UCLouvain

epl

École polytechnique de Louvain

Side channel attacks against the Solo key - HMAC-SHA256 scheme

Author: **Simon COLLIN**
Supervisor: **Francois-Xavier STANDAERT**
Readers: **Olivier BRONCHAIN, Jean-Didier LEGAT, Balazs UDVARHELYI**
Academic year 2019–2020
Master [120] in Electrical Engineering

Abstract

Today, two-factor authentication tokens have become more and more popular due to the increasing number of attacks against the classical user-password online authentication. This master thesis aims to assess the security of an open source security token, the Solo key, against some side channel attacks. The Solo key relies on the U2F authentication protocol from the Fido Alliance to provide a strong second factor authentication. The surface of attack found in the authentication protocol is first described. Then a measurement setup relying on EM emanations is mounted. Next the research of the POI required by the 2 strategies attempted to break part of the HMAC-SHA256 construction is described. After that four side channel attacks, three CPA with different leakage models and a template attack are launched against the HMAC-SHA256 construction. Finally, it is shown that, employing the measurement setup and the trace alignment algorithm described, the outer hash requires a greater number than 200 000 traces to be forged while one can forge the inner hash in more or less 200 000 traces.

Acknowledgements

First, I would like to thank the Pr. François-Xavier Standaert for suggesting this interesting topic of study and accepting to be my thesis supervisor. He gave me the opportunity to choose the subject that interests me the most in this wide field of research while providing me help and feedback during the entire year. He also transmits to me, through his lectures, the passion for cryptography.

Then I would like to thank my two assistants Olivier Bronchain and Balazs Udvarhelyi who accompany me during this whole work by giving precious advice and ideas to explore. They were always available in case of questions and provided me a significant boost when I needed it the most. A particular thanks to Olivier Bronchain for giving me access to his toolboxes.

I also would like to thank Pr. Jean-Didier Legat for accepting to take part in this master thesis as a jury member.

Next, I would like to thank Charles Momin who was my hands at the laboratory during the lockdown period that we went through.

Finally, I want to thank my family, Leslie and my friends and roommates for giving me their support and their love during this master thesis and, in general, during my 5 years of study at EPL.

Contents

Chapter 1

Introduction

Due to increasing of malwares, eavesdropping, phishing, replay attacks,... online authentication through the use of only a user-password pair might not bring sufficient security. This fact encouraged the emergence of second factor authentication or even password-less protocols. As an example, Microsoft started in 2018 to promote password-less protection authentication in [1]. These protocols are often implemented in hardware tokens.

These tokens, for commercial purposes, often have strong portability, memory and price requirements that may lead to security issues. Side channel attacks (SCA), which are an active research field since the first timing attacks performed on different cryptographic implementations [2], are a variety of attacks that take advantage of the physical leakage of the targeted device. They allow to decrease the complexity of the attack from exponential to linear with the size of the cryptographic secret at the cost of physical access to the device. Moreover, this kind of attack can be succeeded without letting any physical traces on the device thanks to the electromagnetic emanations for example. For these reasons, SCA seem to be a natural way to recover the cryptographic secret of authentication hardware tokens.

Hence, one can ask about the security of these tokens in the light of the SCA. In the context of this work, the FIDO second factor authentication protocol implemented in the open source Solo key device will be attacked. While physical security of the Solo key has already been investigated against a fault injection adversary [3], its SCA resistance remains an open question. Therefore, this work aims to study the security provided by the Solo key against SCA relying on electromagnetic emanations of the device.

More in details, the goal is to achieve SCA on this token to allow the adversary to authenticate as the rightful user. The attacker will be assumed to get physical access of the token during a limited among of time, e.g. by stealing the device lying

on a colleague's desk during the weekend. This limited among of time is considered the time before the rightful owner of the device notices its disappearance. This time is a key point as it bounds the power of the attacker. In this situation, it is also required that the attacker does not leave remarkable traces on the device while breaking it. It is also assumed that the attacker owns the user-password pairs linked to the second factor authentication thanks to another kind of attacks. One more remark consists in highlighting the fact that the Solo key device is not certified according to the FIDO 'Authenticator Certification Level'. This certification assesses the level of difficulty of breaking a device including SCA.

Within this framework, the resistance of HMAC-SHA256, since it is the building block of the U2F protocol, against SCA will be analyzed. The HMAC-SHA256, as highlighted by its name, is made up of SHA256 schemes. Some SCA strategies against this scheme have been proposed in the literature. Only the most important works related to this subject will be briefly described. The first attack against this cryptographic architecture was presented in [4]. It consists of an extend and prune way to attack the architecture relying on the Hamming distance leakage model. The authors made their experimentation on a commercial FPGA board and proposed a masked implementation for this scheme. Then in [5] the same attack was reused yet using the Hamming weight model and relaxing the assumptions made in [4]. This attack also highlights the feasibility of applying partial CPA on this architecture. Moreover, it can be used to break the entire HMAC-SHA256 architecture. The same year (2013), a new strategy to attack SHA256 was presented in [6] relying on a divide and conquer way to obtain the secret. This attack requires to control the input of SHA256 and can thus only be managed to attack the inner hash of the HMAC-SHA256 structure. Finally, a more detailed explanation about how this attack is working can be found in [7]. Both [5] and [6] strategies of attack will be attempted in this work. More generally, an attack on a different authentication token relying on another authentication scheme, the one-time password, has been achieved in [8].

The contribution of this work is to assess the security of the open source yet commercialized Solo key that implements the FIDO U2F authentication protocol by performing different SCA against a HMAC-SHA256 construction in a real application. This work is articulated the following way. In chapter two the theoretical background necessary to understand this manuscript is described. Then, in chapter three, the FIDO U2F authentication protocol is described and after a short introduction about the Solo key, the way this protocol is implemented in this device is investigated to find a surface of attack. Finally, in chapter four, the whole train of thought to realize the different side channel attacks through the two strategies against the Solo key is described and the obtained results are exposed.

Chapter 2

Theoretical background

This chapter aims to present the theoretical background necessary to understand this work. First the cryptographic scheme on which the Solo key is relying on is presented. Then the tools used in the evaluation of the SCA are introduced. Finally, the side channel attacks deployed in this work and how they will be applied to recover the initial states of SHA256 are exposed.

2.1 SHA256

SHA256 is a 2001 hash function created by the NSA (National Security Agency). The purpose of this cryptographic tool is to produce a digest that has a shorter size than the size of the message that is hashed while being easy to compute and uniquely identifying the message as a fingerprint of this one. Hash functions are used in many applications such as message authentication codes, proof of work in the bitcoin scheme, password storage, files integrity verification,... The SHA256 hash function takes as input a 512 bits message and a 256 bits randomly chosen initial vector (IV) and outputs a 256 bits digest. Each of these inputs is cut into words of 32 bits leading to 16 words for the message and 8 words for the IV, also called in this framework the initial states. The algorithm of the SHA256 hash function is given in Algorithm 1.

First, concerning Algorithm 1, all operations are performed on 32 bits words. The + denotes a 32 bits modular addition. Then SIG_0, SIG_1, EP_0, EP_1, CH and MAJ function are defined as

- $SIG_0(x) = (x \ggg 7) + (x \ggg 18) + (x \gg 3)$
- $SIG_1(x) = (x \ggg 17) + (x \ggg 19) + (x \gg 10)$
- $EP_0(x) = (x \ggg 2) + (x \ggg 13) + (x \ggg 22)$

- $EP_1(x) = (x \ggg 6) + (x \ggg 11) + (x \ggg 25)$
- $CH(x,y,z) = ((\ x \ \&\ y\ ) \wedge (\sim(x)\ \&\ z\ ))$
- $MAJ(x,y,z) = ((\ x \ \&\ y\ ) \wedge (\ x \ \&\ z\ ) \wedge (\ y \ \&\ z))$

where x, y and z are 32 bits words, $x \gg b$ is the b bits shift to the right performed on x, $x \ggg b$ is the b bits rotation to the right performed on x, & is the bitwise AND operation, $\wedge$ is the bitwise XOR operation and $\sim x$ is the bitwise complement of x. Finally, K_i is a constant hard coded table of 64 random words.

Algorithm 1 SHA-256 hash function

Inputs: the 16 words message $I = (I_1,..,I_{16})$, the 8 words 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** 64 **do** ▷ Message expansion
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: $(A,B,C,D,E,F,G,H) \leftarrow (V_1,..,V_8)$ ▷ Loading the initial states
6: **for** i = 1 **to** 64 **do** ▷ Rounds of transformation
7: $\quad T_1 \leftarrow H + EP_1(E) + CH(E,F,G) + K_i + M_i$
8: $\quad T_2 \leftarrow EP_0(A) + MAJ(A,B,C)$
9: $\quad H \leftarrow G$
10: $\quad G \leftarrow F$
11: $\quad F \leftarrow E$
12: $\quad E \leftarrow D + T_1$
13: $\quad D \leftarrow C$
14: $\quad C \leftarrow B$
15: $\quad B \leftarrow A$
16: $\quad A \leftarrow T_1 + T_2$
17: **end for**
18: **return** $(D_1 \leftarrow A+V_1,...,D_8 \leftarrow H+V_8)$

The SHA256 algorithm has also three properties that are enumerated below[1].

1. Collision resistance: given s, it is difficult to find x and x' such that $H^s(x) = H^s(x')$.

[1]These properties were directly taken from the 'Introduction to Cryptography' course given by F. Koeune and O. Pereira

2. Second pre-image resistance: given s, x, it is difficult to find x' such that $H^s(x) = H^s(x')$.

3. Pre-image resistance: given s and $y = H^s(x)$, it is difficult to find x' such that $H^s(x') = y$.

Where s represents the initial vector, x and x' are two different input messages and $H^s(x)$ is the SHA256 hash function.

2.2 HMAC

The hashed message authentication code (HMAC) is a cryptographic tool developed in [9]. The purpose of this tool is to generate a message authentication code (MAC) that cannot be forged based on a key, an initial vector (IV) and a cryptographic hash function. A MAC is a tag appended to the message and that depends on this message which allows, thanks to the key, to perform an authenticated communication. The algorithm of this construction can be observed in Algorithm 2. In this algorithm, $\wedge$ denotes the bitwise XOR and $||$ the concatenation of the

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

messages. opad and ipad are fixed constants. This function is proved to be secure under some assumptions about the cryptographic hash function used in it. The SHA256 hash function fulfilled these assumptions. In the rest of this work, the HMAC algorithm will be used in combination with the SHA256 function and will be called the HMAC-SHA256 structure. A graphical representation of this structure can be seen in Figure 2.1.

In this Figure one can notice that the structure is separated in two parts, the inner hash and the outer hash. It is also possible to note that k0 and k1 are constant values. Finally, a last observation that can be made is that recovering somehow the value of k1 and k0 makes the tag forgeable.

2.3 Jitter

The jitter denotes the random variation of a presumably constant periodic signal. The clock of a micro controller is such a signal. There are multiple measurements of

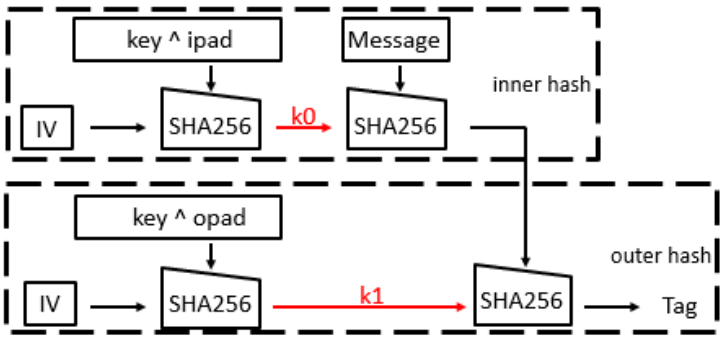

Figure 2.1: Representation of the HMAC-SHA256 structure

the jitter. The long term or absolute jitter denotes the deviation in time with refer to a perfect clock. The cycle-to-cycle jitter is the deviation from one clock cycle with refer to the previous one. The jitter can come from multiple sources, crosstalk with other signals, electromagnetic interference or simply from the generation of the clock.

This jitter has a strong impact on side channel attacks. Indeed, to perform these attacks in a feasible among of traces, it is required to have a really low jitter in the measured leakage, i.e. the number of traces to achieve an attack is increasing with the jitter present in the traces. In fact, the designers of a device can intentionally increase the jitter to reduce the potential of SCA. It thus can be seen as a countermeasure. However, algorithms to align the traces have also been studied in the literature such as in [10]. Hence, it can be seen as a game between the potential jitter that can be added by the designers without touching to the functionality of the device and the trace alignment algorithm developed by the adversaries.

2.4 Signal to noise ratio

The signal to noise ratio (SNR) is a strong cryptographic tool in the scope of the point of interests detection in the leakage and to assess the number of leakage traces needed to recover the secret information. The SNR depicted the ratio between the meaningful part of the signal and its unwanted background noise.

The SNR in the case of the research of a particular state of a cryptographic algorithm inside a set of traces can be estimated as

$$SNR = \frac{var(\mu_c)}{mean(\sigma_c)}$$

for one time sample of the leakage where μ_c and σ_c represent respectively the mean and the variance of the traces linked to a class value. This class value corresponds to a given state value in a cryptographic scheme. It thus requires to compute this state value for each trace and to sort them in the corresponding classes before performing the SNR.

Thanks to the SNR the number of traces N required to reach a high success rate in recovering the secret can be approximated as

$$N \simeq \frac{c}{SNR}$$

where c is an empirical constant value.

2.5 Rank estimation and key enumeration

Rank estimation is a cryptographic tool that allows to assess the computational security of an attack for which an adversary obtained partial part of the secret thanks to a divide and conquer strategy. In case the attack is not successful in perfectly recovering all partial secrets due to, for example, a lack of traces, a key enumeration algorithm allows to search the key starting with the keys that are the most likely. This is the conquer phase of an attack. However, this key enumeration is bound by the power of the adversary. The rank estimation algorithm presented in [11] relying on histograms and used in this work allows to assess the rank of the key with bounds tighter than one bit. It thus allows to know if the key is just above the enumerable key range of the attacker, in which case a small improvement of the computational power of the adversary or of the attack will recover the key, or far away from this one, which is translated by a more secure device.

2.6 Side channel attacks

Side channel attacks take advantage of the information contained in the physical leakage of a device implementing a cryptographic scheme rather than directly attacking the cryptographic scheme which is often proved to be secure through mathematical tools. Indeed, they allow to decrease the complexity of the attack from exponential to linear with the size of the cryptographic secret. This physical leakage can take multiple forms such as time information, power consumption, electromagnetic radiation,... Power consumption and electromagnetic radiation are closely linked as they both depend on the current intensity inside the device. In this framework, SCA taking advantage of the electromagnetic radiation will be investigated. Since the differential power analysis performed in [12], a lot of

different implementations of SCA relying on power consumption varying statistic tools used to recover the secret information or the modeling of the physical leakage have been proposed. In the next part of this section the two SCA used in this work will be described.

2.6.1 Correlation power analysis

In the correlation power analysis (CPA), the adversary makes the assumption that a linear correlation exists between the modeled leakage value and the one measured. The modeled leakage value is linked to the value of a specific operation performed in the device. This model can be simple such as the Hamming weight or the Hamming distance of this value or more complex requiring a profiling phase to obtain it. The linear correlation is captured by the Pearson's correlation coefficient. As a reminder, this coefficient is equal to

$$\rho_{xy} = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2}\sqrt{n\sum y_i^2 - (\sum y_i)^2}}.$$

The algorithm to recover the secret value is given in Algorithm 3. It consists in testing all potential values k of the secret K. A modeled leakage value M is computed for each hypothetical secret and for each input message. The Pearson's correlation coefficient between the set of leakage traces and the modeled leakage is calculated for each time sample of a sampling window. The hypothetical secret that outputs the higher correlation is considered as the right value of the secret.

Algorithm 3 CPA algorithm

Inputs: the set of N leakage traces T in a sampling window S, the set of N input messages P and the length of the secret L.
Output: the secret K.

for k = 0 **to** 2^L **do**
  for i = 1 **to** N **do**
    $M_i \leftarrow$ model(k,P_i)
    for s **in** S **do**
      R[s,k] $\leftarrow \rho(T_s, M_i)$
    end for
  end for
end for
K $\leftarrow$ argmax$_{s,k}$(R[s,k])
return K

If L is too high, the CPA might be computationally too intense to be performed. However, partial CPA makes it possible by separating the secret K into b blocks of length L/b. Indeed, it reduces the computational cost from $\mathcal{O}(2^L)$ to b*$\mathcal{O}(2^{\frac{L}{b}})$. The side effects of applying partial CPA are that it might be not straightforward as blocks can be related to each other and that the L-(L/b) other bits are considered as noise since they are not taken into account in the modeling of the leakage.

2.6.2 Template attack

The univariate template attack is an SCA that allows to trade off the required number of traces to succeed the attack between a profiling phase and an online attack phase.

During the profiling phase, the attacker estimates the leakage relying on the assumption that the leakage of every potential value of a cryptographic operation follows a normal distribution,

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

where μ_i and σ_i are respectively the mean and the variance of the leakage corresponding to this potential value. However, to assess these distributions it requires to have better control of the device targeted. Indeed, it is needed to know the output of the cryptographic operation and the capability of making it varies. Moreover, it requires to know exactly the time sample in the leakage where the targeted cryptographic operation is performed to decrease the computations. However, it allows to compute a precise model of the leakage. This phase is usually performed on a similar device to the one targeted as it might require a long period of time to achieve the profiling phase.

Then, thanks to the leakage model derived by the profiling phase, one can perform the online attack. This attack, requiring hopefully fewer traces than the profiling phase, aims to combine the Bayes probability of each key candidate for each trace. This combination can be made by multiplying the probabilities or accumulating the logarithm of those probabilities. The key is recovered by taking the key candidate that obtained the maximal combined probability. This phase is obviously performed on the targeted device.

2.7 Attacks against SHA256

This section aims to present two strategies in order to find the initial value of A, B, C, D, E, F, G and H, also called the initial states, by performing SCA against the

SHA256 implementation. Both strategies will be described and their requirements exposed. Then the strategies will be summarized in a comparative tabular which intends to highlight the pros and cons of each of them.

2.7.1 Divide and conquer strategy

This attack [6] relies on a divide and conquer strategy meaning that the partial secrets recovered to form the full secret are independent of each other. The attack is divided into two stages. First, intermediate secret values are recovered during four rounds thanks to SCA with the rounds before the one attacked fixed. Then, the initial states are found back relying on the knowledge of these intermediate secret values by inverting the equations of SHA256.

The targeted operations in the SHA256 function are

$$\begin{aligned} E_i &\leftarrow D_{i-1} + H_{i-1} + EP_1(E_{i-1}) + CH(E_{i-1}, F_{i-1}, G_{i-1}) + K_i + M_i \\ A_i &\leftarrow H_{i-1} + EP_1(E_{i-1}) + CH(E_{i-1}, F_{i-1}, G_{i-1}) + K_i + M_i + EP_0(A_{i-1}) \\ &\quad + MAJ(A_{i-1}, B_{i-1}, C_{i-1}) \end{aligned}$$

during the four first rounds of SHA256 where i is the ith round and T_1 and T_2 are developed, i.e. two partial secrets are recovered per round. To highlight the secrets that are recovered per round, one can rewrite the previous equation as

$$\begin{aligned} E_i &\leftarrow E_i secret + M_i \\ A_i &\leftarrow A_i secret + M_i \end{aligned}$$

where $E_i secret$ and $A_i secret$ are respectively the secret related to the operation that outputs E_i and A_i. For those secrets to be a constant value in the ith round, as SCA target a constant secret value manipulated with a known value (M_i here), the messages of the previous rounds must be fixed. Moreover, once the message is fixed to a value to attack the next round, its value must remain the same when the next round is attacked. For example, when attacking the second round the value of M_1 must be fixed. Then when attacking the third round, the value of M_1 and M_2 must be fixed and the value of M_1 must remain the same as the one used during the attack in the round two. Indeed, the value E_1 used in the equations 2.1 to 2.8 depends on the message M_1 fixed. The value E_2 used in the same equations depends on the fixed M_2 and E_1 since $E_2 secret$ is linked to E_1. And so on. Therefore, the validity of the equations 2.1 to 2.8 requires from the attacker to have full control of the three first input messages. When the four first rounds were attacked, $E_{1,2,3,4} secret$ and $A_{1,2,3,4} secret$ are known as well as $E_{1,2,3,4}$ and $A_{1,2,3,4}$.

This attack takes advantage of the inter-round relationships between the states of SHA256 e.g. H_3 is equivalent to G_2, to F_1 and to E_0 where H_3 denotes the value

of H at the third round and so on. As example, the attack to recover E_0 relies on the next equation

$$A_4 \leftarrow H_3 + EP_1(E_3) + CH(E_3, F_3, G_3) + K_4 + M_4 + EP_0(A_3) + MAJ(A_3, B_3, C_3)$$

which can be rewritten as

$$A_4 \leftarrow E_0 + EP_1(E_3) + CH(E_3, E_2, E_1) + K_4 + M_4 + EP_0(A_3) + MAJ(A_3, A_2, A_1).$$

E_0 can thus be obtained since all other terms of the equation are known. An overview of the inter-round relationships can be seen in the Figure 2.2. In this Figure, the known values, recovered thanks to the SCA, are in red and the values that are recovered using the equations 2.1 to 2.8 are in blue.

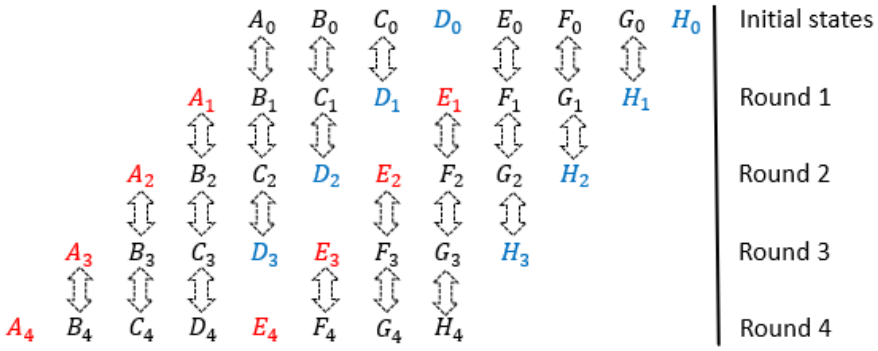

Figure 2.2: Overview of the inter-round relationships. In red the values recovered through the SCA. In blue the values recovered through the equation 2.1 to 2.8.

The equations that allow to recover A_0, B_0, C_0, D_0, E_0, F_0, G_0 and H_0 through the inter-round relationships are

$$H_3 \leftarrow A_4 - (EP_1(E_3) + Ch(E_3, F_3, G_3) + K_4 + EP_0(A_3) + MAJ(A_3, B_3, C_3) + M_4) \tag{2.1}$$

$$D_3 \leftarrow E_4 - (H_3 + EP_1(E_3) + Ch(E_3, F_3, G_3) + K_4 + M_4) \tag{2.2}$$

$$H_2 \leftarrow A_3 - (EP_1(E_2) + Ch(E_2, F_2, G_2) + K_3 + EP_0(A_2) + MAJ(A_2, B_2, C_2) + M_3) \tag{2.3}$$

$$D_2 \leftarrow E_3 - (H_2 + EP_1(E_2) + Ch(E_2, F_2, G_2) + K_3 + M_3) \tag{2.4}$$

$$\begin{aligned} H_1 \leftarrow A_2 - (&EP_1(E_1) + Ch(E_1, F_1, G_1) + K_2 + EP_0(A_1) \\ &+ MAJ(A_1, B_1, C_1) + M_2) \end{aligned} \tag{2.5}$$

$$D_1 \leftarrow E_2 - (H_1 + EP_1(E_1) + Ch(E_1, F_1, G_1) + K_2 + M_2) \tag{2.6}$$

$$\begin{aligned} H_0 \leftarrow A_1 - (&EP_1(E_0) + Ch(E_0, F_0, G_0) + K_1 + EP_0(A_0) \\ &+ MAJ(A_0, B_0, C_0) + M_1) \end{aligned} \tag{2.7}$$

$$D_0 \leftarrow E_1 - (H_0 + EP_1(E_0) + Ch(E_0, F_0, G_0) + K_1 + M_1). \tag{2.8}$$

The disadvantages of this attack are that it requires four, one for each round attacked, different sets of traces and the control of the three input messages of the SHA256 function.

2.7.2 Extend and prune strategy

This second strategy [5] is more delicate as it consists in recovering the whole secret by building it on partial parts of this one, i.e. the first part of the secret is necessary to obtain the second part which is necessary to obtain a third part and so on. It signifies that due to its intrinsic design, it is not possible to make a conquer phase in this attack and a wrongly guessed partial secret at the beginning of this strategy will lead to further errors.

The extend and prune strategy consists of eight SCA that are performed one after the other and are focused during the first and the second rounds of the SHA256 algorithm. The different steps of the attack are :

1. Recover α by performing SCA on $T_1 \leftarrow \alpha + M_1$ where $\alpha = H_0 + EP_1(E_0) + CH(E_0, F_0, G_0) + K_1$. Since H_0, E_0, F_0 , G_0 and K_1 are constant, α is constant making the SCA possible. The variable T_1 can be determined for each M_1 after this attack.

2. Recover D_0 by performing SCA on $E_1 \leftarrow D_0 + T_1$. This second step relies directly on the first stage as it is needed to know the variable T_1. The variable E_1 can be determined for each T_1 after this attack.

3. Recover T_2 by performing SCA on $A_1 \leftarrow T_1 + T_2$. T_2 is constant since A_0, B_0 and C_0 are constant. This step also relies on the first stage and the variable A_1 can also be computed for each T_1.

4. During the second round, recover the constant $F_1 = E_0$ (from the inter-round relationship) by performing SCA on E_1 & F_1 in the CH(E_1,F_1,G_1) function. This step relies on the second stage of the attack.

5. Recover the constant $G_1 = F_0$ by performing SCA on $\sim E_1$ & G_1 in the CH(E_1,F_1,G_1) function. This step also relies on the second step of the attack.

6. Recover the constant $B_1 = A_0$ by performing SCA on A_1 & B_1 in the MAJ(A_1,B_1,C_1) function. This step relies on the third stage of the attack.

7. Recover the constant $C_1 = B_0$ by performing SCA on A_1 & C_1 in the MAJ(A_1,B_1,C_1) function. This step also relies on the third stage of the attack.

8. Recover the constant $H_1 = G_0$ by performing SCA on $T_1 \leftarrow H_1 + EP_1(E_1) + CH(E_1, F_1, G_1) + K_2 + M_2$ where E_1, F_1, G_1, K_2 and M_2 are known. E_1 must be computed from M_1 for each M_2. To compute this step, the fifth, the fourth and the second stage of the attack must be successfully realized.

9. Finally, H_0 and C_0 can be deduced thanks to all the previous stages. H_0 can be figured out in $T_1 \leftarrow H_0 + EP_1(E_0) + CH(E_0, F_0, G_0) + K_1 + M_1$. And C_0 can be found in $T_2 \leftarrow EP_0(A_0)$ + MAJ(A_0,B_0,C_0).

Performing all these steps requires only 1 set of traces and no condition on the input message except its randomness. However, since the targeted states of SHA256 in the stage four to seven of the attack strategy are the result of a bitwise AND, it is important to observe that, in the case of the secret K is equal to 0, performing SCA on K & M_i will not bring any information as the state will be equal to 0 whatever the message is. In other words, the hypothesis K equals to 0 cannot be assessed.

2.7.3 Comparison of the strategies

The Table 2.1 summarizes the different advantages and inconveniences of each strategy. A general analysis is that the extend and prune strategy necessitates fewer requirements than the divide and conquer strategy but is more tricky to set up. An additional remark is that the divide and conquer strategy is not applicable in the outer hash of the HMAC to recover k1 (see section 2.2) because of the conditions on the input messages. Indeed, the input message of the SHA256 construction of the outer hash is the output of the last SHA256 construction of the inner hash and by the propriety of the SHA256 scheme, its output is uncontrollable. Moreover, to apply the extend and prune strategy to the outer hash, the initial states of the inner hash have to be guessed with high confidence since the input messages of

the outer hash have to be guessed from the input messages of the inner hash and its initial states. Finally, the rank estimation and key enumeration cryptographic tools are applicable, by definition, only in the case of the recovery of independent partial secrets. They are thus unusable in the extend and prune strategy.

	Divide and conquer	**Extend and prune**
Number of SCA	8	8
Number of sets of traces	4	1
Number of round attacked	4	2
Targeted operations	Modular additions	Modular additions and bitwise AND operations
Control required on M_i	Full for M_1, M_2 and M_3 Randomness for M_4	Randomness for M_1
Number of hypotheses that cannot be assessed	0	1, key = 0
Propagation of the errors between the SCA	No	Yes
Rank estimation and key enumeration	Yes	No
Applicable to inner hash of HMAC	Yes	Yes
Applicable to outer hash of HMAC	No	Yes

Table 2.1: Comparison between the divide and conquer strategy and the the extend and prune strategy

Chapter 3

FIDO U2F protocol and Solo key

In this chapter, the FIDO U2F authentication protocol implemented in the Solo key will be described. Then the different characteristics of the Solo key will be shortly exposed and the implementation of the FIDO U2F protocol on it will be analyzed.

3.1 FIDO U2F protocol

The FIDO Universal Second Factor (U2F), also called CTAP1, authentication protocol has been developed by FIDO Alliance. The purpose of this protocol is to amplify the security of web services such as Microsoft, Google, GitLab and many others, by relying on a second factor authentication device that implements the U2F protocol, such as a security key, in addition to the existing and traditional username-password authentication to perform a login.

The U2F protocol is performed in two distinct phases. First, the registration phase during which the U2F token of a user is registered with the service. Then, the authentication phase is performed each time the user wants to access his account. During this phase, the service verifies that the token is the one used in the registration phase. Once the registration phase has been done, it is impossible to access an account without a U2F token bound to this account.

The 2 phases rely on cryptographic assertions that are needed to be verified by a relying party. The service is thus connected to the relying party which can be, for example, a web server. These two phases will now be described in details and an overview of these phases can be seen in Figure 3.1 and Figure 3.2. This protocol can be found in [13] and [14].

3.1.1 Registration phase

The different steps of the registration phase go as follows:

1. The service sends to the U2F token the concatenation of the challenge parameter given by the relying party and the application parameter representing the service identity requesting the registration. Note that the challenge parameter is saved by the relying party.

2. The U2F protocol requires proof of the user presence by pressing a button on the device for example.

3. The U2F token generates a new private-public key pair and a key handle that allows the token to recognize the generated key pair. The public key is derived from the private key as the coordinates of point on the P-256 National Institute of Standards and Technology elliptic curve.

4. The U2F token saves the private key.

5. The U2F token signs with its attestation key corresponding to its attestation formulate thanks to the elliptic curve digital signature algorithm (ECDSA)[15], the concatenation of 0x00 (hexadecimal number representation), the application parameter, the challenge parameter, the key handle and the public key.

6. The token sends to the service the concatenation of 0x05 (legacy reason), the public key, the key handle length, the key handle, the attestation certificate and the signature previously computed.

7. The service verifies through the relying party the attestation certificate which has to be issued by a trusted certification authority and the signature. The signature can verified since the message as well as the elliptic curve parameters and the public key which is contained in the attestation certificate are known by the relying party.

8. If the conditions are fulfilled, the relying party stores the key handle and the public key and it sends a succeed registration message to the service.

3.1.2 Authentication phase

The steps of the authentication phase are described below:

1. The service contacts the relying party and receives the key handle length, the key handle and the challenge parameter corresponding to the registration of this account.

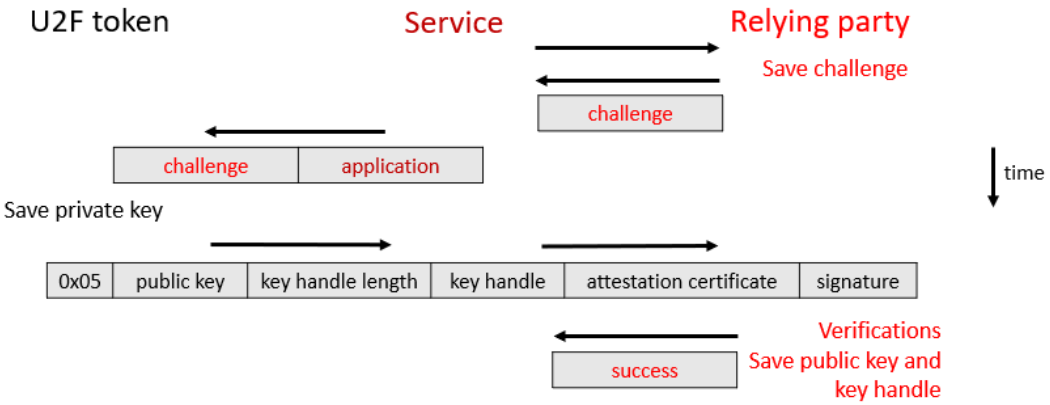

Figure 3.1: Overview of the registration phase.

2. The service sends to the token the concatenation of a control byte, the challenge parameter, the application parameter, the key handle length and the key handle. The control byte has 3 possible values. 0x07, the U2F token only verifies if the key handle and the application parameter correspond and were created by it. No signature is provided in this case. 0x03, requires the presence of the user to perform the end of the protocol. 0x08, the presence of the user is not required to perform the end of the protocol.

3. Depending on the control byte, the U2F may or may not require proof of the user's presence.

4. The token verifies if the key handle was created by it and corresponds with the application parameter.

5. The token signs with the private key corresponding to this key handle, still with the ECDSA, the concatenation of the application parameter, the user presence which is just a byte at 0 or 1 whether the presence of the user was verified or not, a counter value that is incremented by 1 each time an authentication occurs and the challenge parameter.

6. The token sends to the service the concatenation of the user byte, the counter and the signature.

7. The service verifies through the relying party that the signature is correct thanks to the public key saved during the registration. The relying party also verified the counter value which is a countermeasure from malicious duplicating of the key. Effectively by verifying that the counter value associated with a key handle value is only increasing a mismatch between a rightful key

and its duplication is likely to occur. In other words, if the duplicated key uses a counter value lower than the one contained by the relying party, then, the authentication will be rejected. If, in the opposite way, the duplicated key uses a counter value higher than the authentication will occur. However, the user of the rightful key might be rejected in its further authentication if its counter value is lower than the one used by the duplicated one. The user thus knows that his authenticating device is compromised.

8. If the conditions are fulfilled, the relying party authorizes the service to proceed to the authentication and saves the counter value.

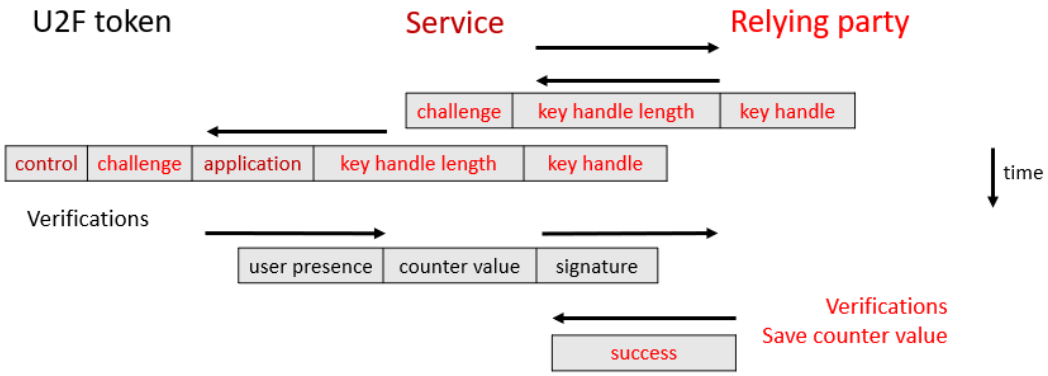

Figure 3.2: Overview of the authentication phase.

3.2 Solo key

Figure 3.3: Solo key authentication token

The Solo key is the first open source security key that implements the FIDO U2F protocol as well as the FIDO CTAP2 protocol, another protocol for passwordless, second factor or multi-factors authentication also developed by the FIDO Alliance group. There are different kinds of Solo keys but the ones this work is focused on are the "normal" Solo and the "hacker" Solo. Both of these Solo keys have the

same purpose and thus rely on the same hardware and firmware. They diverge by the fact that the solo hacker's firmware is programmable, i.e. it is possible to load unsigned by SoloKeys application. It is important to note that the implementation of the U2F protocol on the Solo key is not FIDO certified by their 'Authenticator Certification Level' meaning that there might exist some hardware attack on it. It has already been demonstrated by [3].

The token relies on the STM32L432 micro controller (MCU) running at 48 MHz to perform its cryptographic operations. This MCU also has a true random number generator that guarantees the randomness needed by those cryptographic operations. The open source repository of the Solo key can be found in [16]. In this repository the whole hardware and software of the Solo keys can be found as well as some Python functions. These Python functions are useful especially for programming the hacker Solo key or simulating cryptographic functions. Thanks to this repository the implementation of the U2F protocol on the Solo key can be analyzed and, more precisely, the way the private key is computed during the registration phase or the authentication phase of the protocol.

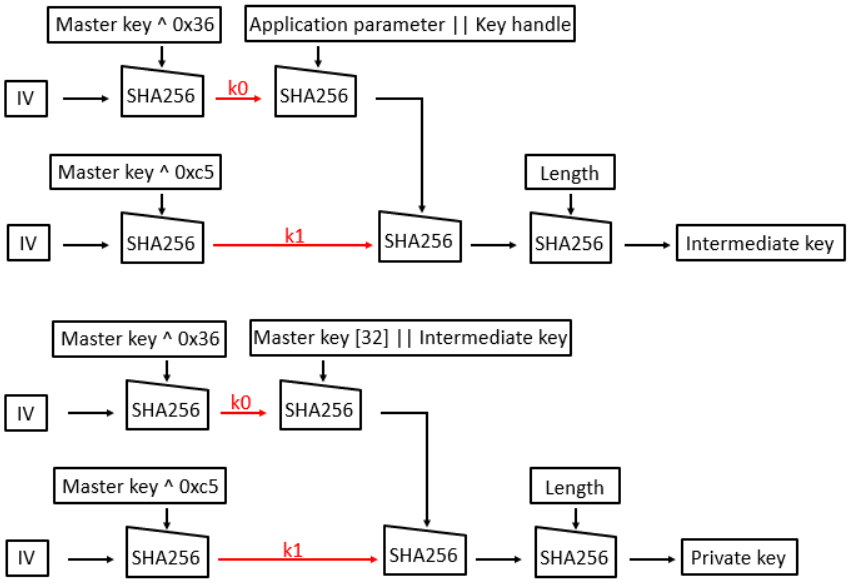

Figure 3.4: Derivation of the private key in the Solo key

During the registration phase, as it can be seen in the Figure 3.4, it consists of two HMAC-SHA256 structures in cascade. The first one is fed with the master key (64 bytes), a secret key that is generated at the first boot of the token and which is independent from one token to the other, then with the concatenation of a random number generated by the token (32 bytes), the key handle, and the application

parameter (32 bytes) received by the service. After that, a second HMAC-SHA256 is fed with the master key as well and the result of the first HMAC-SHA256 concatenated with the 32 LSBs (in the context of this work, LSB refers to least significant *byte*) of the master key. The "Length" represents the addition of the length of all previous input parameters of the HMAC construction. During the authentication phase, the private key is computed following the same structure except that the key handle is now a message sent by the service. The verification that the key handle was created by it and corresponds to the application parameter is made by comparing whether or not the private key (re)generated by this scheme was saved, earlier in a registration phase, on the token.

Since the initialization vector (IV) and the master key are constant, k0 and k1 (see Figure 3.4) are constant and represent a surface of attack to recover all private keys. Effectively by finding those two initial states of the SHA256 construction and with the parameters given by the service during the authentication phase (see section 3.1.2), it is possible to derive the private key corresponding to that key handle and to that application parameter.

Chapter 4

Experimental results

This chapter has as purpose to expose the train of thought of the side channel attacks through the divide and conquer strategy and the extend and prune strategy realized on the Solo key to recover k0 and k1 and the experimental results that accompany it. First the measurement acquisition setup will be described. Then the required trace alignment algorithm will be exposed. Next, the search of the points of interest (POI) for each strategy that are necessary to mount the attacks against the HMAC-SHA256 construction present in the Solo key will be described. Finally, the different side channel attacks used will be explained, launched and if succeeded compared to each other in terms of number of traces needed to find back the secret. All the background necessary to understand this section is available in Chapter two.

4.1 Measurement acquisition setup

The first step to carry out a side channel attack is to define the way the leakage of the micro controller, the traces, is measured and how the architecture is stimulated to deliver encryptions.

4.1.1 Leakage measurement

The acquirement of the leakage is performed by an electromagnetic near field probe R&S HZ-15 and a matched preamplifier R&S HZ-16. The preamplifier which is performing a 20dB gain allows to measure very weak high frequency electromagnetic field. The obtained signal is sampled at 500 [MSamples/s], 10,42 times the micro controller frequency, at 12 bits resolution with a PicoScope 5000 series USB oscilloscope. The voltage range of the PicoScope is set to $\pm$20mV with as requirement to maximize the signal while preserving it from an overflow. The

probe is placed just above the micro controller at the position that magnify the information contained in the signal. An overview of the measurement setup can be observed in Figure 4.1. This leakage measurement is non-invasive meaning that among other benefits no footprints are left on the targeted device.

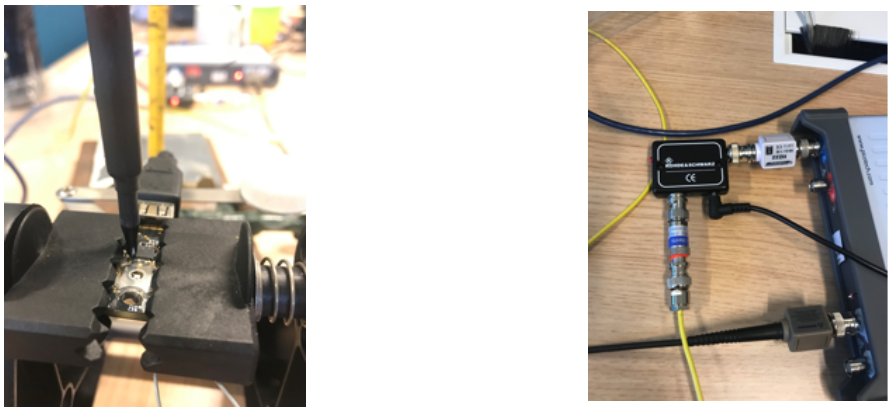

Figure 4.1: Overview of the setup of measurements. On the left: R&S HZ-15 probe just above the MCU of the Solo key. On the right: matched R&S HZ-16 preamplifier and PicoScope.

Then, another important point is the triggering technique for this measurement setup in order to measure only the part of the leakage that is interesting to perform the attack and so decrease the memory needed to store all the leakage measurements necessary to mount the attack. It also suppresses the long term jitter that can be present due to noisy clock and thus performs a synchronization of the leakage traces. To achieve this trigger on the hacker Solo key, a modified software version is used to send a message with the Universal Asynchronous Transmitter Receiver (UART) communication protocol. This message, containing only zeros, is used as a trigger from the key to the external world. The UART communication protocol was already implemented in the Solo key and the electric signal is easily recoverable as pins to measure it are available on the surface of the Solo key. These pins can be seen in the Figure 4.2.

This triggering technique applied just before the HMAC construction of the Solo key allows to output the leakage measurement in Figure 4.3. In this Figure, five distinct groups related to the five executions of the SHA256 inside the HMAC construction (see Figure 3.4) can be observed. It can be noticed that this leakage shape is really similar to the one in [17]. The trace is also really noisy since all the different spikes that occur at regular intervals are unwanted and are shifted from one trace to the other.

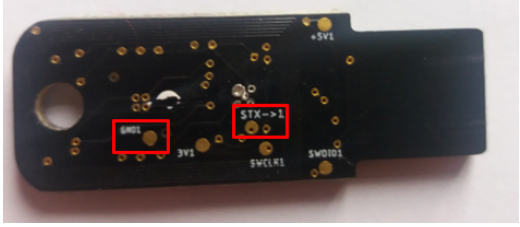

Figure 4.2: Back of the Solo key. The right most pin highlighted by the red square corresponds to the UART sending pin. The other highlighted pin is a ground reference voltage.

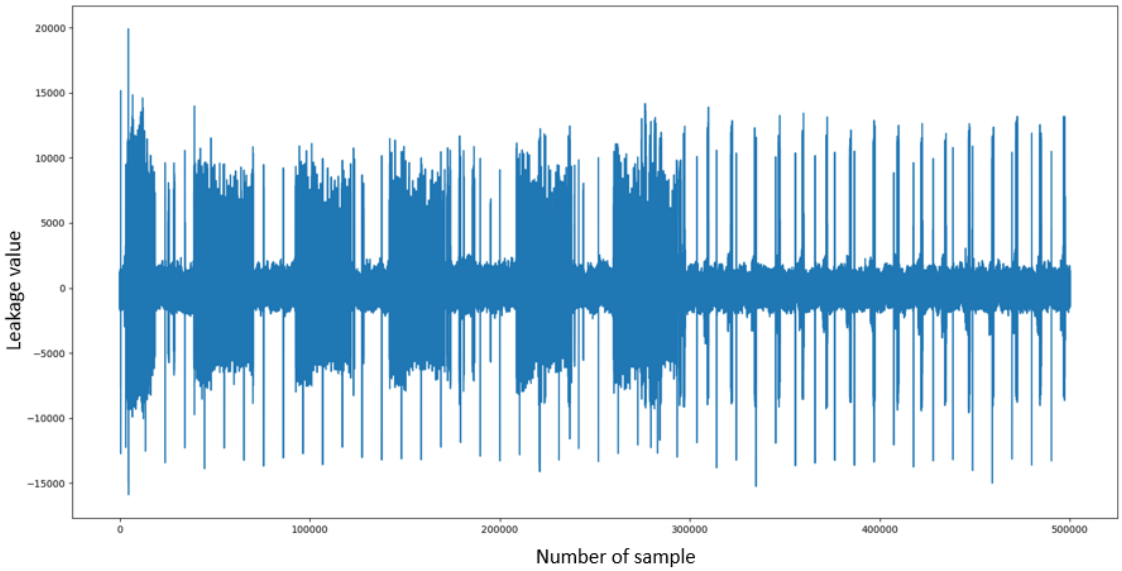

Figure 4.3: EM leakage observed by triggering at the begin of the HMAC protocol.

4.1.2 Stimulation of the encryption

To stimulate the encryption, a python method from the Solo keys library is used. This method named "challenge-response" performs an HMAC(K, H(challenge)) where K is a key linked to a previous credential generation, the challenge is a chosen input and H is a SHA256 hash function. This method thus stimulates the HMAC implementation of the Solo key which is the surface of attack. However, some changes have been brought to this method. The first modification allows to choose K without relying on a credential generation. Then, the second argument of the HMAC function has been changed to simply the challenge and not the digest of this one allowing to directly manipulate the input of the HMAC structure. The need of a user's presence (pushing the button) required by the method was also

disabled. Thanks to all of these modifications and to the measurement setup, the automated recording traces rate is around 1.5 traces per second.

These changes allowing an automatic encryption stimulation choosing both the key and the plaintext (the input message) are possible due to the reprogramming capability of the hacker version of the Solo key. Nonetheless they are not straightforward at all to apply to the normal version of the Solo key.

4.2 Search of the points of interest

In this section, the points of interest corresponding to the samples of the leakage that are the most interesting will be searched for both strategies. This research, which relies on the SNR statistical tool, is essential since it allows to save a lot of computational power during the attack phase. As a reminder the two different strategies to attack the HMAC construction are the divide and conquer strategy and the extend and prune strategy. Both of those strategies have their advantages and their inconveniences (see Section 2.7.3) but the most important thing to notice is that only the extend and prune strategy is applicable to recover the secret of the outer hash, k1. Solely the search of the points of interest of the inner hash second SHA256 computation for the two strategies will be exposed in this section aiming to recover k0. However, remember that one willing to break the HMAC construction allowing to compute the private key of the Solo key needs to recover both k0 and k1 and one should thus also apply the methodology to recover the points of interest of the extend and prune strategy on the outer hash second SHA256 computation.

4.2.1 Alignment of the traces

The statistical tools used in the search of the POI but also in the attacks are really efficient only if the traces are aligned, meaning that the samples of one MCU operation in one trace have to correspond to the same samples in time of another trace. Moreover, the attacks performed on non aligned traces require much bigger to infeasible traces set. Despite the trigger, the traces are not aligned due to cycle to cycle jitter coming from the clock of the micro controller. This can be seen on the upper graph in Figure 4.4. Effectively since there is no external (from the MCU point of view) clock generation in the Solo key, the clock is self produced by the micro controller. This kind of clock generation is very likely to be the source of all of this cycle to cycle jitter. A crystal quartz could be mounted on the micro controller to counteract this jitter but would leave noticeable physical fingerprints on the device.

To counteract this jitter, a trace alignment algorithm is necessary. There exist two kinds of methods to realign the traces, the static ones and the dynamic ones.

The static methods consist in evaluating the offset difference between two traces and then shifting back this offset in one of the traces. The dynamic ones propose a non-linear resampling of the traces to be aligned to match a trace. One static algorithm that requires low computations is the cross-correlation alignment.

Algorithm 4 Cross-correlation alignment

Inputs: the model trace M, the set of N traces to be aligned T = (T_1,..,T_N), the alignment window AW, the correlation threshold CT, the interest window IW and the spike noise threshold NT.
Output: the aligned traces set.

```
1: for i = 1 to N_traces do
2:     C ← cross_correlation(M[AW],T_i[AW])
3:     if max(C) > CT && max(T_i[IW]) < NT then
4:         Offset ← argmax(C) - len(AW)
5:         T_i ← roll(T_i, Offset)
6:     else
7:         supress T_i from T
8:     end if
9: end for
10: return T
```

This algorithm, referring to the pseudocode Algorithm 4, consists in measuring the cross-correlation in a well chosen "alignment window" between a reference trace and all traces belonging to the set of traces. Then it shifts (roll function in Algorithm 4) the traces according to the amount of offset that maximizes this cross-correlation. Two additional steps are made before the shifting. First, all the traces that do not reach a certain amount of cross-correlation are discarded meaning that the discarded trace will not be well aligned or not aligned at all with refer to the model trace. Then all traces that possess a random spike in the windows of interest are also discarded allowing to reduce the noise in the remaining traces set. Multiple parameters have to be wisely selected in this algorithm. First the alignment window is chosen such that it contains two non noisy spikes. This has as purpose to maximize the cross-correlation when the spikes are aligned and to allow an easy threshold to discard traces not well aligned. Then the reference trace must be chosen such that these 2 non noisy spikes lie in more or less the middle of the potential range of offset. This reference trace must also be noiseless in the window of interest. All of this can be observed in Figure 4.4. The alignment window is from sample number 11600 to sample number 11750 and the window of interest from sample number 11800 to sample number 12500. In general, more or less 30% of the initial traces are discarded.

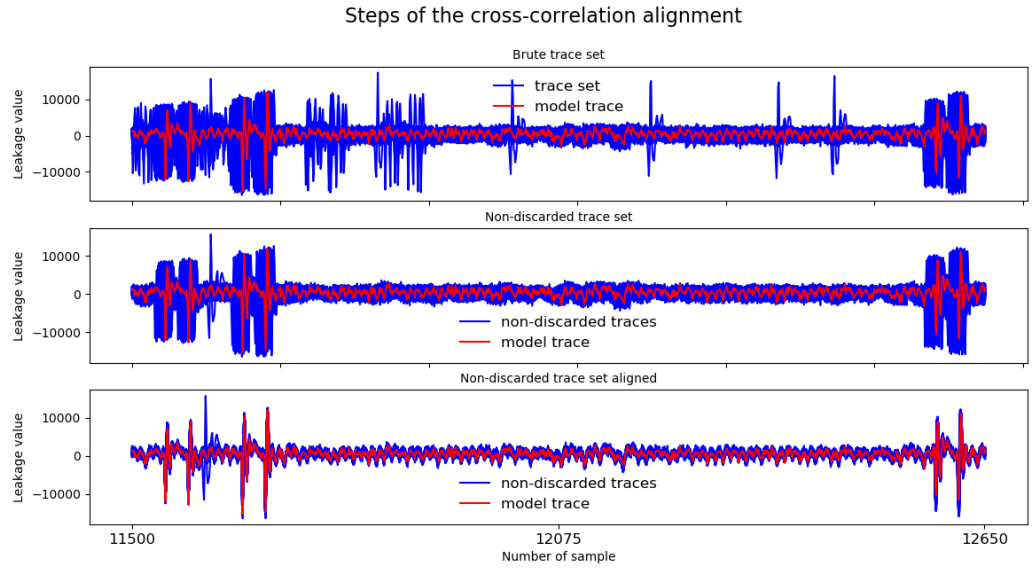

Figure 4.4: Steps of the cross-correlation alignment algorithm. The upper graph shows the initial trace set, the middle one shows the remaining traces after discarding and the lower one shows the alignment of those traces

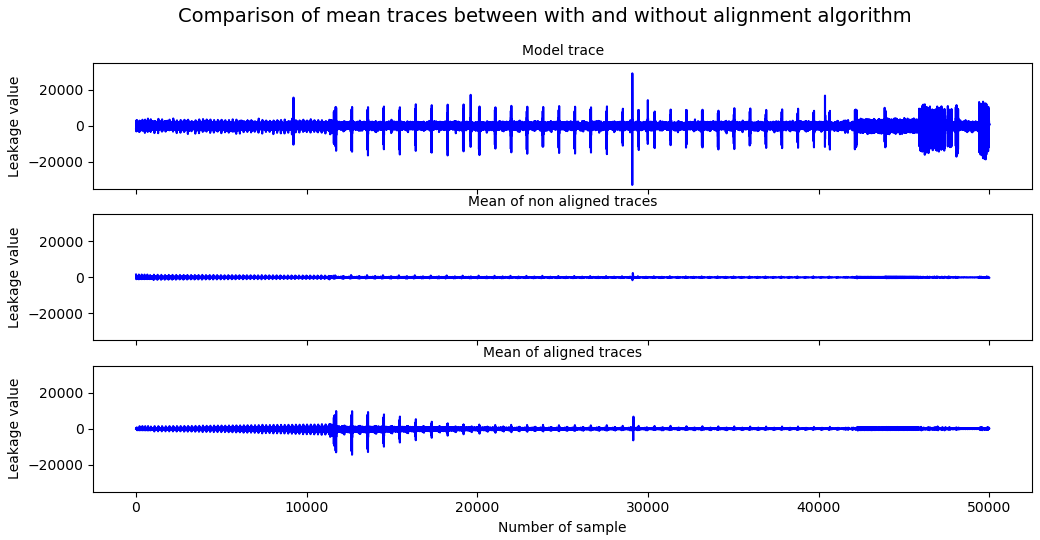

Figure 4.5: The upper graph corresponds to the reference trace, the graph in the middle is the mean of 50000 traces without the alignment algorithm and the lower one is the mean of the traces obtained by the alignment algorithm.

The main drawbacks of this algorithm are its effective windows as highlighted in the Figure 4.5. In this Figure, it can be observed that the mean of the aligned traces has a similar leakage value to the model trace only in small range roughly from sample number 11000 to sample number 13000. Outside this range, the mean of the aligned traces falls off with refer to the model trace. A dynamic alignment algorithm, such as the elastic alignment algorithm from [10], could counteract this side effect but would require a lot more computing power while one could translate the effective window depending on the point of interest targeted. One can also notice in Figure 4.5 that the mean of non aligned traces leads in barely indistinguishable signal. This highlights once again the need of the trace alignment algorithm. All following experiments are made with traces aligned by the Algorithm 4.

4.2.2 Search of the POI

The focus being on the inner hash, the second group in the Figure 4.3 corresponding to the second SHA256 computation will be targeted in a window of 50000 samples. In fact, to reduce the jitter, the trigger is performed just before this execution of SHA256.

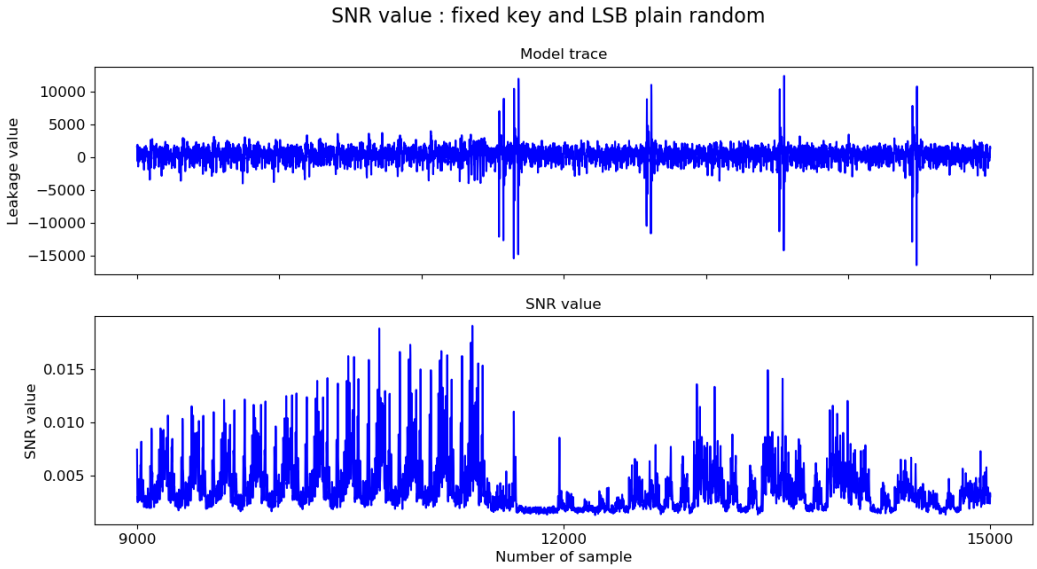

Figure 4.6: SNR value computed with a fixed key and only the least significant byte of the plain random. The model trace is shown as a reference

The SNR statistical tool is used to localize a precise state, such as a modular addition or the bitwise AND between 2 values, or any bijective state related to this

particular state in time. The SNR is said to converge at a state or its bijection when a particular peak is standing out of the noise level. The POI of a particular state is the points of convergence of this particular state.

The first analysis that is performed consists in evaluating the SNR of the SHA256 algorithm with fixed input states and a fixed plaintext unless the least significant byte which is random. This gives an idea about where the first input word is manipulated in the SHA256 structure. The result of this evaluation is shown in Figure 4.6. Since only a byte is random, a lot of bijections states exist due to the SHA256 structure. This allows to guess the different parts of the SHA256 algorithm. Effectively one could guess that the moment where there is only noise in the SNR is the moment where the initial states are loaded since they are not related at all with the first word message. The pattern that can be observed on the left of these loading samples corresponds to the extension algorithm of the message present at the beginning of SHA256 and the pattern that can be noticed on the right of these samples corresponds to the rounds of transformation of SHA256. The difference of amplitude of the SNR on the sides is explained by the end of the effective window of the trace alignment algorithm. The difference of amplitude at the beginning of the rounds of SHA256 is related to the fact that the dependence of the different states of the algorithm with the first LSB of the plaintext increases with the rounds.

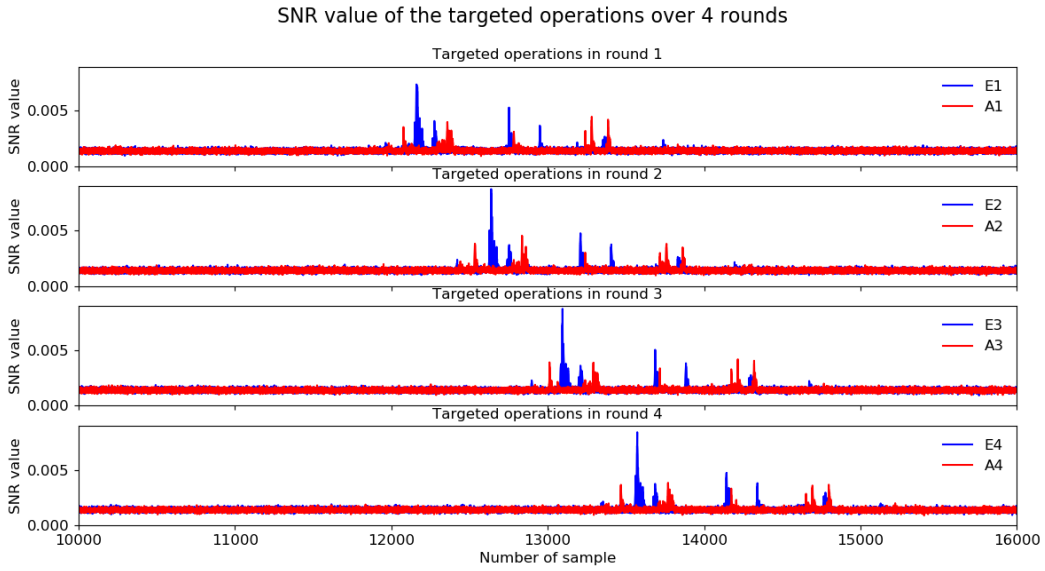

Figure 4.7: SNR value of targeted operations needed by the divide and conquer strategy from round one to round four.

Then the second analysis made consists in targeting the modular addition operations through the four rounds that are necessary to perform the divide and conquer attack strategy (see Section 2.7.1). To suppress a maximum of bijective dependence between two states and thus obtain only the aimed state, the SNR is performed on a set of traces with the initial states and the plaintext random. The SNR for each targeted operation is shown in Figure 4.7. The same shape can be observed from one round to the other meaning that each round leaks more or less the same amount of information. Thanks to this test, it is possible to find the POI to perform the divide and conquer attack. One important thing to note is that the modular addition updating the value of the state A delivers less information than the one actuating the value of the state E. Another remark is that the leftmost POI found are due to bijective dependence.

After that, the same methodology is applied to find the POI of the extend and prune strategy by targeting modular additions and bitwise AND operations through the first and the second round (as described in Section 2.7.2). However, the eight bits SNR does not converge at all while targeting a bitwise AND operation using the same number of traces as the one used when computing the SNR of the modular addition. This non convergence can be observed in the third sub figure of the Figure 4.8. This effect can be explained by the fact that unlike the modular addition the output of the bitwise AND is not uniform. For example, as it is highlighted in the Table 4.1 the bitwise AND between two random variables of two bits leads to a probability of 9/16 to output 00, 3/16 to output 01 and 10 and 1/16 to output 11.

In 1	In 2	Out
00	00	00
00	01	00
00	10	00
00	11	00
01	00	00
01	01	01
01	10	00
01	11	01

In 1	In 2	Out
10	00	00
10	01	00
10	10	10
10	11	10
11	00	00
11	01	01
11	10	10
11	11	11

Table 4.1: Output of two bits bitwise AND between two variables

This non uniformity induces a difference of accuracy in the estimation of the different classes needed in the computation of the SNR. Moreover, this bias is increasing with the size of the field in which the SNR is computed. The field is considered here as the bit size of the targeted state taken into account by the SNR. Indeed, the worst probability of the output of the bitwise AND operation in case of

random inputs is given by 2^{-2n}, n being the size of the field. Hence, it is needed to increase the number of traces used to compute the SNR in order to obtain the POI for an eight bits field. Decrease the size of the field can also lead to a convergence of an SNR since there are fewer classes to estimate and the non uniformity is less accentuated. This effect can be observed in the Figure 4.8. In this Figure, the SNR is computed using 200 000 traces and targets the bitwise AND operation between A1 and B1 which is happening at the beginning of the round two. The SNR in the two and four bits fields converge while it does not converge in the eight bits field. One can also notice that the noise floor is higher in the four bits field than the one in the 2 bits. The spikes found in the two first sub figures can be used as POI.

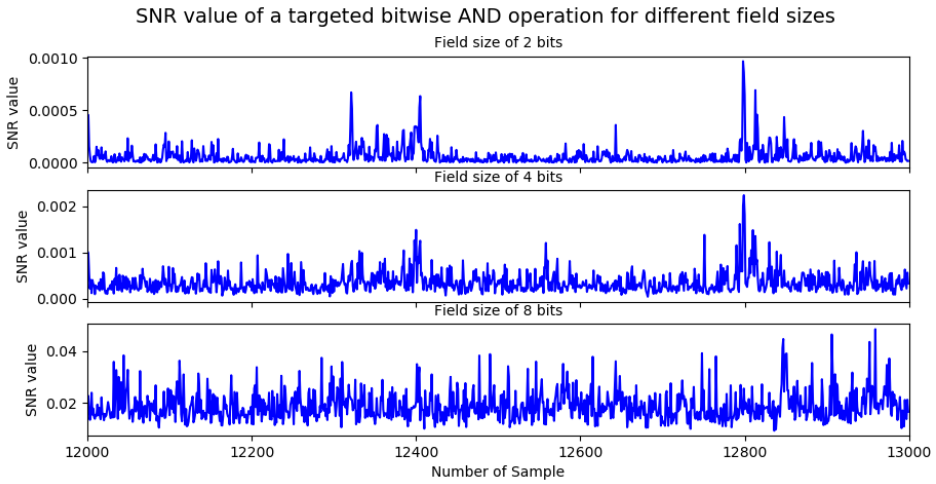

Figure 4.8: SNR value computed with 200 000 traces of a targeted AND operation for the field sizes 2,4 and 8 bits.

The other POI of this strategy can be searched using the same method and can be observed in the Figure 4.9. This Figure aims to present the POI of the eight SCA that are necessary to recover the secret. One can notice that the convergence of the different SNR reaches different values related to the information recovered from the traces for this specific operation.

4.3 Results of the attacks

Attacks can now be mounted based on the knowledge of the POI. Four different SCA will be experienced and if succeeded compared to each other in terms of number of traces needed to recover the initial states of the SHA256 execution. The number of traces gives a value of the attack efficiency while being also linked to

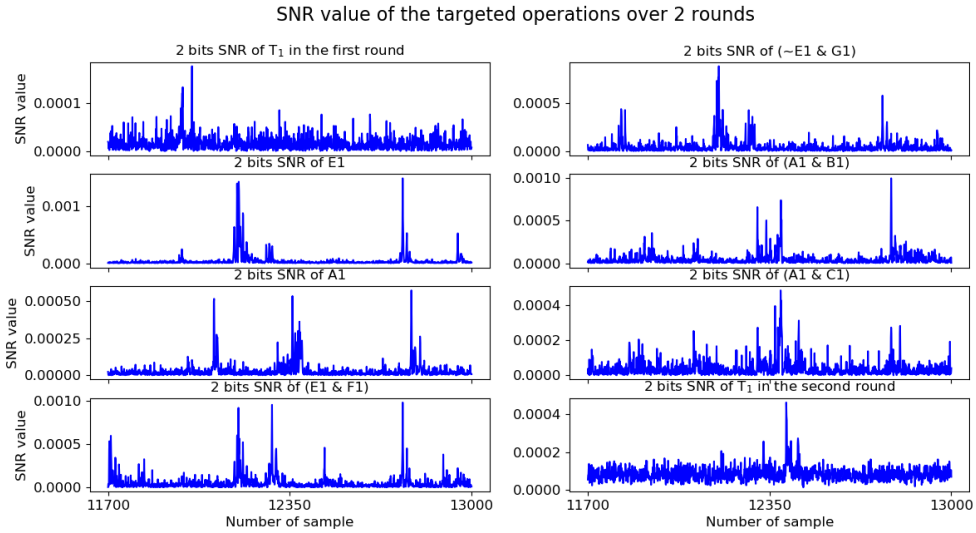

Figure 4.9: Two bits SNR value of targeted operations needed by the extend and prune strategy over round one and two.

the malicious time needed in possession of the Solo key to be broken. In order to fix a reasonable amount of this malicious time, a limit of 200 000 traces is fixed for the online attack representing more or less a day and half of time. First, the different SCA will be described. Then, the results of the divide and conquer strategy will be exposed and interpreted. This strategy is presented first as it is the less tricky one since it relies only on modular addition operation and the partial secrets are independent of each other. Finally, the results of the extend and prune strategy will be shown. The results of the attacks presented in this section are only done to recover the inner hash. However, since the outer hash relies on the same cryptographic algorithm computed on the same MCU, there is no reason for the attacks of the extend and prune strategy to not obtain similar results in the recovery of the outer hash.

4.3.1 Side channel attacks description

As the micro controller present in the Solo key processes 32 bits words, partial CPA or partial template attack is needed to reduce the computational cost of the attack. The idea of a partial attack is to recover the secret of 32 bits by splitting it into for example four partial attacks, each targeting eight bits of the secret. Indeed, performing the attacks this way reduces the computational cost from 2^{32} hypotheses to be assessed by the attacks to $4 * 2^8$ hypotheses. The main drawback of this technique is that it assimilates the bits that are not used in the attack to noise

as the leakage depends on every bit of the secret. Moreover, particular attention needs to be given if an attacked block has a dependence on the other blocks. For example, in modular addition the carry out of the previous assessed blocks needs to be taken into account in the next partial attack. In the context of this work, eight bits partial attacks on a modular addition operation and bitwise AND operation will be used. In the case of modular addition operation, the carry in between the different attacked blocks, since it propagates from the LSB to the MSB, will be considered as found by the previous successful partial attack starting with the LSB partial attack which requires no carry. This assumption will be assessed later in the next section.

The four different SCA used in this framework are described in the following part. The value of the targeted operation can be, depending on the strategy, the result of the modular addition between a secret value byte, a known byte related to the plaintext byte and a potential carry out coming from the LSB addition or the result of the bitwise AND between a secret value byte and a known byte related to the plaintext byte. In the case of the profiling phase required for some SCA, the secret value byte is random and known.

1. A non profiled CPA. The profile used in such an attack is the Hamming weight of the value of the targeted operation. The advantage of this attack is that it does not require a model computation however it makes the strong assumption that the targeted device leaks according to the Hamming weight. Another advantage of the non profiled CPA is that the POI are not necessary to launch the attack even if it allows lower computation time and avoid potential higher bijective spikes that are linked to another hypothesis key value.

2. A linear regression (LR) profiled CPA [18]. This attack works with a precomputed profile that aims to capture any linear relation between the leakage value and its corresponding value of the targeted operation. The linear regression model is computed such that it fits a coefficient vector to minimize the residual sum of squares between the value of the targeted operation, and its leakage value, predicted by the linear approximation. This LR profiled CPA requires a set of profiling traces and the knowledge of the POI but improves the modeling of the leakage compared to the non profiled CPA and relaxes the assumption about the way the targeted device is leaking. The POI value used corresponds to one of the SNR spikes that has been observed in Figure 4.7 and in Figure 4.9.

3. An exhaustive CPA with first-order moments [19]. This attack also works with a precomputed profile but this one aims to capture any relation between

the leakage and its corresponding value of the targeted operation. The modeling of leakage is performed through the first order statistical moment, meaning that each trace present in the profiling traces set is first sorted in the group of its corresponding value of targeted operation and then the mean of each group is computed and used as profile. This attack also requires POI which will be the same as the ones used in the LR CPA. The advantage of this method is that it perfectly models the leakage with no assumption on it. However, it may require more profiling traces to converge than the LR profiled CPA.

4. A univariate template attack (see Section 2.6.2). This attack as the LR and exhaustive CPA also requires POI knowledge.

The main motivation to perform all three different CPA is that, in the case of modular addition, the correlation between the modeled output of the targeted operation and the leakage, that can be observed in Figure 4.10, is really similar in terms of shape and amplitude in all three cases. Thanks to this figure, one can conclude that the Solo key is leaking according to the Hamming weight model or some relation really close to it. The three models will thus be compared in terms of needed traces to recover the secret.

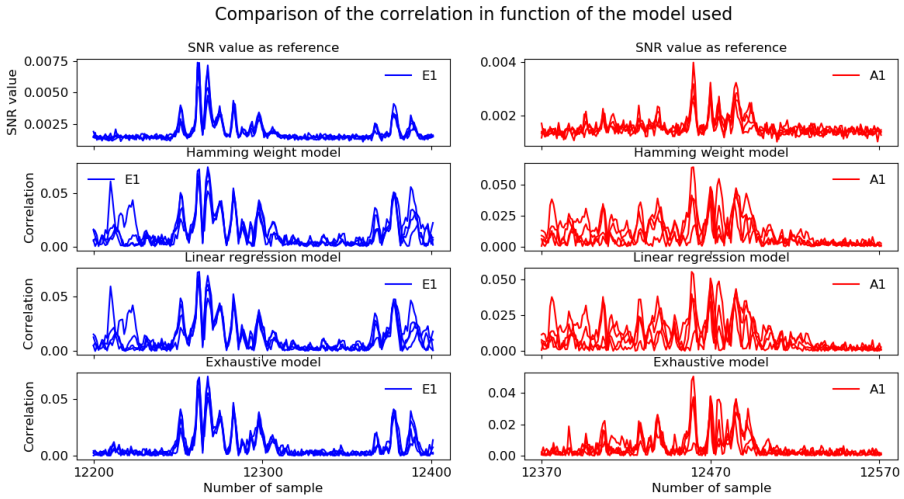

Figure 4.10: Comparison of the correlation coefficient for the POI of the first round for all three models in the case of the divide and conquer strategy. On the left the output of aimed modular addition E1 and on the right, A1. The corresponding SNR value of the operation is given as reference.

4.3.2 Divide and conquer strategy : results and comparison of the SCA

First, the results of the CPA according to the Hamming weight profile model are shown in the Figure 4.11. The eight secrets that have to be found to recompute the initial states of the SHA256 algorithm are separated in four partial CPA, one for each byte of the 32 bits secret. The graphs show in blue the evolution of the correlation coefficient of all hypothesis tested in regard to the number of traces. The correct hypothesis is highlighted in red. One can observe that in less than 50 000 traces all secret bytes are found. Another remark is that it seems more difficult to recover the secret corresponding to the A state of SHA256 than the one corresponding to the E state. This is linked to the fact that the peak of the SNR corresponding to the A state of SHA256 is more or less 3 times smaller than the peak corresponding to the E state.

Secondly, the results corresponding to the linear regression are presented the same way as the ones of the non profiled CPA and can be observed in Figure 4.12. The profiled model is obtained with 200 000 traces set with random initial states and random plaintext. By analyzing the results, one can conclude that they are really close to the ones obtained by the non profiled CPA and that all secrets are also recovered in less than 50 000 traces. The same explanation regarding the poorer performance to recover the secret of the A state of SHA256 as the one given in the non profiled CPA can be used here.

Thirdly, the results corresponding to the exhaustive CPA can be seen in Figure 4.13 and are really less convincing than the ones in the other CPA. The profiling phase is performed with the same traces set as the LR CPA. In one way, the results corresponding to the secret related to the E state are better than in the previous attack, especially for the E2 secret where the correlation of the correct hypothesis detaches more than in the previous attacks. In another way, the results corresponding to the secret related to the A state are far poorer than the previous attacks. All the secrets cannot be recovered in less than 50 000 traces in the exhaustive profiled CPA. This can be explained by the fact that a lesser SNR requires more traces to achieve a great profiling and to succeed the attack phase.

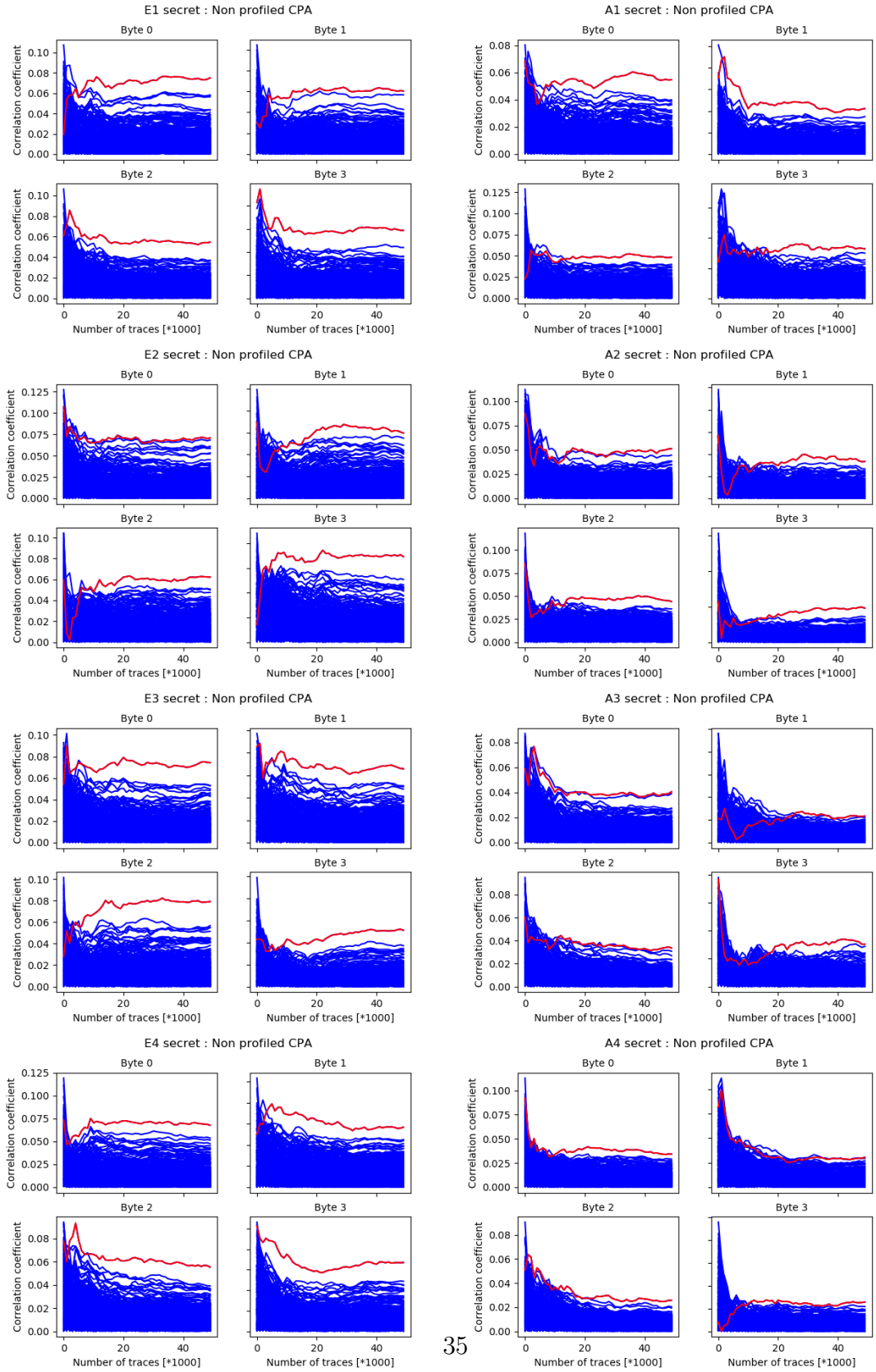

Figure 4.11: Correlation coefficients of all hypothesis of all partial secrets obtained by the non profiled CPA through the divide and conquer strategy in function of the number of traces. The correct hypothesis is represented in red.

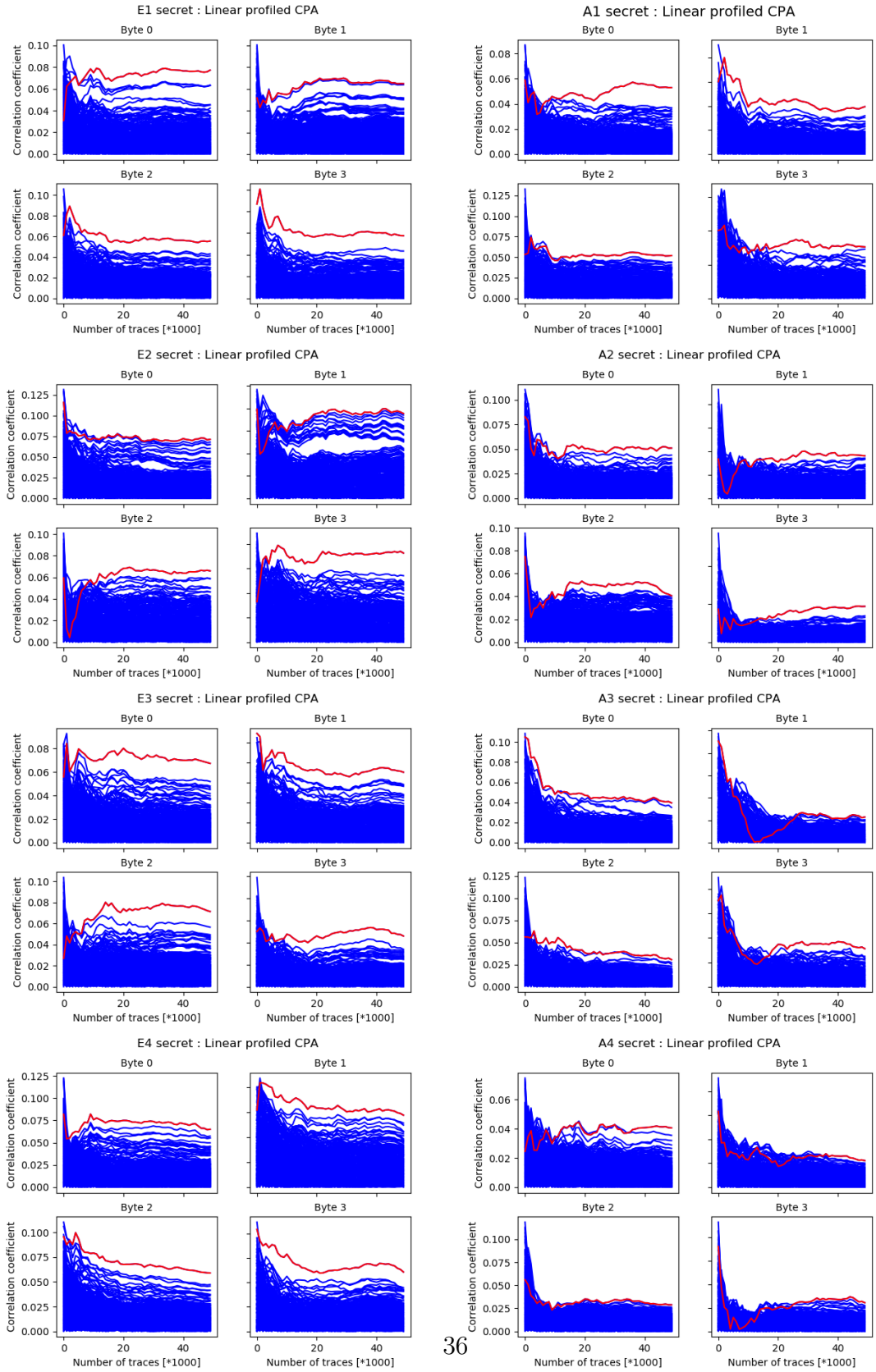

Figure 4.12: Correlation coefficients of all hypotheses of all partial secrets obtained by the linear profiled CPA through the divide and conquer strategy in function of the number of traces. The correct hypothesis is represented in red.

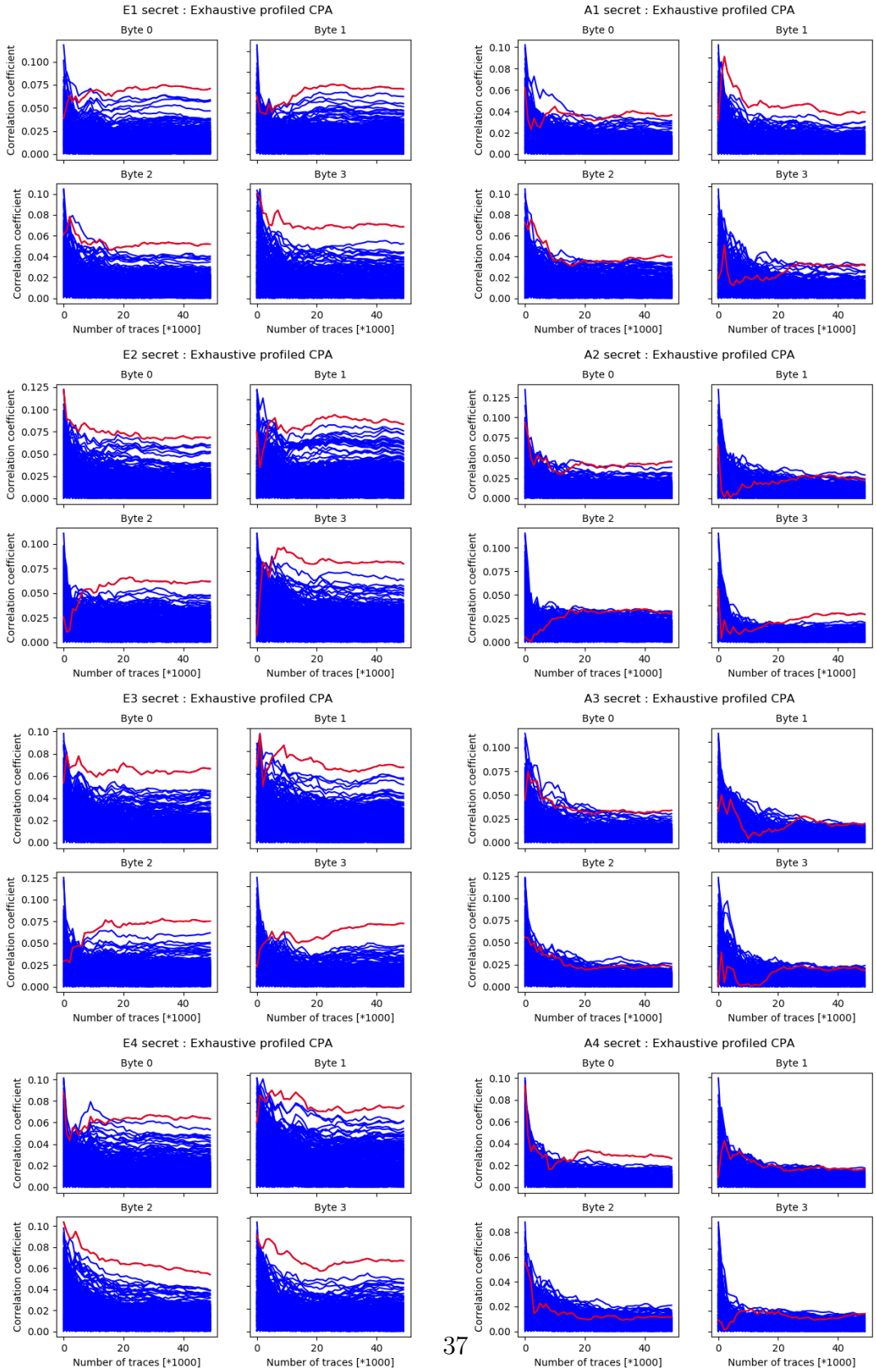

Figure 4.13: Correlation coefficients of all hypotheses of all partial secrets obtained by the exhaustive profiled CPA through the divide and conquer strategy in function of the number of traces. The correct hypothesis is represented in red.

In order to compare the different attacks, the rank estimations of the initial states of each of them have been computed in function of the number of traces used in the Figure 4.14. With no surprise the Hamming weight model and the LR model give significantly better results than the exhaustive model. Since they lead to the similar results, the non profiled CPA should be preferred in regard to the LR CPA as it does not require any profiling phase.

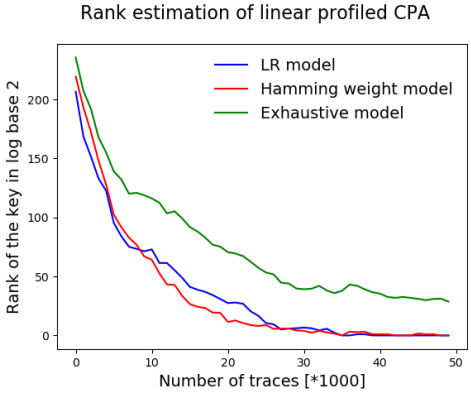

Figure 4.14: Rank estimation of the initial states in function of number of traces used for the 3 leakage model of the CPA.

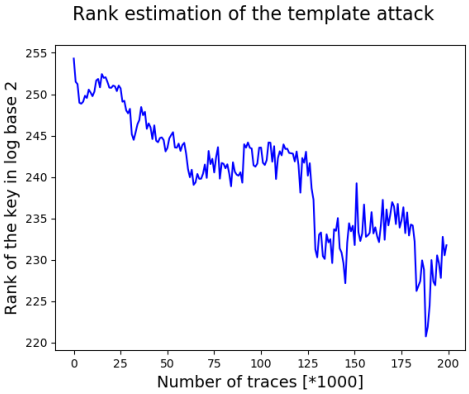

Figure 4.15: Rank estimation of the initial states in function of the number of traces used in the template attack.

The performance of the template attack is shown in the Figure 4.15. As it can be observed its performance is much poorer than the 3 previous attacks as the template attack does not allow to recover the initial states of SHA256 in an acceptable enumeration range even after 200 000 traces. This is likely to be due to an insufficient number of traces during the profiling phase as this one was fed with the same amount of traces as the profiling phase of the profiled CPA. It could also be due to the remaining jitter that can be left even after the trace alignment algorithm.

The Figure 4.16 aims to assess the assumption that the carry is found by the attack made on the LSB blocks. Indeed, so far the carry in propagation was taken as a known value while performing the attacks. This leads to overestimate the previous attacks made. Correctly estimating this assumption requires an infeasible memory requirement as it would need eight, one for each secret, tables of 2^{32} probabilities (128 GB of RAM) as input of the rank estimation algorithm. However, one could take into account the carry propagation between the first and the second LSB and between the third and the fourth LSB. Indeed, it will relax the initial hypothesis while still being in an acceptable range of memory size needed and

analyzes the impact on the attacks. This is what is shown in the Figure 4.16. In this Figure, it can be noticed that the curve that takes the carry in propagation into account is slightly shifted to the right meaning that the previous attacks were effectively overestimated. Nevertheless, this overestimation is quite small and does not change the final trend of the rank estimation of the secret. The attack used to perform this figure is the non profiled CPA.

Finally, the rank estimation of another random initial states of the SHA256 algorithm is also performed in Figure 4.17. The 2 curves really follow the same trend which allow to somehow validate the attack.

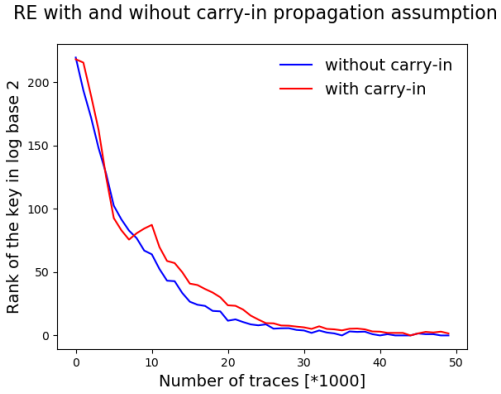

Figure 4.16: Rank estimation of the input states in function of the number of traces used computed with the non profiled CPA with and without the carry in assumption.

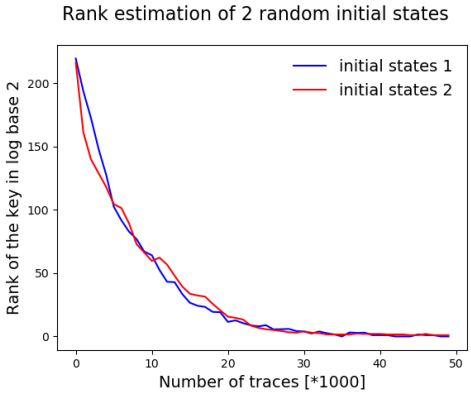

Figure 4.17: Rank estimation of 2 random input states in function of the number of traces used computed with the non profiled CPA.

In conclusion, the inner hash of the HMAC-SHA256 scheme implemented in the Solo key can be broken thanks to the non profiled CPA applied to the divide and conquer strategy. This attack requires about 200 000 traces to succeed without conquer phase. Indeed, four sets of 50 000 traces (one for each round attacked) are necessary to complete the attack. The conquer phase is not used here because the recovery of k0 is the first step of breaking the HMAC-SHA256 construction and the attacker, in real conditions, has not the knowledge of the intermediate state at the output of the inner hash. Moreover, the second step which consists in finding the initial states of the outer hash through the extend and prune strategy needs a strong level of confidence in k0 in order to be realizable. Effectively, the input messages of the outer hash, which relies on k0 and the input messages of the inner hash, need to be guessed right since the whole attack of the outer hash will depend on these input messages.

4.3.3 Extend and prune strategy : results of the SCA

First of all, an additional preliminary hypothesis is made due to the specificity of the extend and prune strategy. This strong hypothesis consists in assuming that the previous secrets needed to perform this stage of the strategy are well guessed. This hypothesis will be evaluated in case of successfully recovering the initial states of SHA256 with an attack.

Secondly, only the non profiled CPA and the linear profiled CPA will be attempted. Indeed, in the light of their performance during the divide and conquer strategy, they are the attacks with the greater potential of success. Moreover, it avoids the complexity of computing the exhaustive profile of the bitwise AND operation since this complexity is related to the convergence of the eight bits SNR of this operation.

Next, the results of the non profiled CPA are shown in the Figure 4.18. These results are quite mixed. D0 and T2 are the only partial secrets that are completely recovered. These two attacks targets the same states of SHA256 than the attacks of the divide and conquer strategy. The other results are less convincing certainly due to a lack of traces. In the case of the non profiled CPA against a bitwise AND operation (E0, F0, A0, B0) the necessity of increasing the number of traces can be due to its intrinsic nature. Indeed as highlighted in [5], if one takes a key hypothesis K_1 and another $K_2 = K_1 \wedge 1$ then $HW(K_1 \& M_i) = HW(K_2 \& M_i)$ for 50% of the random input messages M_i and for the other 50% M_i the Hamming weight will only differ by 1. The hypotheses will thus have similar correlation coefficients. In the remaining cases, α and B0, the need of more traces could be explained more simply by the fact that the operation targeted in both cases, the updating of T_1, leaks less than the operations that update the states A and E. This can be seen in the Figure 4.9 where the peaks of SNR related to the update of T_1 are smaller than ones related to the SNR of E1 and A1.

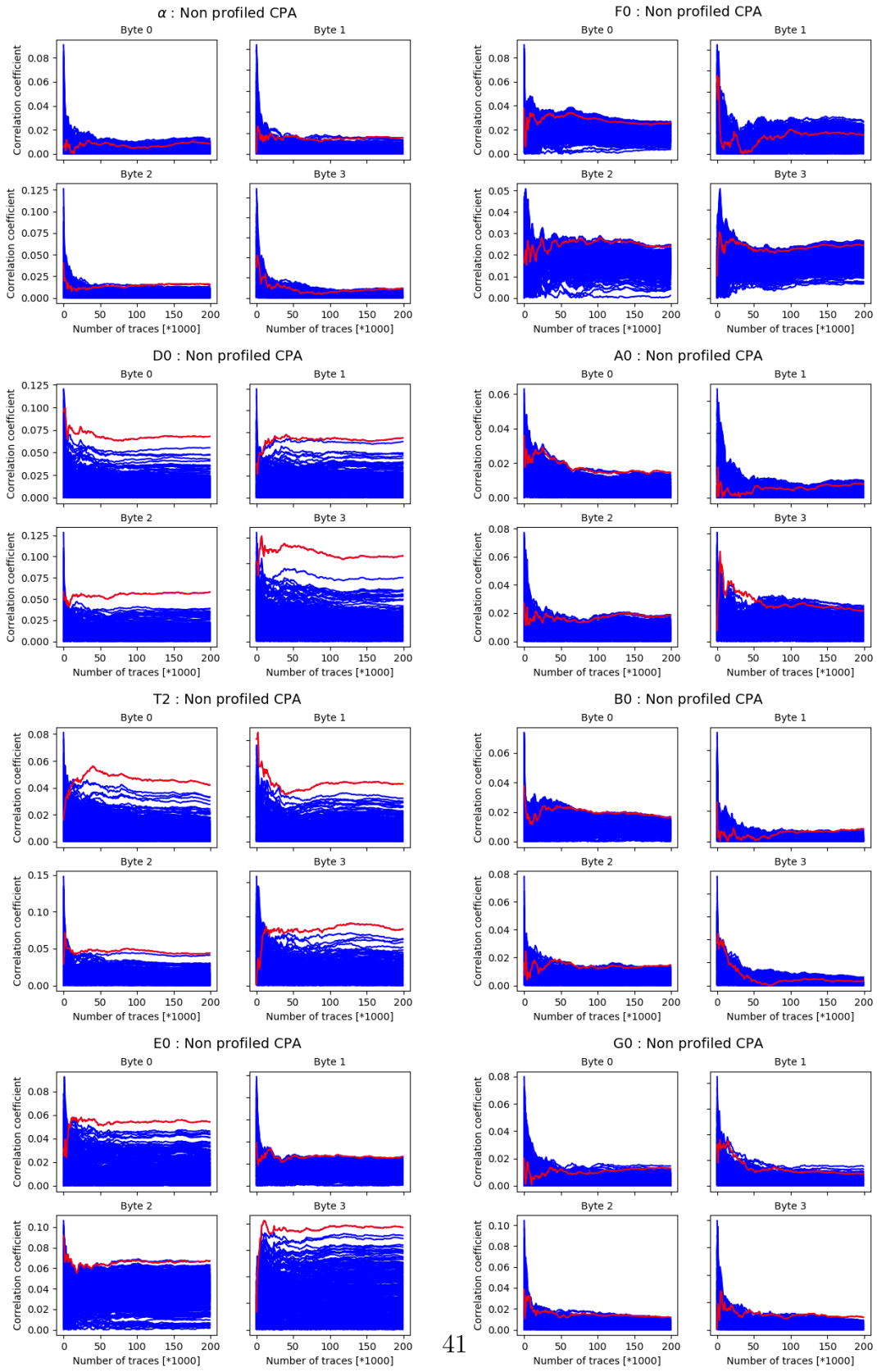

Figure 4.18: Correlation coefficients of all hypothesis of all partial secrets obtained by the non profiled CPA through the extend and prune strategy in function of the number of traces. The correct hypothesis is represented in red.

Finally, the result of the linear profiled CPA can be observed in the Figure 4.19. The profiling phase is made with 200 000 traces with the same set of traces as the one used during the divide and conquer strategy. These results are really comparable to the ones obtained with the non profiled CPA. This is explained by a quite accurate profiling model that is close to the Hamming weight as already highlighted above. The same explanation as the one provided for the non profiled CPA is therefore applicable for the linear profiled CPA. Only one noticeable difference can be observed concerning G0 which is well recovered here. However, this is not surprising since the peak of the SNR of T_1 in the second round, even if it is smaller, is not very far from the one concerning A1.

To conclude, the different attacks through the extend and prune strategy do not allow to recover the entire secret using 200 000 traces. The absence of success is likely to be due to a lack of traces resulting from small SNR spikes and the nature of the bitwise AND operation. The failure of this strategy implies a failure to break the entire HMAC-SHA256 construction since k1 cannot be found. And so, a failure of recovering the private key of the Solo key.

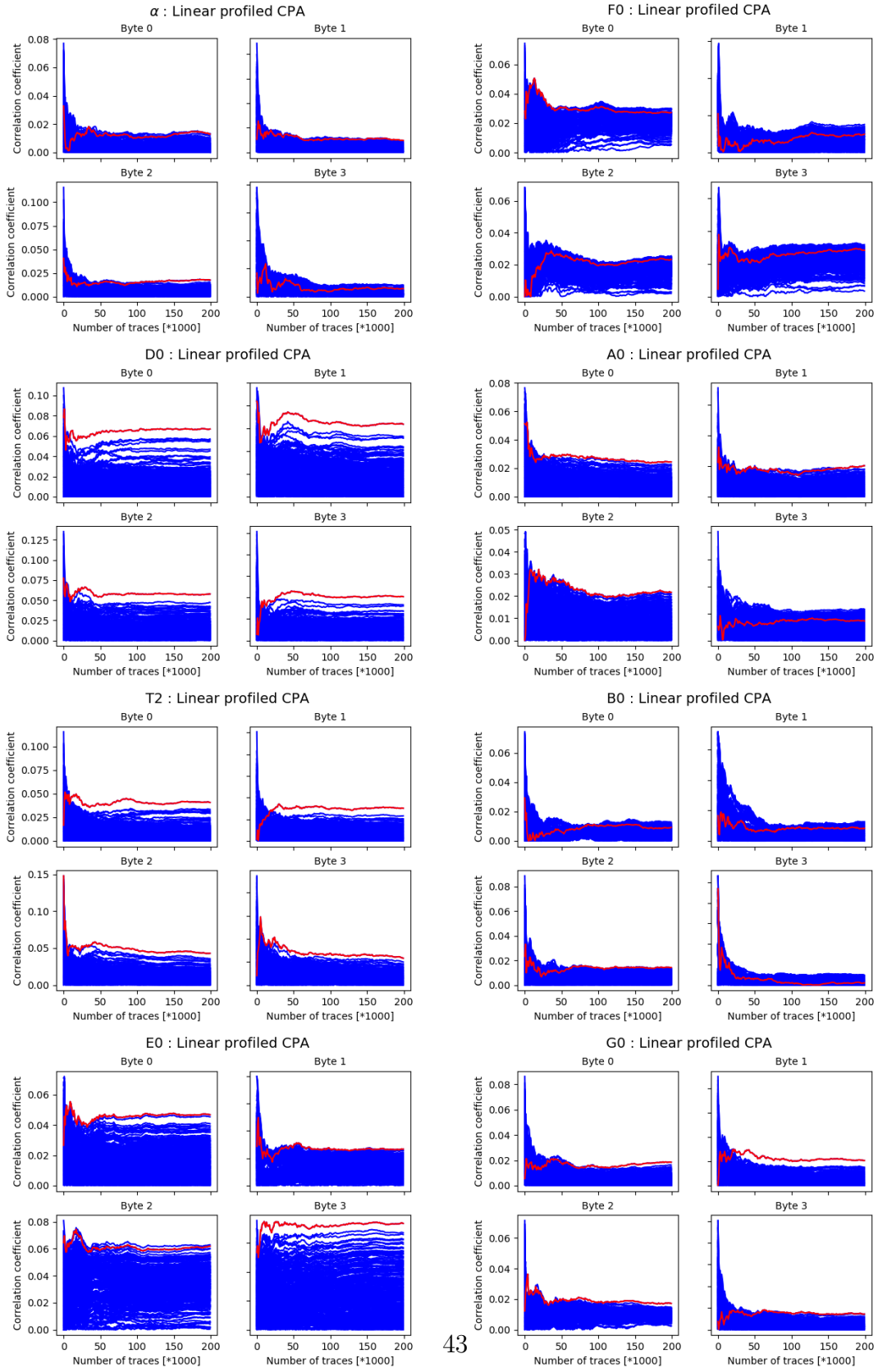

Figure 4.19: Correlation coefficients of all hypothesis of all partial secrets obtained by the linear profiled CPA through the extend and prune strategy in function of the number of traces. The correct hypothesis is represented in red.

Chapter 5

Conclusion

The goal of this master thesis was to evaluate the resistance of the open source FIDO U2F protocol implemented in the Solo key. To realize this objective, the protocol was first investigated to find a surface of attack. From this stage it can be concluded that breaking the HMAC-SHA256 present in the protocol allows the adversary to perform authentication as the rightful user. Then the measurement setup needed to acquire the leakage traces necessary to mount the attack was described. After that, the analysis of the leakage was performed, thanks to a trace alignment algorithm, to recover the points of interest hidden in this leakage for two strategies, the divide and conquer one and the extend and prune one. Finally, different attacks were performed and compared to each other. It results that in the case of the divide and conquer strategy, the simplest attack corresponding to a non profiled CPA is the best and is able to recover the initial states of the inner hash of the HMAC-SHA256 algorithm in 200 000 traces. The results of the extend and prune strategy, the only strategy that can be employed to recover the initial states of the outer hash of HMAC-SHA256, are not satisfying likely due to a lack of traces used to perform the attacks. Hence, the FIDO U2F protocol implemented in the Solo key was not broken in the context of this master thesis.

However, a lot of improvements can be managed. The needs of increasing the number of traces and so the malicious time in possession of the targeted Solo key could be reduced by improving the attacks. Indeed, better synchronization of traces (through the elastic alignment algorithm [10] for example), better measurement setup or preprocessing the traces such as frequency filtering will hopefully lead to a reduction of the number of traces required by the attacks. The use of more powerful attacks or increasing the number of traces during the profiling phase could also go in this direction.

Finally, due to the potential feasibility of the described attacks, some countermeasures could be considered. Those countermeasures can take multiple aspects.

For example, it can be an implementation of a masked version the SHA256 algorithm [4] which will only involve more time to perform the authentication protocol, but will require more to an infeasible number of traces to break it. Indeed, using only two shares will increase the number of traces needed to break the inner hash to 200 000 traces squared. Or more simply, protecting the micro controller with an electromagnetic shield requiring that the attacker leaves fingerprints on the targeted token while only increasing the cost of the device.

Bibliography

[1] Microsoft Corporation. Password-less protection. `https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE2KEup`.

[2] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. *CRYPTO '96, Advances in Cryptology*, pages 104–113, 1996.

[3] Colin O'Flynn. Min()imum failure: Emfi attacks against usb stacks. *WOOT'19: Proceedings of the 13th USENIX Conference on Offensive Technologies*, page 15, 2019.

[4] Robert McEvoy, Michael Tunstall, Colin C. Murphy, and William P. Marnane. Differential power analysis of hmac based on sha-2, and countermeasures. *Information Security Applications: 8th International Workshop, WISA 2007, 2007, Revised Selected Papers*, pages 317–332, 2007.

[5] Sonia Belaïd, Luk Bettale, Emmanuelle Dottax, Laurie Genelle, and Franck Rondepierre. Differential power analysis of hmac sha-2 in the hamming weight model. *2013 International Conference on Security and Cryptography (SECRYPT)*, 2013.

[6] Pankaj Rohatgi and Mark Marson. Nsa suite b crypto, keys, and side channel attacks - rsa conference 2013. `https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE2KEup`.

[7] Catherine H Gebotys, Brian A White, and Edgar Mateos. Preaveraging and carry propagate approaches to side-channel analysis of hmac-sha256. *ACM Transactions on Embedded Computing Systems - Article No.: 4*, 2016.

[8] David Oswald, Bastian Richter, and Christof Paar. Side-channel attacks on the yubikey 2 one-time password generator. *Research in Attacks, Intrusions, and Defenses: 16th International Symposium, RAID 2013, Rodney Bay, St. Lucia, October 23-25, 2013. Proceedings*, pages 204–222, 2013.

[9] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying hash functions for message authentication. *Advances in Cryptology, Proceedings Crypto '96*, pages 1–15, 1996.

[10] Marc F Witteman Jasper G. J. van Woudenberg and Bram Bakker. Improving differential power analysis by elastic alignment. *CT-RSA'11: Proceedings of the 11th international conference on Topics in cryptology*, pages 104–119, 2011.

[11] Cezary Glowacz, Vincent Grosso, Joachim Schüth Romain Poussier, and François-Xavier Standaert. Simpler and more efficient rank estimation for side-channel security assessment. *FSE 2015: Fast Software Encryption*, pages 117–129, 2015.

[12] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. *CRYPTO '99, Advances in Cryptology*, pages 388–397, 1999.

[13] Sampath Srinivas, Dirk Balfanz, Eric Tiffany, and Alexei Czeskis. Universal 2nd factor (u2f) overview. `https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-overview-v1.2-ps-20170411.html`.

[14] Dirk Balfanz, Jakob Ehrensvard, and Juan Lang. Fido u2f raw message formats. `https://fidoalliance.org/specs/fido-u2f-v1.2-ps-20170411/fido-u2f-raw-message-formats-v1.2-ps-20170411.html`.

[15] Mehmet Adalier and Antara Teknik. Efficient and secure elliptic curve cryptography implementation of curve p-256. *Workshop on Elliptic Curve Cryptography Standards*, 2015.

[16] SoloKeys. Solokeys github repository. `https://github.com/solokeys`.

[17] Pierre-Alain Fouque, Gaëtan Leurent, Denis Réal, and Frédéric Valette. Practical electromagnetic template attack on hmac. *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 60–80, 2009.

[18] Werner Schindler, Kerstin Lemke, and Christof Paar. A stochastic model for differential side channel cryptanalysis. *Cryptographic Hardware and Embedded Systems - CHES 2005, 7th International Workshop*, pages 30–46, 2005.

[19] Amir Moradi and François Xavier Standaert. Moments-correlating dpa. *TIS '16: Proceedings of the 2016 ACM Workshop on Theory of Implementation Security*, pages 5–15, 2016.

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