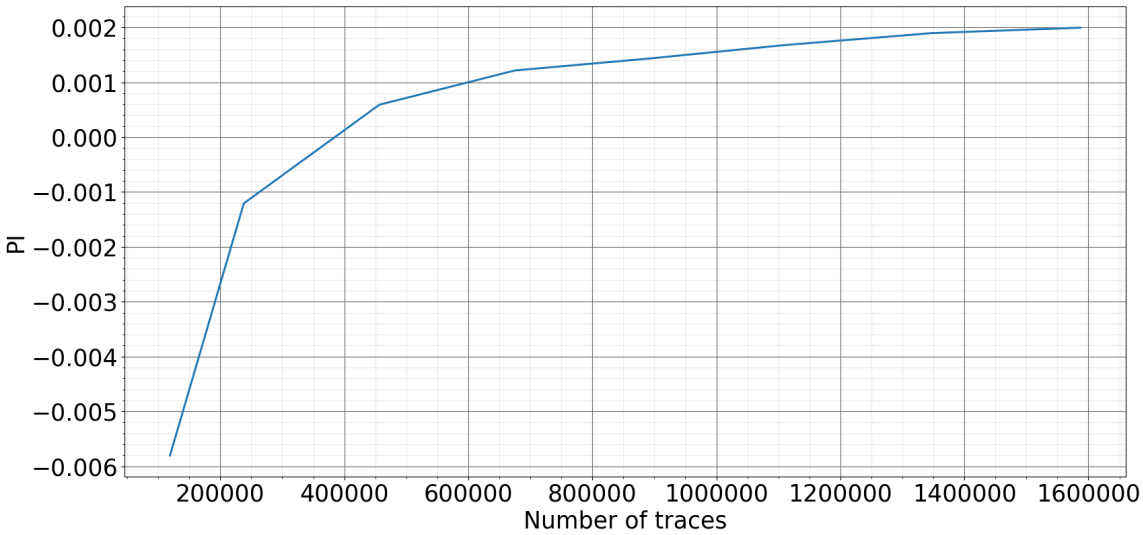

Figure 7.10: PI computed with selected samples with LDA dimensionality reduction to 1 dimension

PI with selected samples

Then, because of the non-convergence of the PI computed with all the samples of the window, we performed a computation of the PI based on 28 selected samples with the use of a LDA dimensionality reduction to 15 and then 1 dimensions. The PI has been computed thanks to the POIs revealed by the SNR and the result is visible in Figure 7.11. The red curve that uses a dimensionality reduction to 1 dimension converges faster but to a lower value, around 0.0002. The blue curve represents the PI computed with the dimensionality reduction to 15 dimensions isn't stabilized yet but has already a bigger value than the other, around 0.0007, but lower than the PI computed with the whole window of samples. Also, it won't be interesting to compute the PI without dimensionality reduction given that it wouldn't converge either with this number of traces.

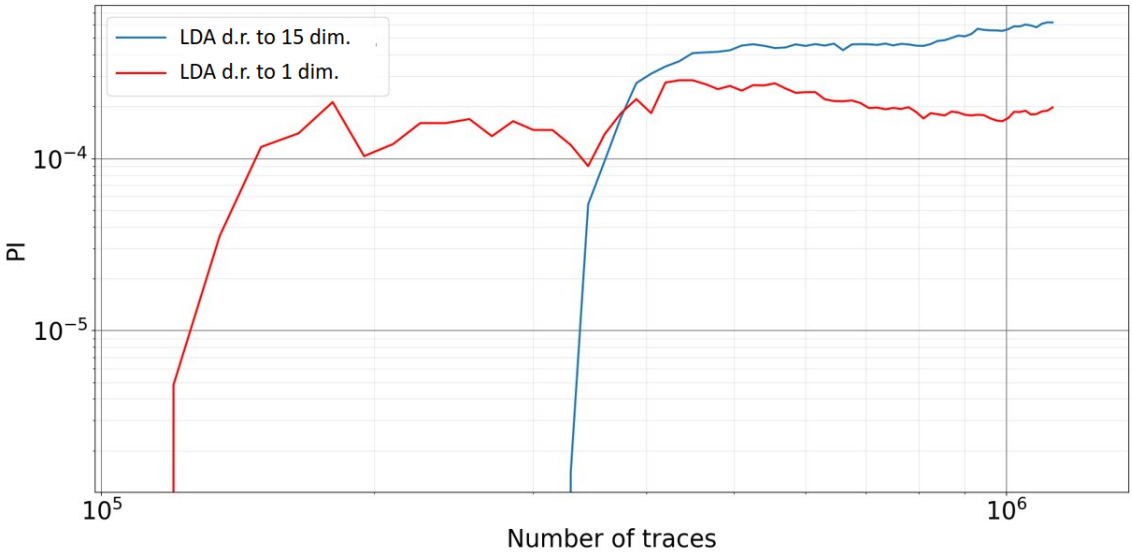

Figure 7.11: PI computed with selected samples with LDA dimensionality reduction to 1 and 15 dimensions

In this specific case, making a pre-selection of the samples thanks to the SNR and thus dropping some samples that potentially contain information can be seen as the combination of different univariate models but it is not exactly equivalent to compute a real multivariate. And in this case, averaging the traces before computing the PI could be helpful because it will have for effect to limit the impact of taking into account a sample that only contains noise due to the misalignment and thus "refocus" the information. Therefore, it would be interesting to observe the behaviour of this PI computed with LDA dimensionality reduction to 15 dimensions with averaged traces in order to observe if a higher PI value could be obtained than the one computed with the whole window of samples that reached 0.002.

Figure 7.12 shows the evolution of that PI in function of the number of traces for different amount of averaging. The level of averaging increases the PI and it reaches a value of 0.35 for 4500 traces averaged 400 times. Given the explanations previously developed, it shows that there is still a misalignment between the traces despite the alignment algorithm. The averaging has thus the effect to limit this impact and allows to obtain a better PI value.
Then, it would be interesting to evaluate the value that could be obtained with traces averaged more times. Figure 7.13 shows the evolution of the PI in function of the averaging and the value isn't stabilized for 400 averages so the PI would keep climbing if more averaging were done. However, the limitation of the number

of traces available prevents us to compute it. It is therefore difficult to predict the final value that could be obtained.

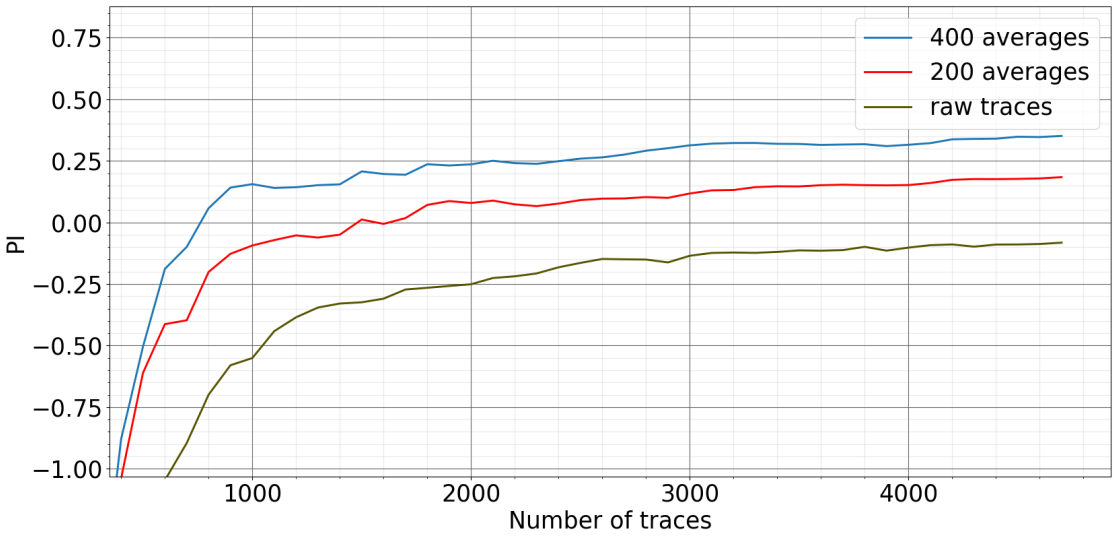

Figure 7.12: Evolution of the PI based on multivariate models in function of the number of traces, for different levels of averaging

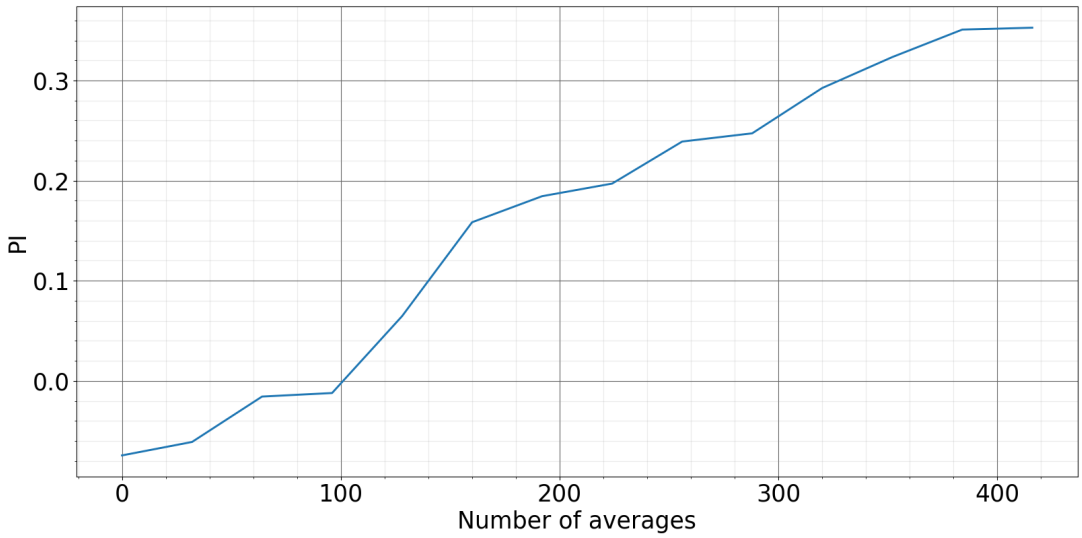

Figure 7.13: Evolution of the PI based on multivariate models in function of the level of averaging

7.3 Simplified attack

The implementation of the real attack has been seen in Section 2.4.4. But due to different limitations reminded below, this attack would be difficult to perform so we made some simplifications. The goal of this section is therefore to present the simplified version of this attack which has been performed during this thesis.

Given the limitations of the number of traces available and the low values obtained for the PI, it complicates the implementation of an attack with real traces. Knowing that, a first simplification has been made, by studying the attack on simulated traces.
Then, it has been seen that the complete SASCA graph contains 56 intermediate values on 16 bits. For building good models with noisy traces, a big number of traces is necessary for each class. And as explained before, the distribution of the traces is not uniform in the classes for some intermediate values because of the bitwise AND operation. Even if working with simulated traces allows us to use more traces, there are some limitations because of the computational complexity of using a big number of traces.

Thus, the second simplification of the attack will consider that the intermediates values are on 8 bits in order to lighten the number of traces needed. All the values

considered from here will therefore be the 8 LSBs for all the values in the graphs. This implies the shift operations in the graph available in Figures A.1 and A.2 are no longer studied as 15 bits shifts but as 7 bits shifts. It also implies that the attack will try to recover 32 bits of the key at each round and no more 64.

Besides, working with simulations will allow to study the impact of the PI on the success rate of the attack and the average key rank. For this simulated attack, it is assumed that the leakage models used in the simulations is the same for all the intermediate values. This choice has been made in order to simplify the implementation.
Then, the following steps are performed to implement the SASCA:

1. Generation of the traces according to a leakage model and given a noise level.
2. Construction of the models for the intermediate values.
3. Estimating the PI using these models.
4. Construction of the SASCA graph as explained in section 5.5.2.
5. Running the BP algorithm on it.

The strategy used to conclude the possibility of success of this attack in the real case is the following: first, simulated traces containing different noise level are generated. Then, the success rate and the average key rank of the SASCA on these different sets of traces will be observed in function of the noise level, which will be evaluated thanks to the PI. After that, one can fix a value for which the average key rank would be considered as low enough to perform the whole attack according to the divide conquer strategy. Once the result of SASCA will exceed this average key rank level, it will mean that the noise is too important. This level will therefore be compared to the one contained in the real traces to evaluate if it would be possible to reach the corresponding PI value with the real traces.
As Satochip aims to be supported by different devices, three different leakage models will be studied. As a weak PI was observed on the smart card under study, studying other leakage models could help to know if using other devices could lead to a successful attack against Satochip. Indeed, other devices could have different leakage models than the one used here, that's why it is interesting to compare the success rate and the average key rank of the SASCA with other leakage models.

7.4 Results of the attack

This section presents the results of the attack described above and an analysis of its possibility of success using three different leakage models. All the attacks have

been built by progressively adding secret parts to the graph and thus increasing its size. Four curves are therefore presented for each model, representing the success rates (SRs) and the average key rank for targeting simultaneously 1,2,3 and 4 bytes of the subkey. These four attacked (sub)graphs are circled in Figure A.3. Each point of the presented curves has been computed with 500,300,200 and 200 SASCAs for respectively recovering 1,2,3 and 4 bytes of the secret.
Then, one would want to evaluate the capacity of success for the entire 5 rounds simplified attack. As a reminder, in this attack, the 4 first rounds have the same structure and manipulate 32 bits of the key while the last round just manipulates 8 bits of it. The evaluation of success of the attack will be based on the rank estimation of the keys. Assuming that the attacker has the computing ability to exhaustively try 2^{32} enumerated keys. The product of the rank estimations for the five rounds should therefore be maximum 2^{32} (4 times recovering 32 bits and 1 time 8 bits). The minimal value for the PI which gives this result for the round will therefore be considered as the threshold for a successful attack.

7.4.1 Measured leakage model

With this model, the leakages have been built following the model of the POI that maximizes the SNR of the modular addition (Figure 7.4). The mean of the corresponding leakage has been computed for each class of the targeted intermediate value. And then Gaussian models have been computed for each of the classes, with the corresponding mean and with the standard deviation depending on the noise level. As an illustration, 8 models of leakages for 8 different classes of an intermediate value are represented in the Figure B.1. Thanks to those models, the simulated traces have been created and the SASCA has been performed.
Figure 7.14 represents the SR of that SASCA depending on the PI.

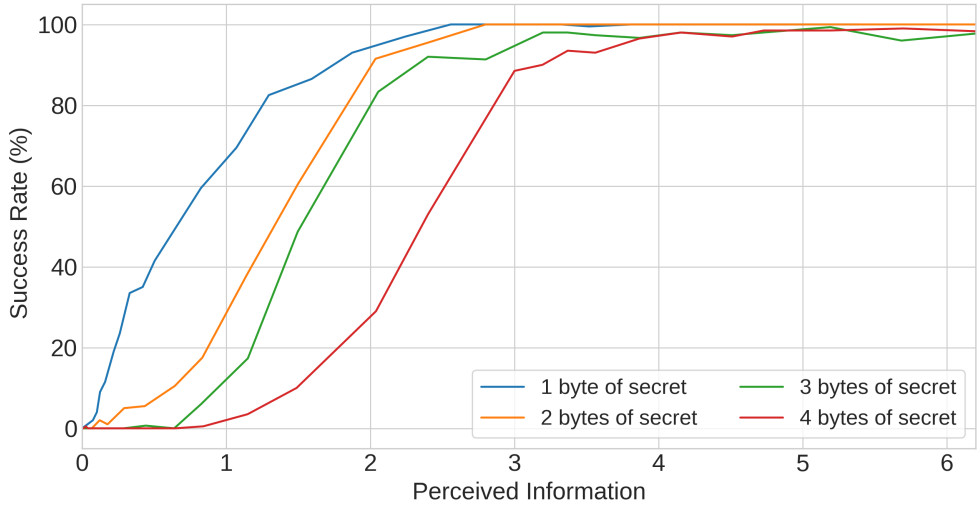

Figure 7.14: SR of the SASCA in function of the PI, measured leakage model

One can notice that, as expected, the success rate of the attack on a round increases significantly with the PI. Furthermore, it can be seen that the fact of attacking more bytes decreases the success rate significantly but with a high PI, the attack always converges towards a 100% success rate.

Then, in order to evaluate the entire attack, the rank estimation of the secret has been computed depending on the PI. The result is available in Figure 7.15.

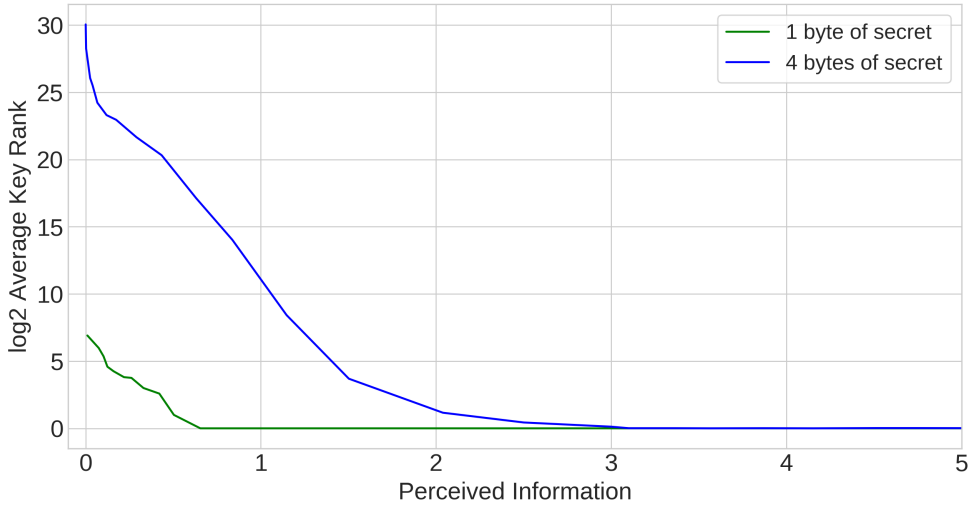

Figure 7.15: Rank estimation of the key in function of the PI, measured leakage model

Following the reasoning presented in the introduction of this section, a maximal value of 2^8 for the blue curve, which represents the 4 first rounds, and 2^0 for the green curve, which represents the 5^{th}, will be considered as a threshold for the success of the attack using a key enumeration. These values correspond to a PI value of 1.25 for this leakage model.
As a reminder, the maximum PI obtained with the real traces is around 0.35. A 350% augmentation of the PI would therefore be necessary to reach a sufficiently small rank. This wouldn't be possible with the limitations of the card and with the experimental setup. Going further with the real attack on the card is therefore not an option anymore in the framework of this thesis. However, this PI value is usually obtained with other devices such as microcontrollers as observed in [51]. The attack could therefore succeed on other devices than Java card smart cards.

7.4.2 Linear leakage model

This model consists in assigning a different weight at each 1-bit value of the intermediate value. The value of the leakage for a given intermediate value will therefore be the sum of these weights, added with the noise and then multiplied by a constant. As an illustration, 8 models of leakages for 8 different classes of an intermediate value are represented in Figure B.2. Thanks to those models, the simulated traces have been created and the SASCA has been performed.
Figure 7.16 below represents the success rate of the SASCA depending on the PI.

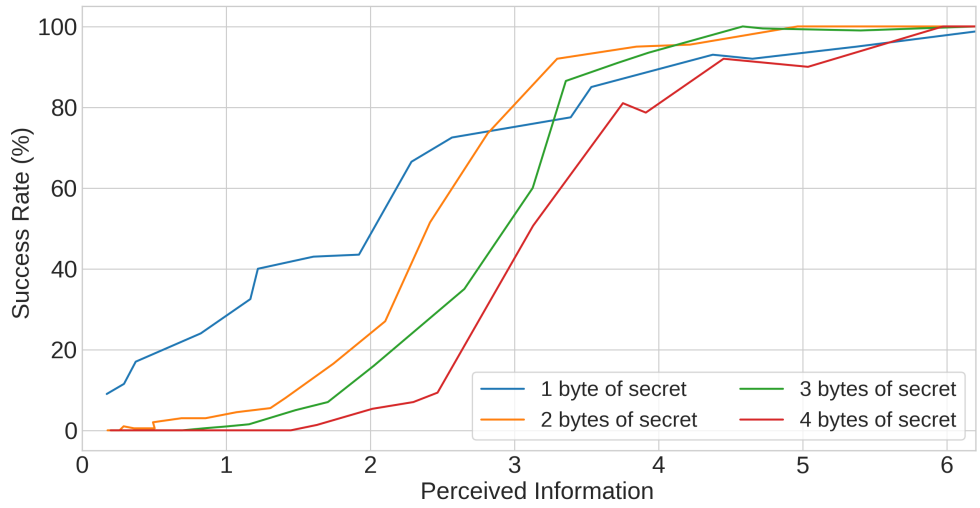

Figure 7.16: SR of the SASCA in function of the PI, linear leakage model

As observed for the measured leakage model, the success rate of the attack increases significantly with the PI but converges slower to 100% when increasing the number of attacked bytes. However, one can notice that with this model, targeting one byte of the secret doesn't converge as fast as it was expected to 100%. Indeed, that is this part of the secret which is the last to reach a maximum SR.
Then, in order to evaluate the whole attack, the rank estimation of the secret has been computed depending on the PI. The result is available in Figure 7.17 below:

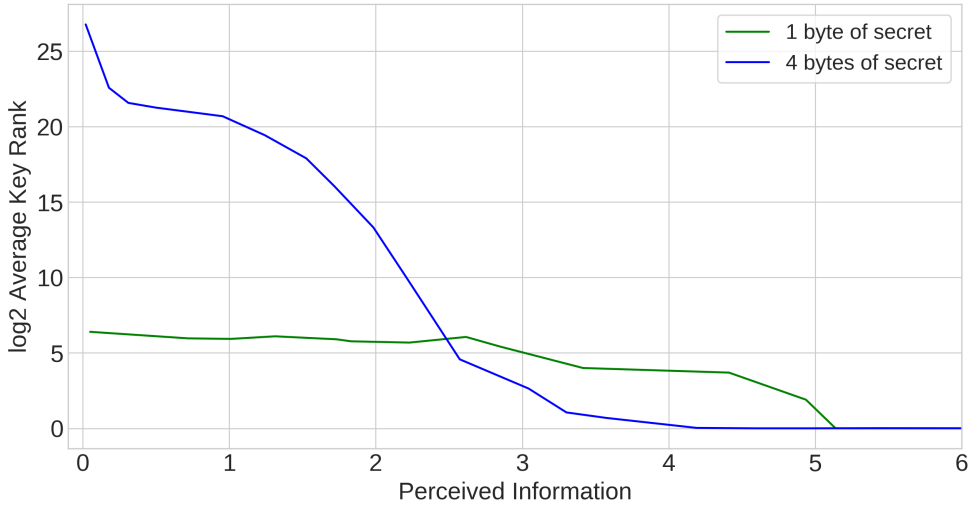

Figure 7.17: Rank estimation of the key in function of the PI, measured leakage model

Again, following the reasoning presented in the introduction, a maximal value of 2^6 for the blue curve and 2^6 for the green curve will be considered as the threshold for the success of the attack using key enumeration. These values correspond to a PI value of 2.5 for this leakage model.

7.4.3 Hamming Weight Leakage model

This last model consists in assigning a weight 1 at each 1-bit value of the intermediate value and 0 for the 0-bit values. The particularity of this model is that there are thus not 256 different means for the normal models for a 8-bits intermediate value, but only 8. The value of the leakage will therefore be the hamming weight of the intermediate value, added with some noise and then multiplied by a constant. As an illustration, 8 models of leakages for 8 different classes of an intermediate value are represented in Figure B.3. Figure 7.18 below represents the success rate of the SASCA depending on the PI.

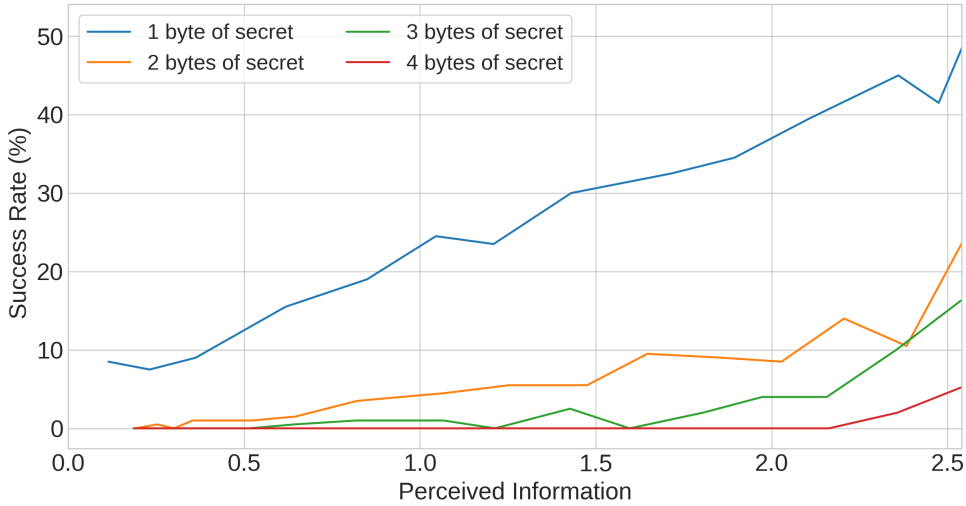

Figure 7.18: SR of the SASCA in function of the PI, HW leakage model

Once again, the success rate of the attack increases with the PI. However, the success rate doesn't converge toward 100% anymore since for the hamming weight model, the maximal PI is 2.54. Furthermore, the success decreases significantly when increasing the number of bytes, so much that the success rate of the SASCA targeting 4 bytes of secret is under 10% even with the maximal PI.

Then, in order to evaluate the entire attack, the rank estimation of the secret has been computed depending on the PI. The result is available in Figure 7.19 below:

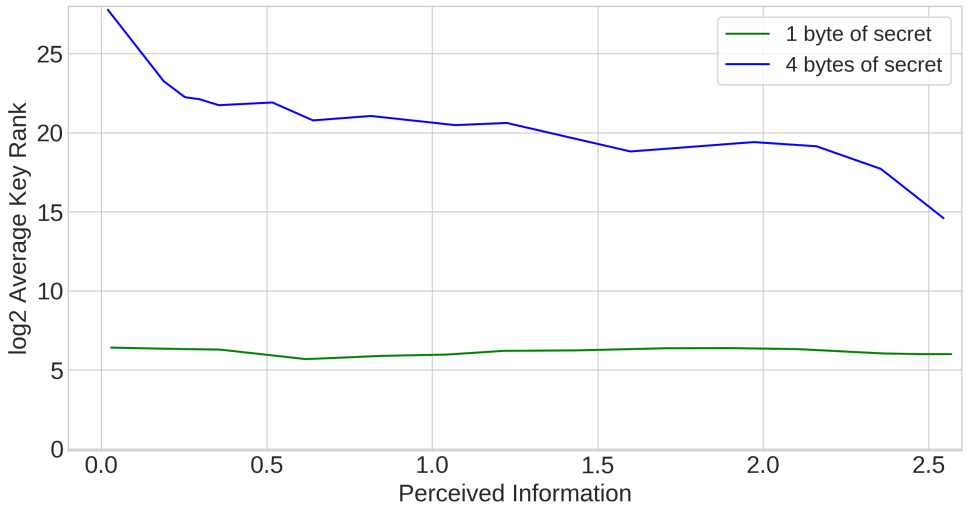

Figure 7.19: Rank estimation of the key in function of the PI, HW leakage model

With the same reasoning as the one used with the 2 previous models, the rank estimation never reaches values that could allow the attacker to perform a key enumeration. Thus, this attack performed on a device following a hamming weight leakage model wouldn't be possible.

7.4.4 Comparison of the leakages models

Now that we have the results of the three leakage models, we can study the differences between them and conclude which one would be the easiest to attack. The difference between the three leakage models is obvious when looking at the PI level needed to mount the attack. Indeed, the attack is not possible for the HW model, while the linear leakage model needs a PI of 2.5 to be considered as successful. Then, a PI of 1.25 is needed for the measured leakage model. The measured leakage model is therefore the easier to attack and by looking at the SR for one round, we can notice that these curves are coherent with this ranking too. The measured leakage model reaches a 100% SR for a PI value of 5, the linear model for a PI value of 6 while the HW model never reaches this SR.
Even while looking at a low PI values (e.g. 1 or less), this ranking is still respected. We can therefore conclude that the measured leakage model always provides better results, followed by the linear one while the HW one is the worst.

Chapter 8

Conclusion

This master thesis aimed to mount a side channel attack against the Satochip hardware wallet. The first step was to identify the different attack vectors and identify which was the weakest, in order to design an attack against it. The hardened key derivation algorithm based on a HMAC SHA-512 has been chosen. In this application, it has the particularity that its key is public and its message contains a secret private key that we want to recover. An attack by DPA has been ruled out so the attack which has finally been designed against this HMAC relies on an extend and prune strategy containing 5 successive SASCAs in order to recover the whole private key.
Then, thanks to simulations of a simplified attack, it has been shown that attacking the wallet is possible and that it could succeed. Indeed, the results of the simulations demonstrated that with a PI value of 1.25, a key enumeration would success in recovering the secret key. This is typically the kind of values observed on embedded microcontrollers, so the attack would potentially succeed on this kind of devices.
Unfortunately, we faced some difficulties with the device we were working on. First, we didn't have a lot of information on the hardware and this obscurity was problematic, unlike Udvarhelyi observed in [21], especially when setting up a trigger. Then, we had a limitation of the number of traces due to the lifespan of the card and there was jitter in the measures. Those difficulties slowed us down in our work and moreover they led to noisy measurements. Therefore, the PI value observed wasn't higher than 0.35. Thus, an augmentation of 350% of this value should be necessary to try the attack on the real traces.

It is therefore difficult to conclude on the security without having the ability to perform the attack and we can't draw serious claims on the security of the wallet at this step. We are far from the evaluation of worst-case security as presented by Azouaoui et al. in [57]. However, with more time and more resources, it would be interesting to see to what extent an attacker could get better measurements and maybe reach a better PI in order to mount this attack with real traces. Proceeding this way would allow a better assessment of the security of the wallet.

Bibliography

[1] M. San Pedro, V. Servant, and C. Guillemet, "Side-channel assessment of open source hardware wallets.," *IACR Cryptol. Eprint Arch.*, vol. 2019, p. 401, 2019.

[2] K. Lemke, K. Schramm, and C. Paar, "Dpa on n-bit sized boolean and arithmetic operations and its application to idea, rc6, and the hmac-construction," in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2004, pp. 205–219.

[3] K. Okeya, "Side channel attacks against hmacs based on block-cipher based hash functions," in *Australasian Conference on Information Security and Privacy*, Springer, 2006, pp. 432–443.

[4] P. Gauravaram and K. Okeya, "An update on the side channel cryptanalysis of macs based on cryptographic hash functions," in *International Conference on Cryptology in India*, Springer, 2007, pp. 393–403.

[5] ——, "Side channel analysis of some hash based macs: A response to sha-3 requirements," in *International Conference on Information and Communications Security*, Springer, 2008, pp. 111–127.

[6] R. McEvoy, M. Tunstall, C. C. Murphy, and W. P. Marnane, "Differential power analysis of hmac based on sha-2, and countermeasures," in *International Workshop on Information Security Applications*, Springer, 2007, pp. 317–332.

[7] P.-A. Fouque, G. Leurent, D. Réal, and F. Valette, "Practical electromagnetic template attack on hmac," in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2009, pp. 66–80.

[8] F. Zhang and Z. J. Shi, "Differential and correlation power analysis attacks on hmac-whirlpool," in *2011 Eighth International Conference on Information Technology: New Generations*, IEEE, 2011, pp. 359–365.

[9] M. Zohner, M. Kasper, M. Stöttinger, and S. A. Huss, "Side channel analysis of the sha-3 finalists," in *2012 Design, Automation & Test in Europe Conference & Exhibition*, IEEE, 2012, pp. 1012–1017.

[10] G. Bertoni, J. Daemen, N. Debande, T.-H. Le, M. Peeters, and G. Van Assche, "Power analysis of hardware implementations protected with secret sharing," in *2012 45th Annual IEEE/ACM International Symposium on Microarchitecture Workshops*, IEEE, 2012, pp. 9–16.

[11] S. Belaid, L. Bettale, E. Dottax, L. Genelle, and F. Rondepierre, "Differential power analysis of hmac sha-2 in the hamming weight model," in *2013 International Conference on Security and Cryptography (SECRYPT)*, IEEE, 2013, pp. 1–12.

[12] S. Collin and F.-X. Standaert, "Side channel attacks against the solo key-hmac-sha256 scheme,"

[13] P. Rogaway and T. Shrimpton, "Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance," in *International workshop on fast software encryption*, Springer, 2004, pp. 371–388.

[14] N. Sklavos, *Book review: Stallings, w. cryptography and network security: Principles and practice: Upper saddle river, nj: Prentice hall, 2013, 752p.* 2014.

[15] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2020.

[16] A. Corbellini, "Elliptic curve cryptography: A gentle introduction," *May*, vol. 17, pp. 1–20, 2015.

[17] H. Cohen, G. Frey, R. Avanzi, C. Doche, T. Lange, K. Nguyen, and F. Vercauteren, *Handbook of elliptic and hyperelliptic curve cryptography*. CRC press, 2005.

[18] R. Poussier, F.-X. Standaert, and V. Grosso, "Simple key enumeration (and rank estimation) using histograms: An integrated approach," in *International Conference on Cryptographic Hardware and Embedded Systems*, Springer, 2016, pp. 61–81.

[19] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.

[20] D. Johnson, A. Menezes, and S. Vanstone, "The elliptic curve digital signature algorithm (ecdsa)," *International journal of information security*, vol. 1, no. 1, pp. 36–63, 2001.

[21] B. Udvarhelyi, A. van Wassenhove, O. Bronchain, and F.-X. Standaert, "On the security of off-the-shelf microcontrollers: Hardware is not enough," in *International Conference on Smart Card Research and Advanced Applications*, Springer, 2020, pp. 103–118.

[22] B. L. Welch, "The generalization of student's problem when several different population variances are involved," *Biometrika*, vol. 34, no. 1-2, pp. 28–35, 1947.

[23] T. Schneider and A. Moradi, "Leakage assessment methodology," *Journal of Cryptographic Engineering*, vol. 6, no. 2, pp. 85–99, 2016.

[24] O. Bronchain, J. M. Hendrickx, C. Massart, A. Olshevsky, and F.-X. Standaert, "Leakage certification revisited: Bounding model errors in side-channel security evaluations," in *Annual International Cryptology Conference*, Springer, 2019, pp. 713–737.

[25] R. B. ARELLANO-VALLE, J. E. CONTRERAS-REYES, and M. G. Genton, "Shannon entropy and mutual information for multivariate skew-elliptical distributions," *Scandinavian Journal of Statistics*, vol. 40, no. 1, pp. 42–62, 2013.

[26] C. E. Shannon, "A mathematical theory of communication," *The Bell system technical journal*, vol. 27, no. 3, pp. 379–423, 1948.

[27] F.-X. Standaert, "Introduction to side-channel attacks," in *International Conference on Smart Card Research and Advanced Applications*, Springer, 2020, pp. 103–118.

[28] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual international cryptology conference*, Springer, 1999, pp. 388–397.

[29] F. Koeune and F.-X. Standaert, "A tutorial on physical security and side-channel attacks," *Foundations of Security Analysis and Design III*, pp. 78–108, 2005.

[30] Q. Guo, V. Grosso, F.-X. Standaert, and O. Bronchain, "Modeling soft analytical side-channel attacks from a coding theory viewpoint," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 209–238, 2020.

[31] N. Veyrat-Charvillon, B. Gérard, and F.-X. Standaert, "Soft analytical side-channel attacks," in *International Conference on the Theory and Application of Cryptology and Information Security*, Springer, 2014, pp. 282–296.

[32] R. Poussier, "Key enumeration, rank estimation and horizontal side-channel attacks," Ph.D. dissertation, PhD thesis, ICTEAM Institute, 2017.

[33] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Manubot, Tech. Rep., 2019.

[34] U. Gallersdörfer, L. Klaaßen, and C. Stoll, "Energy consumption of cryptocurrencies beyond bitcoin," *Joule*, vol. 4, no. 9, pp. 1843–1846, 2020.

[35] Toporin. (). "Satochipapplet," [Online]. Available: https://github.com/Toporin/SatochipApplet/tree/c14be7ab3e051d8be88666fb603f9a6f020b8565. (accessed: 27/4/2021).

[36] *Secure high-performance dual interface smart card controller*, SmartMX2 family P60D080 and P60D144, Rev. 1, NXP Semiconductors, 2010. [Online]. Available: https://elcodis.com/parts/6030143/P60D080.html#datasheet.

[37] *Iso7816 uart product specification*, Systemyde International Corporations, 2015. [Online]. Available: http://www.systemyde.com/pdf/ISO7816.pdf.

[38] Z. Chen, *Java card technology for smart cards: architecture and programmer's guide*. Addison-Wesley Professional, 2000, pp. 49–55.

[39] S. Mangard, E. Oswald, and T. Popp, *Power analysis attacks: Revealing the secrets of smart cards*. Springer Science & Business Media, 2008, vol. 31.

[40] E. Mateos Santillan, "Side channel analysis of a java-based contactless smart card," 2012.

[41] P. Wuille. (). "Bip32, hierarchical deterministic wallets," [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0032.mediawiki. (accessed: 12/4/2021).

[42] M. Palatinus and P. Rusnak. (). "Bip44, multi-account hierarchy for deterministic wallets," [Online]. Available: https://github.com/bitcoin/bips/blob/master/bip-0044.mediawiki. (accessed: 12/4/2021).

[43] J. H. Silverman and J. T. Tate, *Rational points on elliptic curves*. Springer, 1992, vol. 9.

[44] H. Le Bouder, T. Barry, D. Couroussé, J.-L. Lanet, and R. Lashermes, "A template attack against verify pin algorithms," in *SECRYPT 2016*, 2016, pp. 231–238.

[45] W. Wang and S. Fan, "Attacking openssl ecdsa with a small amount of side-channel information," *Science China Information Sciences*, vol. 61, no. 3, pp. 1–14, 2018.

[46] G. De Micheli, R. Piau, and C. Pierrot, "A tale of three signatures: Practical attack of ecdsa with wnaf," in *International Conference on Cryptology in Africa*, Springer, 2020, pp. 361–381.

[47] T. Roche, V. Lomné, C. Mutschler, and L. Imbert, "A side journey to titan," in *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021, pp. 231–248.

[48] T. Roche, L. Imbert, and V. Lomné, "Side-channel attacks on blinded scalar multiplications revisited," in *CARDIS: Smart Card Research and Advanced Applications*, 2019.

[49] P.-A. Fouque and F. Valette, "The doubling attack–why upwards is better than downwards," in *International Workshop on Cryptographic Hardware and Embedded Systems*, Springer, 2003, pp. 269–280.

[50] A. Beckers, J. Balasch, B. Gierlichs, and I. Verbauwhede, "Design and implementation of a waveform-matching based triggering system," in *International Workshop on Constructive Side-Channel Analysis and Secure Design*, Springer, 2016, pp. 184–198.

[51] O. Bronchain and F.-X. Standaert, "Breaking masked implementations with many shares on 32-bit software platforms," *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 202–234, 2021.

[52] D. W. Boyd, *Systems analysis and modeling: a macro-to-micro approach with multidisciplinary applications*. Elsevier, 2000.

[53] A. L. Rockwood, D. K. Crockett, J. R. Oliphant, and K. S. Elenitoba-Johnson, "Sequence alignment by cross-correlation," *Journal of biomolecular techniques: JBT*, vol. 16, no. 4, p. 453, 2005.

[54] A. Beckers, B. Gierlichs, J. Balasch, and I. Verbauwhede, "Comparison of two setups for contactless power measurements for side-channel analysis," in *2018 IEEE International Symposium on Electromagnetic Compatibility and 2018 IEEE Asia-Pacific Symposium on Electromagnetic Compatibility (EMC/APEMC)*, IEEE, 2018, pp. 739–744.

[55] E. Cagli, C. Dumas, and E. Prouff, "Enhancing dimensionality reduction methods for side-channel attacks," in *International Conference on Smart Card Research and Advanced Applications*, Springer, 2015, pp. 15–33.

[56] L. Lerman, R. Poussier, O. Markowitch, and F.-X. Standaert, "Template attacks versus machine learning revisited and the curse of dimensionality in side-channel analysis: Extended version," *Journal of Cryptographic Engineering*, vol. 8, no. 4, pp. 301–313, 2018.

[57] M. Azouaoui, F. Durvaux, R. Poussier, F.-X. Standaert, K. Papagiannopoulos, and V. Verneuil, "On the worst-case side-channel security of ecc point randomization in embedded devices," in *International Conference on Cryptology in India*, Springer, 2020, pp. 205–227.

Appendix A

SASCA graphs

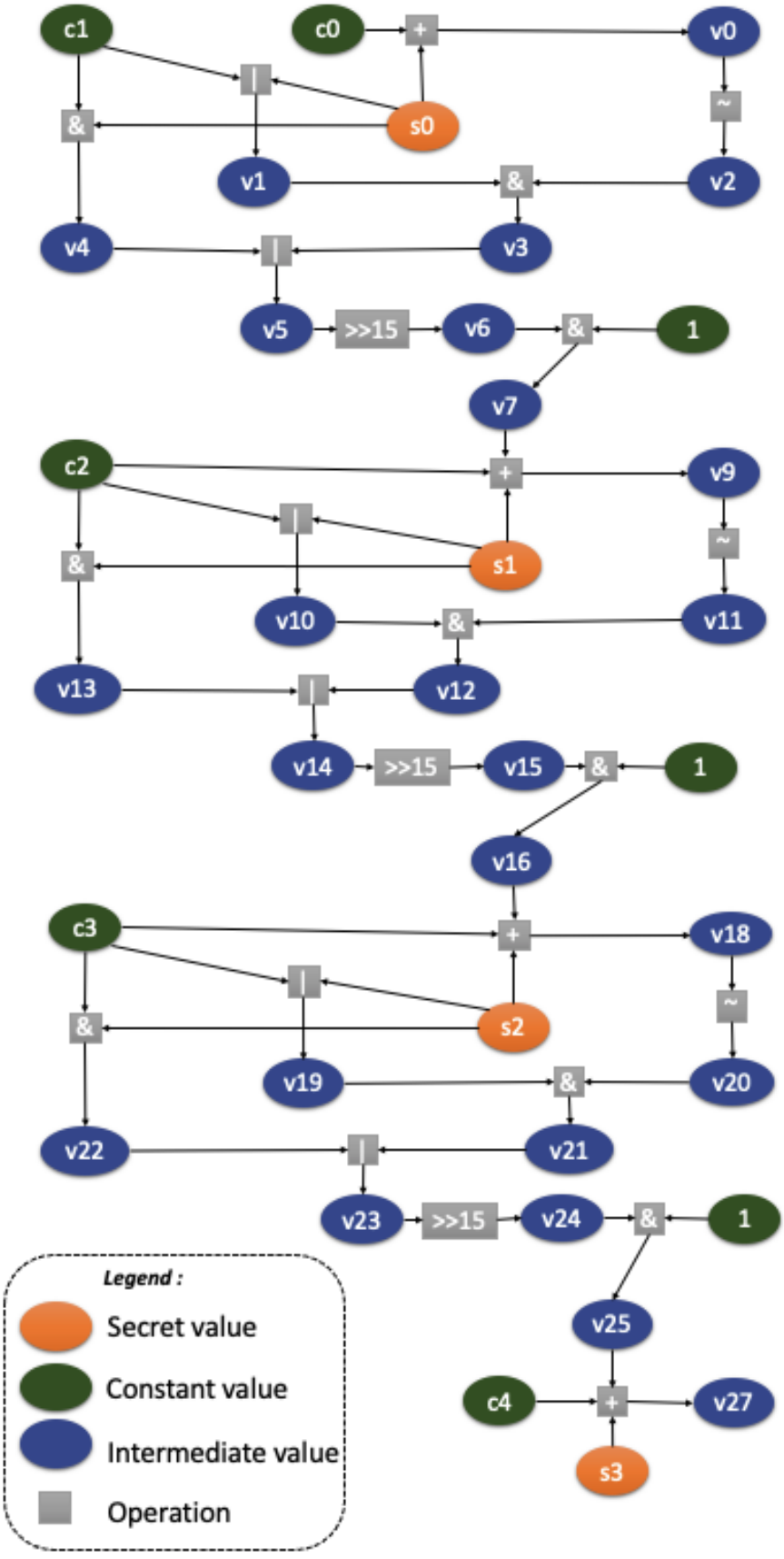

Figure A.1: SASCA graph part. 1

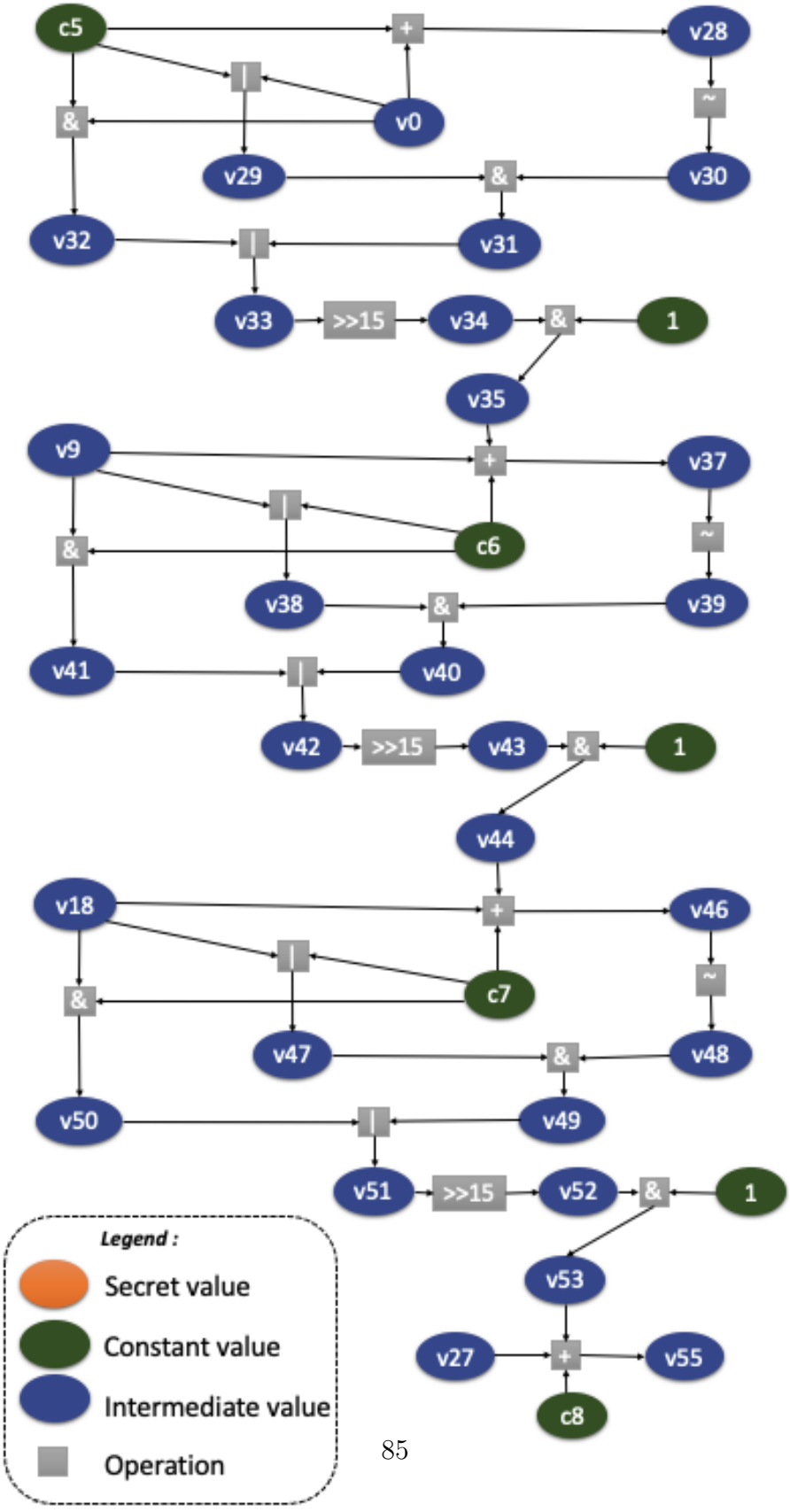

Figure A.2: SASCA graph part. 2

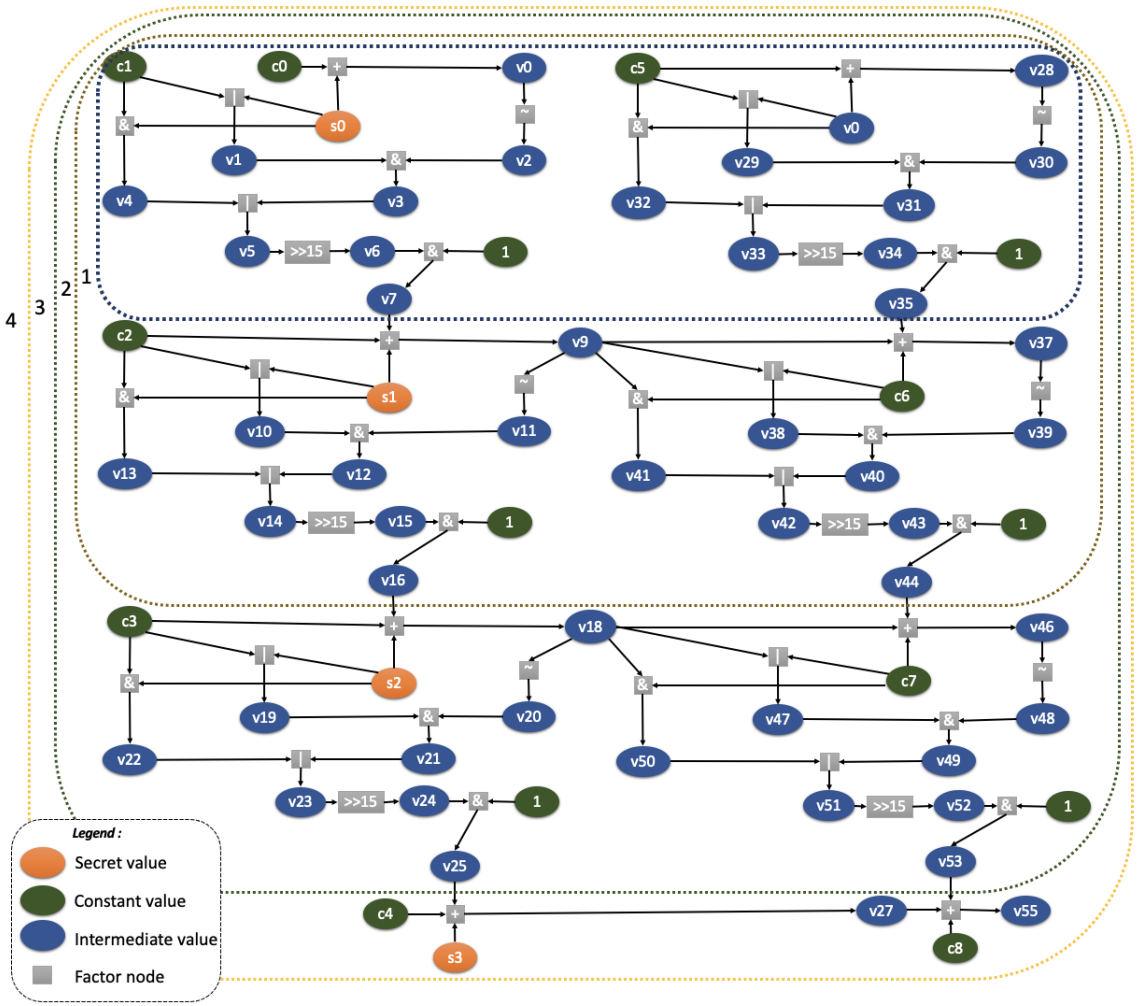

Figure A.3: Targeted parts of the graph in function of the number of secrets

Appendix B

Gaussian templates illustration

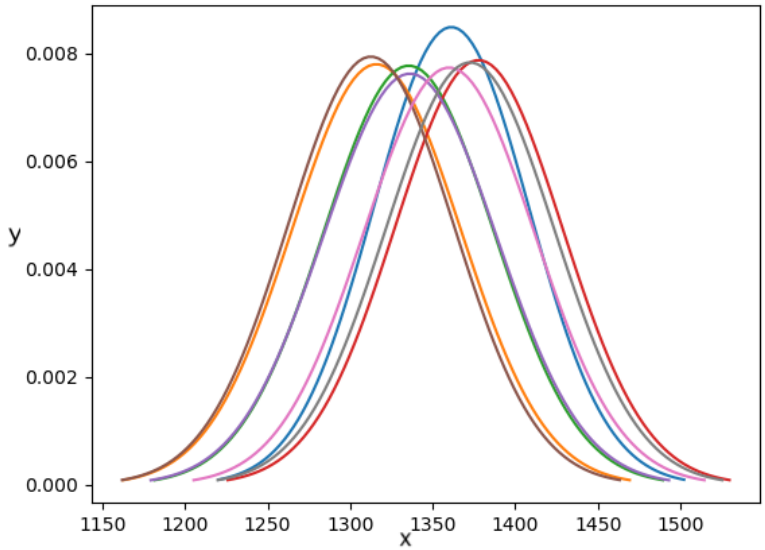

Figure B.1: Example of gaussian models built with the true leakage model for the 8 first classes

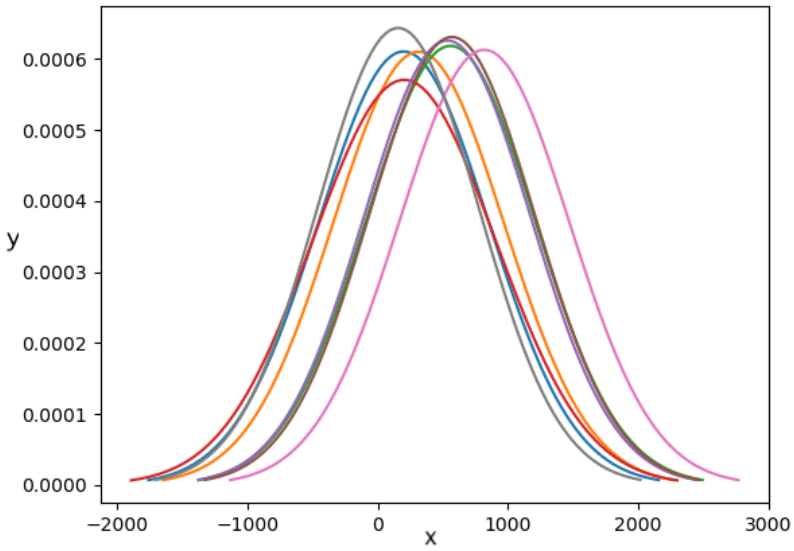

Figure B.2: Example of gaussian models built with the linear leakage model for the 8 first classes

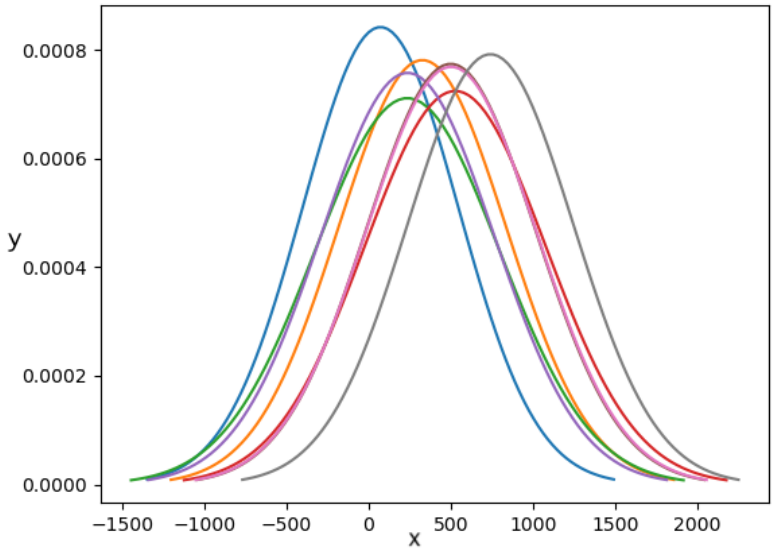

Figure B.3: Example of gaussian models built with the hamming weight leakage model for the 8 first classes

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