POLITECNICO DI TORINO

Master degree course in Electronic engineering

Master Degree Thesis

# Side-channel leakage assessment methodology applied to Post Quantum Cryptography algorithm


**Supervisor:**
prof. Guido Masera

**Co-supervisor:**
prof. Maurizio Martina

**Candidate:**
Lorenzo Cecchetti

Accademic year 2021-2022

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In the last decades the most important factor that has changed our lifestyle is the technological evolution. In the electronic and computer science field a huge step forward will happen with the rise of quantum computers that with their exponential increase in the computation capabilities are going to break all the cryptography algorithms that are commonly used. That is why the NIST (*National Institute of Standards and Technology*) has launched a competition open to everyone whose goal is to find new standards and algorithms that could be used in the future era of quantum computers. The focus of this competition are algorithms on Public Key Cryptography (asymmetric cryptography) and on the digital signature problem. The most promising solutions for PKC are:

- McEliece which is a code-based cryptosystem;
- NTRU that instead is a lattice-based cryptosystem.

Code-based cryptosystems have the Public Key (PK) and Secret Key (SK) based on the Encoding/Decoding matrices of an Error Correcting Code and in McEliece cryptosystems Goppa Codes are employed. McEliece has been proved to be secure against current quantum computer attacks since it relies on a problem to be solved by the attacker which is different from the ones used nowadays like the factorization of a very big number. In fact the property that this cryptosystem has is that it is impossible to recover the private code from the public key because the attacker does not know any characteristic of it.
The main problem of McEliece is the huge dimension of the public and private keys which limits its adoption in most of the systems. Low Complexity variants has been proposed in the NIST competition, among them LEDAcrypt/BIKE are code based proposals that adopt LDPC/MDPC codes. LEDAcrypt is composed by two algorithms: a key encapsulation mechanism for symmetric systems called LEDAkem and an algorithm for asymmetric cryptosystems named LEDApkc. The advantage brought by LEDA with the respect of McEliece is the use of Low Density Parity Check Codes

(LDPC) which makes the memory storage of the Secret Key (SK) much more feasible. Moreover, an efficient hardware implementation has been found to compute the product between the Secret Key (SK) matrix which is both sparse and Quasi-Cyclic (QC) and the message vector. The hardware Implementation of LEDAcrypt appeared to be the most efficient among the code based ones due to the very low requirement of hardware resources that are needed, but its security towards side-channel attacks has to be addressed. In the present work its robustness against a specific typology of attacks called Power Analysis Attacks has been tested.
Those attacks tries to recover the private key used during the encryption or decryption of the message by exploiting the device power consumption over the time. The three most important typologies are:

- Simple Power Analysis (SPA);
- Differential Power Analysis (DPA);
- Correlation Power Analysis (CPA).

Simple power analysis is a technique that aims at finding recurrent patterns in a single power trace in order to find the secret key used, while DPA and CPA requires the use of multiple traces to be performed. CPA computes the correlation between the measured power traces and a predicted power consumption value and finds the private key by searching the highest correlation value obtained. Instead DPA divides the collected power traces into subsets based on some key binary selection functions, then the traces belonging to the same subset are averaged and the trace obtained is subtracted to the one computed on the other subset of the same selection function. The correct key is the one where the differential trace assumes non-null values (has some spikes).
The present work proposes a methodology that could be used by designers that work in the cryptographic field to test in the design phase if the developed ASIC architecture is safe from power analysis attacks. Being quite general, this method can be inserted as an additional step into the ASIC design flow and consists in performing a simulated power analysis attack on the device post-synthesis netlist. The simulated power traces are obtained using two tools: QuestaSim by Mentor Graphics and PrimeTime by Synopsys. The former is used to produce the Value Change Dump file of the simulation in which the netlist receives in input a private key and a message and has to produce in output the product between them. The latter is used to compute the instantaneous power consumption of the netlist during the simulation time, which consists in the generation of the so called *power trace*. Depending on the attack that will be executed next, this procedure can be repeated several times to obtain the necessary power traces. The following step involved the validation of the method. Then the simulated attack has been executed giving to the designer an idea on the robustness of the architecture. In order to determine how much reliable are the result obtained, the same attack has been conducted on a physical device, in particular an FPGA. The board that has been

used is a prototype developed by the Politecnico di Torino named VirtLab which can contain both the device under test and the measuring equipment by emulating a Digital Storage Oscilloscope (DSO) using the Analog Front-End (AFE) of a microcontroller present in it. Before executing the attack on a real device, a study has been conducted to determine which is the minimum working frequency of the multiplier in the simulation environment that allows to obtain a successful result. The outcome of this study has been useful to have an idea on the minimum working frequency that in the real world is necessary to reach in order to have a chance of guessing the secret key used during the multiplication. In the end, the result of the real attack and the simulated one have been compared and the appropriate conclusions have been drawn.

# Chapter 2

# Post-quantum cryptography

Quantum computers will soon become reality and their coming brings a lot of advantages and facilities for the human being, but at the same time serious problems arise in the cryptographic field, especially in the Public Key Cryptography (PKC), due to their impressive computing power which outscales the one of modern machines. In fact today cryptographic algorithm are based on problems which result computationally impossibles for modern computers, but it has been demonstrated that quantum computers can resolve them pretty easily. That is why in 2016 the NIST (National Institute of Standards and Technology) has begun the selection process for a new possible standard for the PKC and the digital signature problem. The proposed algorithms belong to one of the following categories[4]:

- **Code-based cryptosystem** [12];
- **Lattice-based cryptosystem** [11];
- **Isogeny algorithms** [15];
- **Multivariate cryptosystem** [3].

One of the most promising proposals is called *Classic McEliece* which belongs to the category of code-based cryptosystems and one of his variant is named *LEDAcrypt*.

## 2.1 Classic McEliece

McEliece is a public key (asymmetric) cryptosystem based on algebraic coding theory[10]. This system still nowadays results to be unbroken which means that it has not been discovered yet a polynomial-time algorithm able to successfully break its asymmetry. Its robustness stands in the fact that both the private and public keys are generator matrices of a binary linear error correcting code but by looking at the public key matrix it is impossible to recover the structure of the code and as a consequence discover the private key. The derivation of the Public Key (PK) from the Secret Key

(SK) has a polynomial time complexity, while the inverse problem is considered NP-hard.
More exhaustive explanations on error correcting codes can be found in [1], while in the following only some key concepts are reported. Let $GF_2 = (0,1,+,\times)$ be the Galois field of order 2 with the operations of addition and multiplication; and let $GF_2^k$ be its k-dimensional vector space. A binary code of length $n$ and dimension $k$ is defined as a map which uniquely associate to a k-bit information vector $u$ a n-bit codeword $c$.

$$C : GF_2^k \mapsto GF_2^n \tag{2.1}$$

A code is linear if, being $\Gamma$ the image of $C$, $\Gamma$ is a subspace of dimension k of $GF_2^n$, which means that there are k independent codewords $\{g_0, ..., g_{k-1}\}$ of $n$ bits that form a basis of $\Gamma$. A linear code has the property that the sum or difference of two codewords is still a codeword. As a consequence, any codeword can be expressed as a linear combination of the elements of the basis as shown in equation 2.2:

$$c = u_0 g_0 + u_1 g_1 + ... + u_{k-1} g_{k-1} \tag{2.2}$$

where the coefficients $u_0, ..., u_{k-1}$ are the bits of the information word $u$.
Equation 2.2 written in a matrix form becomes:

$$c = u \cdot G \tag{2.3}$$

where G is called *Generator matrix* of the code $C$ and its dimension is $k \times n$. It is important to notice that the generator matrix $G$ is not unique.
Now let's consider the vector space which is the orthogonal complement of the space $\Gamma$ and whose dimension is:

$$\dim(\Gamma^\perp) = n - \dim(\Gamma) = n - k = r \tag{2.4}$$

This means that there are $r$ linearly independent words of $n$ bits which can form a basis of $\Gamma^\perp$. As for the matrix $G$, also this basis can be written in a matrix form that is indicated as $H$ and whose name is *Parity Check Matrix*.

$$H = \begin{bmatrix} h_0 \\ . \\ . \\ . \\ h_{r-1} \end{bmatrix}$$

Being the orthogonal space of $\Gamma$ each codeword $c$ must satisfy the following equation:

$$H \cdot c^T = 0 \tag{2.5}$$

The quantity $S = H \cdot c^T$ is called *Syndrome* of the codeword $c$ through $H$; so each codeword belonging to the code $C$ must have a null syndrome, which means that must

satisfy all the parity check equations of the code. Moreover, as for $G$, also the parity check matrix is not unique for the code. A code is an ECC (*Error Correcting Code*) with an error correcting capability $t$ if, when it is used for the encryption and decryption of an information word that must be sent through a communication channel, it is able to correct up to $t$ errors added to the codeword sent.

McEliece uses a particular family of codes named Goppa codes as private codes. The first parameters to be chosen are the length $n$ of the code, its dimension $k$ and its error correcting capability $t$. Then an irreducible polynomial called $a(x)$ of degree $t$ over $GF(2^m$ has to be found.

### 2.1.1 Key and matrices generation

From the polynomial $a(x)$, the generator matrix $G$ is computed. After that other 2 matrices must be generated in order to compose the private key: a permutation matrix P with size $n \times n$ and a dense, non singular, scrambling matrix $S$ whose dimension is $k \times k$. The public key is then computing by multiplying together the 3 matrices as shown in equation .

$$G' = S \cdot G \cdot P \tag{2.6}$$

### 2.1.2 Encryption

When the user A wants to send an information word $u$ to the user B, he multiplies it with the public key of the user B and add to the result an error vector $e$ of weight $t$.

$$x = u \cdot G' + e = c + e \tag{2.7}$$

If the transmission channel is not ideal, which means that some bits of the message can be flipped during the transmission, the weight of $e$ must be reduced in order to be sure to not exceed the maximum error capability $t$ of the code.

### 2.1.3 Decryption

When the user B receives the encrypted message $x$, he firstly computes the inverse of the permutation matrix $P$ and multiplies it with the message $x$ [1].

$$y = x \cdot P^{-1} = (u \cdot S \cdot G \cdot P + e) \cdot P^{-1}$$

$$\Downarrow$$

$$y = u \cdot S \cdot G + e \cdot P^{-1}$$

After that, using some known algorithms as the Patterson's one, B is able to remove the error vector contribution obtaining [1]:

$$y = u \cdot S$$

The last step consists in inverting the scrambling matrix $S$ and multiplying it by $u$ to recover the original message word.

## 2.2 LEDAcrypt

LEDAcrypt[9] cryptosystem has been developed by an italian research team and it includes 2 algorithms inside:

- **LEDAkem** which is a key encapsulation mechanism based on the Niederreiter cryptosystem using QC-LDPC (*Quasi Cyclic Low Density Parity Check*) codes;
- **LEDApkc** that consists in an asymmetric cryptosystem based on McEliece using QC-LDPC codes.

In the following the LEDApkc algorithm is described since it is the one based on the McEliece cryptosystem.
LEDA overcome one of the major problems of the classic McEliece cryptosystem which is the low amount of memory required to store the key due to the cyclicity and sparsity of the key.

### 2.2.1 Key and matrices generation

The secret key is composed of a QC matrix $H$ composed of $n_0$ cyclic square matrices of size $p$[6]. That is why this matrix can be stored in memory by just saving its first row, in particular the index of the bits set to 1 in it. Its structure is[9]:

$$H = [H_0, H_1, ..., H_{n_0-1}]$$

where each square matrix $H_i$ is:

$$H_i = \begin{bmatrix} h_0 & h_1 & ... & h_{p-1} \\ h_{p-1} & h_0 & ... & h_{p-2} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ h_1 & h_2 & ... & h_0 \end{bmatrix}$$

In order to produce the public key, another matrix is needed. It is a transformation matrix $Q$ which is quasi cyclic and its structure is[9]:

$$Q = \begin{bmatrix} Q_{0,0} & Q_{0,1} & \dots & Q_{0,n_0-1} \\ Q_{1,0} & Q_{1,1} & \dots & Q_{1,n_0-1} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ Q_{n_0-1,0} & Q_{n_0-1,1} & \dots & Q_{n_0-1,n_0-1} \end{bmatrix}$$

Each block $Q_{i,j}$ is a square circulant matrix of size $p$ and their weights are defined by another QC matrix[9]:

$$w(Q) = \begin{bmatrix} m_0 & m_1 & \dots & m_{n_0-1} \\ m_{n_0-1} & m_0 & \dots & m_{n_0-2} \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ m_1 & m_2 & \dots & m_0 \end{bmatrix}$$

where the sum of all the $m_i$ of the same row or column is constant. By doing the product between $H$ and $Q$, another circulant matrix is obtained and it is called $L$[9]:

$$L = HQ = [L_0, L_1, ..., L_{n_0-1}] \tag{2.8}$$

In order to compute the public key from L two conditions must be satisfied[9]:

- the weight of each row of $H$ and the sum of the $m_i$ must be odd;
- the matrix Q must have maximum rank.

The public key, which means its generator matrix $M$, is computed in the following way[9]:

$$M = L_{n_0-1}^{-1} L = [M_l, I_p] \tag{2.9}$$

### 2.2.2 Encryption

The steps performed to encrypt an information word $u$ are the following[9]:

1. the information word is multiplied with the public key:

$$m = u \cdot M^T$$

2. a vector error of weight t is generated;
3. the vector error is added to $m$ to produce the final cyphertext:

$$x = m \oplus e$$

### 2.2.3 Decryption

To retrieve the original information word LEDApkc uses the Q-Decoder algorithm which first of all computes the syndrome of the received message as $s = Hx^T$ and then execute a finite number of iterations where in each of them the following operations are performed[9]:

1. $\sigma^{(l)} = s^{(l-1)}H$
2. $\rho^{(l)} = \sigma^{(l)}Q = [\rho_1^{(l)}, \rho_2^{(l)}, ..., \rho_n^{(l)}]$
3. the correlation thresholds are computed;
4. the position of the correlation values higher than the threshold are saved;
5. the error vector is updated in the following way:

   $$e^{(l)} = e^{(l-1)} \oplus e^{(new)}$$

   where $e^{(new)}$ is a vector of $n$ bits where those set to 1 are positions of the correlation values higher than the threshold;
6. the new syndrome is computed : $s^{(l)} = s^{(l-1)} + e^{(l)}L$;
7. if $s^{(l)} = 0$ the decoding is successful. Instead if the syndrome is not null and the current iteration is the last one the decoding has failed, otherwise a new iteration is performed.

# Chapter 3

# Vector By Sparse Circulant binary multiplier

Most of the code-based post-quantum algorithms that are participating to the NIST selection process uses Low Density Parity Check codes (LDPC) or Moderate Density Parity Check codes (MDPC). In those algorithms there is a frequent use of multiplications between a vector and a sparse circulant matrix; so in order to efficiently compute the products ad-hoc architectures it have been designed. The one described in this chapter is a simplified version of the Schoolbook multiplier adapted for the use of very sparse matrices [7] that has been used inside an hardware implementation of the LEDAkem, the key encapsulation mechanism of the LEDAcrypt cryptosystem.

## 3.1 Algorithm

The algorithm used to perform the vector by matrix multiplication is very similar to the canonical one, but there is a huge improvement in the performance due to the sparsity and cyclicality of the matrix. In fact the final product can be expressed as the sum of partial products each of them being a rotated version of the input vector. The number of partial products is equal to the number of bits set to 1 present in the first row of the matrix. To better clarify it an example is proposed where the length of the vector $v$ is $p = 5$ and the number of 1 set in the first row of the matrix $A$ is $d_A = 3$. The vector-matrix product $r$ can be expressed as follows[7]:

$$r_0 = a_0v_0 + a_1v_1 + a_2v_2 + a_3v_3 + a_4v_4 \tag{3.1}$$
$$r_1 = a_4v_0 + a_0v_1 + a_1v_2 + a_2v_3 + a_3v_4 \tag{3.2}$$
$$r_2 = a_3v_0 + a_4v_1 + a_0v_2 + a_1v_3 + a_2v_4 \tag{3.3}$$
$$r_3 = a_2v_0 + a_3v_1 + a_4v_2 + a_0v_3 + a_1v_4 \tag{3.4}$$
$$r_4 = a_1v_0 + a_2v_1 + a_3v_2 + a_4v_3 + a_0v_4 \tag{3.5}$$
$$\tag{3.6}$$

Since $d_A = 3$, let's suppose that $a_0 = a_2 = a_3 = 1$ and $a_1 = a_4 = 0$. In this case the final product become[7]:

$$r_0 = v_0 + v_2 + v_3 = v_0 + v_2 + v_3 \tag{3.7}$$
$$r_1 = v_1 + v_3 + v_4 = v_1 + v_3 + v_4 \tag{3.8}$$
$$r_2 = v_0 + v_2 + v_4 = v_2 + v_4 + v_0 \tag{3.9}$$
$$r_3 = v_0 + v_1 + v_3 = v_3 + v_0 + v_1 \tag{3.10}$$
$$r_4 = v_1 + v_2 + v_4 = v_4 + v_1 + v_2 \tag{3.11}$$
$$\tag{3.12}$$

It can be seen that r is composed by the sum of 3 partial products and each of them consists in the input vector $v$ rotated by an amount of positions equal to the index of a 1 present in the first row of the matrix.
The formal description of the algorithm is provided below[7]:

**Input**: length-$p$ vector **p**, weight of **A**
(interpreted as the weight of each row and column of A) $\mathbf{d}_A \in N$,
first row of **A** represented as a list of positions $S_A$ with size $d_A$
**Output**: length-$p$ vector **r=vA**
1: $\mathbf{r} = \mathbf{0}$
2: **for** $i = 0$ **to** $d_A$ - 1 **do**
3: $k = S_A(i)$
4: $\mathbf{v}^{(i)} = [v_k, v_{k+1}, ..., v_{p-1}, v_0, v_1, ..., v_{k-1}]$
5: $\mathbf{r} = \mathbf{r}+\mathbf{v}^i$
6: **end for**
7: **return r**

## 3.2 Architecture

The architecture of the multiplier can be divided in 2 blocks:

- the **Control Unit**;
- the **Datapath**;

### 3.2.1 Datapath

The datapath is composed by the following units and their interconnections can be seen in the schematic shown in Figure 3.2[6]:

- **Message memory**: it is the memory storing the Message. It is divided in words of 8 bits, but since the message length must be a prime number, the second-last word in partially filled with zeros and the last one contains only zeros. This is necessary to guarantee a correct behaviour when the last word of the message has been read and the new one to be consumed is the first one;
- **Ltr memory**: it is the memory storing the secret key. As introduced before, the key is a huge sparse circulant matrix and in order to occupy less resources as possible to store it, by exploiting its circularity it is sufficient to store the positions of the bits set to 1 in its first row since the position of the 1s in the other rows can be retrieved by the first one. Each cell of the memory is composed of 15 bits where the MSB is set to 0 or 1 depending on the position of the shift with the respect of the index of the last bit in the second-last row of the Message memory. This bit is useful to correctly manage the transition from the last word of the message memory to the first one. The remaining 14 bits contains the effective position of the 1 in the key matrix;
- **Syndrome memory**: it is the memory storing the final product and it is divided in words of 8 bits;
- **Message counter**: it is in charge of providing the address in order to read the correct word from the Message memory. Its initialization value is taken from the content of the Ltr register, in particular $Ltr[3,14]$; then it is always incremented by one until the message has been entirely read. After that, when a new key position is read, it is initialized again;
- **Ltr counter**: it provides the address used to read the correct position from the Ltr memory. It is initialized to 0 and when a partial product has been entirely computed it is incremented by 1;
- **Syndrome counter**: it is used to index properly the Syndrome memory. It is initialized to 0 and it is incremented by one until the entire partial product has been saved; then it is reset to 0 and the cycle restarts;
- **Msg2 register**: it receives in input the word coming from the Message memory and its output is connected to the input of the Msg1 register and to the Collapse Unit;

- **Msg1 register**: its input is connected to the output of the Msg2 register in order to act as a sort of shift register where instead of shifting a single bit an entire byte is shifted. Its output is connected to the Collapse Unit;
- **SynNew register**: it is the register where each word of a partial product is saved before being stored into the Syndrome memory. The input data is the result of the xor operation between the word read from the Syndrome memory and the output of the Collapse Unit;
- **Ltr register**: it contains the current key position read from the Ltr memory. Its content pass through some logic which transform it into control signals for the Collapse Unit;
- **Collapse unit**: this is the core unit of the datapath since it is the component in charge of computing the shifted message words. It receives in input 2 message words coming from the Msg1 and Msg2 register and select a portion of 8 bits based on the shift that must be applied to the message. This unit is composed of 4 levels where the $l^{th}$ level performs a rotation of $8/2^{l-1}$ bits. Each level is composed by a multiplexer with 2 inputs each one composed by $n_b$ bits and produces in output a word composed of $n_b + n_b/2$ bits as shown in Figure 3.1.

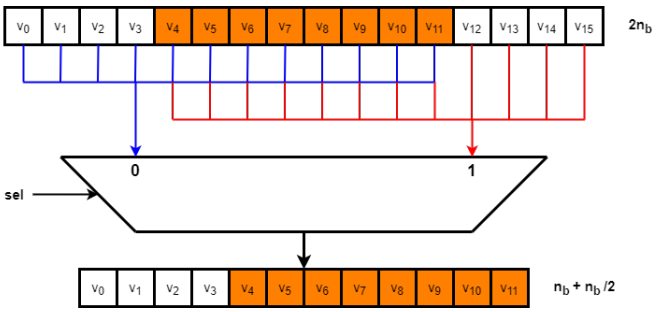



Figure 3.1: First level of the Collapse Unit

### 3.2.2 Control unit

The control unit is implemented as a finite state machine that pass through the following states in the same order in which they are listed below[6]:

1. **Load_Ltr_FirstCycle**: here the first position of the key is read and saved in a register. At the same time the register used to read the content of a syndrome memory word is cleared to 0;
2. **Load_Msg**: based on the value of the Ltr position read, the first word of the Message is read and saved in the Msg2 register and the Message counter is initialized;

3. **Load_Msg_Syn**: a new Message word is read and saved in Msg2 while the previous content of Msg2 is transferred to the register Msg1. Simultaneously the Syndrome memory is read and the content of the first word is saved in the Syndrome register;

4. **Cmp**: Msg1 and Msg2 register values are given in input to the collapse unit which produce the shifted message word. This word is xored with the content of the Syndrome register to produce the initial word of the first partial product that is stored in the SyndromeNew (SynNew) register;

5. **Store**: the content of the SynNew register is saved in the Syndrome memory.

After that, the CU pass through the following states in loop to produce the entire first partial product:

1. **Load_Msg_Syn**;

2. **Cmp**;

3. **Store**.

The loop terminates when the last word of the Syndrome memory is written, then another position from the key Ltr memory is read and the entire flow described above is repeated with the exception that instead of going through the *Load_Ltr_FirstCycle* state the CU goes into a normal *Load_Ltr* state.
Since the length of the message must be a prime number, the second-last word of the message saved in memory contains partially the last message bits and the remaining part is filled with zeros and the last one is completely filled with zeros. When this word is read, a *PartialStore* is done instead of a normal *Store* which means that the Syndrome word is computing by connecting together the words produced by two successive iterations.

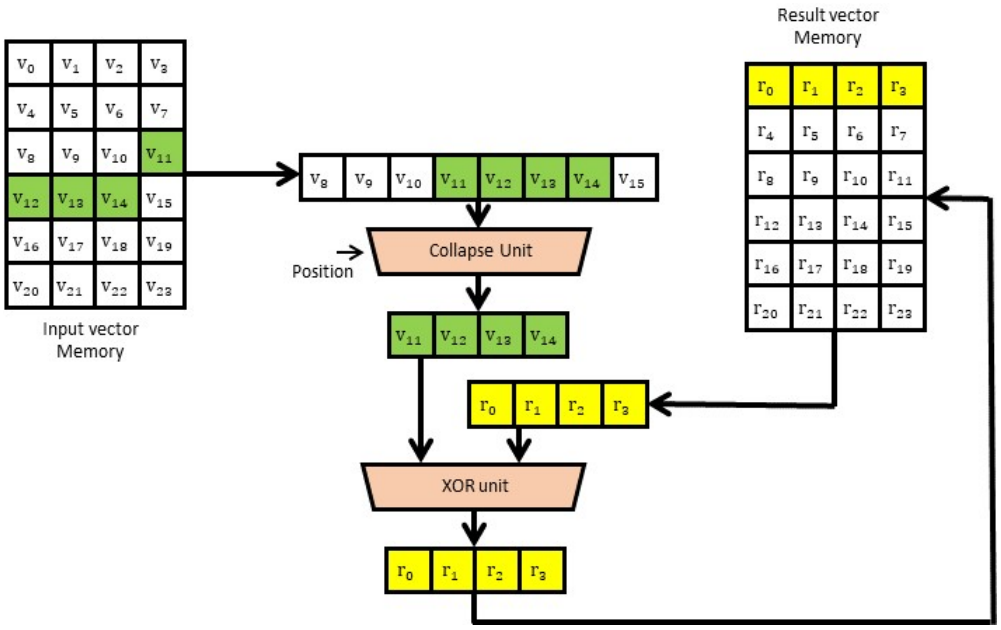



Figure 3.2: Schematic of the multiplier datapath.

# Chapter 4

# Power Analysis Attacks

***P**ower **A**nalysis **A**ttacks* refers to a method applied to a device in order to discover information on the secret key. PAA belong to the category of *passive non-invasive Attacks* which means that only accessible interfaces of the device are exploited, no evidence of the attack is left behind and the key is obtained by exploiting physical properties of the device[8]. Passive non-invasive attacks are also called *side-channel attacks* and have gained a lot of attention during the last years. The three most important types of side-channel attacks are:

- **timing attacks** which exploits the execution time of the device;
- **power analysis attacks** which exploit its power consumption;
- **electromagnetic attacks** which exploit the electromagnetic field emitted by the device.

In this chapter the focus is put on Power Analysis Attacks. In particular in the following is provided the description of the 3 most important attacks that belong to this category and they are:

- **Simple Power Analysis** attack(SPA);
- **Differential Power Analysis** attack(DPA);
- **Correlation Power Analysis** attack(CPA).

All this types of attack require to have a recording of the power consumption of the device over the time in which it is executing some cryptographic operation, which is also called *power trace*. The number of traces that are needed to perform the attack depends on the type of attack that is considered, in fact SPA require a single trace while DPA and CPA need multiple traces to be executed.

## 4.1 Simple Power Analysis attack

In Simple Power Analysis attacks the attacker tries to retrieve the secret key using only a single power trace. This attack requires the knowledge of the architecture of the device where the cryptographic algorithm is executed and since only a single trace is used, the key must have a huge impact on the power consumed by the device[8].
The idea behind SPA is to recognize patterns within the trace and that is why it is usually used when the algorithm is run on a microcontroller. In this case the algorithm is translated into a sequence of instructions which have a unique power consumption pattern because each of them works with different hardware components of the microcontroller that consumes different amount of power. The fact that each instruction has a unique pattern can be exploited by the attacker if the sequence of instructions executed depends on the key value, especially on public-key cryptography. In Figure **??** it is shown the power consumption of a microprocessor while it is performing an RSA decryption. It is clearly visible that in the power trace there are several patterns each of them corresponding to a single or a sequence of instructions. If the visual inspection of the trace is not enough to determine the key used during the cryptographic operation, there are more advanced techniques such as template based attacks.

### 4.1.1 Template based attack

In template based attacks the power trace is characterized by the means of a multivariate normal distribution which has a mean vector $m$ and a covariance matrix $C$. The couple composed by $m$ and $C$ is named *template*[8]. In this type of attack it is assumed that the attacked device can be fully characterized, which means that, for example, the attacker has a copy of the target device and uses it to record the power consumed when different inputs are provided. If multiple traces can be recorded for each possible pair of inputs ($d_i$,$k_j$), where $d_i$ is a possible message and $k_j$ is a possible key, the attacker groups together the traces regarding the same inputs and compute their mean vector and covariance matrix. In this way he has obtained a template for each possible couple of inputs.
After that the attacker computes the probability:

$$p(t;(m,C)_{d_i,k_j}) = \frac{exp(-\frac{1}{2}\cdot(t-m)'\cdot C^{-1}\cdot(t-m))}{\sqrt{(2\cdot\pi)^T\cdot det(C)}} \tag{4.1}$$

This probability is computed for every template and the higher is p, the higher the template fits to the power trace $t$. Since each template is associated to a key value, the correct key value is the one associated to the template that gives the highest probability.
In the following, additional strategies and hints used in the template building phase and in the template matching phase are provided.

### Template building phase

In the first phase when the attacker tries to build suitable templates, in addition to the technique described above which consists in creating a template for every possible couple of inputs, there are other solutions that can be adopted. One of them is to build templates referred to an intermediate value produced during the algorithm flow. The intermediate value must be chosen carefully since it must be a function of both the inputs (message and key), but the advantage is that this technique reduces the number of template to be produced.
A further improvement consists in associating each intermediate value to a predicted power consumption. There are several power model that could be used and in this way it is possible to reduce further more the templates adopted because more values can be associated to the same power consumption. At the end of the attack when the most suitable template is known, it is straightforward to retrieve the key used during the operation by just going backwards.

### Template matching phase

In this phase in order to avoid the exponentiation, the logarithm is applied to the equation 4.1[8]:

$$ln(p(t;(m,C)_{d_i,k_j})) = ln(\frac{exp(-\frac{1}{2}\cdot(t-m)'\cdot C^{-1}\cdot(t-m))}{\sqrt{(2\cdot\pi)^T\cdot det(C)}}) \tag{4.2}$$

$$\downarrow$$

$$ln(p(t;(m,C))) = -\frac{1}{2}(ln((2\cdot\pi)^{N_{IP}}\cdot det(C)) + (t-m)'\cdot C^{-1}\cdot(t-m)) \tag{4.3}$$

In this way the template that leads to the smallest absolute value of the logarithm of the probability is the correct one.

$$|ln(p(t;(m,C)_{d_i,k_j}))| < |ln(p(t;(m,C)_{d_i,k_l}))| \quad \forall l \neq j \tag{4.4}$$

A further simplification can be done by considering the covariance matrix equal to the identity matrix in order to avoid computing the inverse of the covariance matrix; in this way the correlation between the trace points is not considered and the resulting template composed only by the mean vector is called *reduced template*. Using the reduced template, the equation 4.1 becomes[8]:

$$p(t;m) = \frac{exp(-\frac{1}{2}\cdot(t-m)'\cdot(t-m))}{\sqrt{(2\cdot\pi)^{N_{IP}}}} \tag{4.5}$$

Also in this case the logarithm can be applied:

$$ln(p(t;m)) = -\frac{1}{2}(ln(2\cdot\pi)^{N_{IP}} + (t-m)'\cdot(t-m)) \tag{4.6}$$

The method that uses *reduced templates* is also called *least square* (LSQ) test and since the most relevant term in the equation is the square of the difference between $t$ and $m$, the rule that can be used to identify the correct template is the following:

$$(t-m_{d_i,k_j})'\cdot(t-m_{d_i,k_j}) < (t-m_{d_i,k_l})'\cdot(t-m_{d_i,k_l})\forall l \neq j \tag{4.7}$$

## 4.2 Differential Power Analysis attack

With the respect of SPA, Differential Power Analysis attacks[5] need multiple power traces to be executed. The methodology consists in dividing the set of traces into different subsets, computing point by point the average of these subsets and then making the difference between them. If the choice of assigning each trace to a specific subset is correlated to the trace measurements, the difference of the averages will be a value far from 0, otherwise the difference will be very close to 0. This difference can be plotted in a XY graph where the X axis represents the time and the Y axis indicates the difference value, as shown in Figure 4.1.

Going more into the details the DPA attack is composed of the following stages[5]:

1. Choice of the instrumentation;
2. Measurement;
3. Signal processing;
4. Prediction and selection function generation;
5. Differential traces production;
6. Results evaluation;

### 4.2.1 Choice of the instrumentation

The first thing to do is to choose the instruments to communicate with the device under attack and to record its power consumption during the time in which it is executing the cryptographic operation. Usually, to record the power consumption we use a digital oscilloscope that measures the voltage drop across a resistor put in the supply line of the device or, thanks to the use of current probes, the current that it absorb. The typical setup used to perform a DPA attack is shown in Figure 4.2.

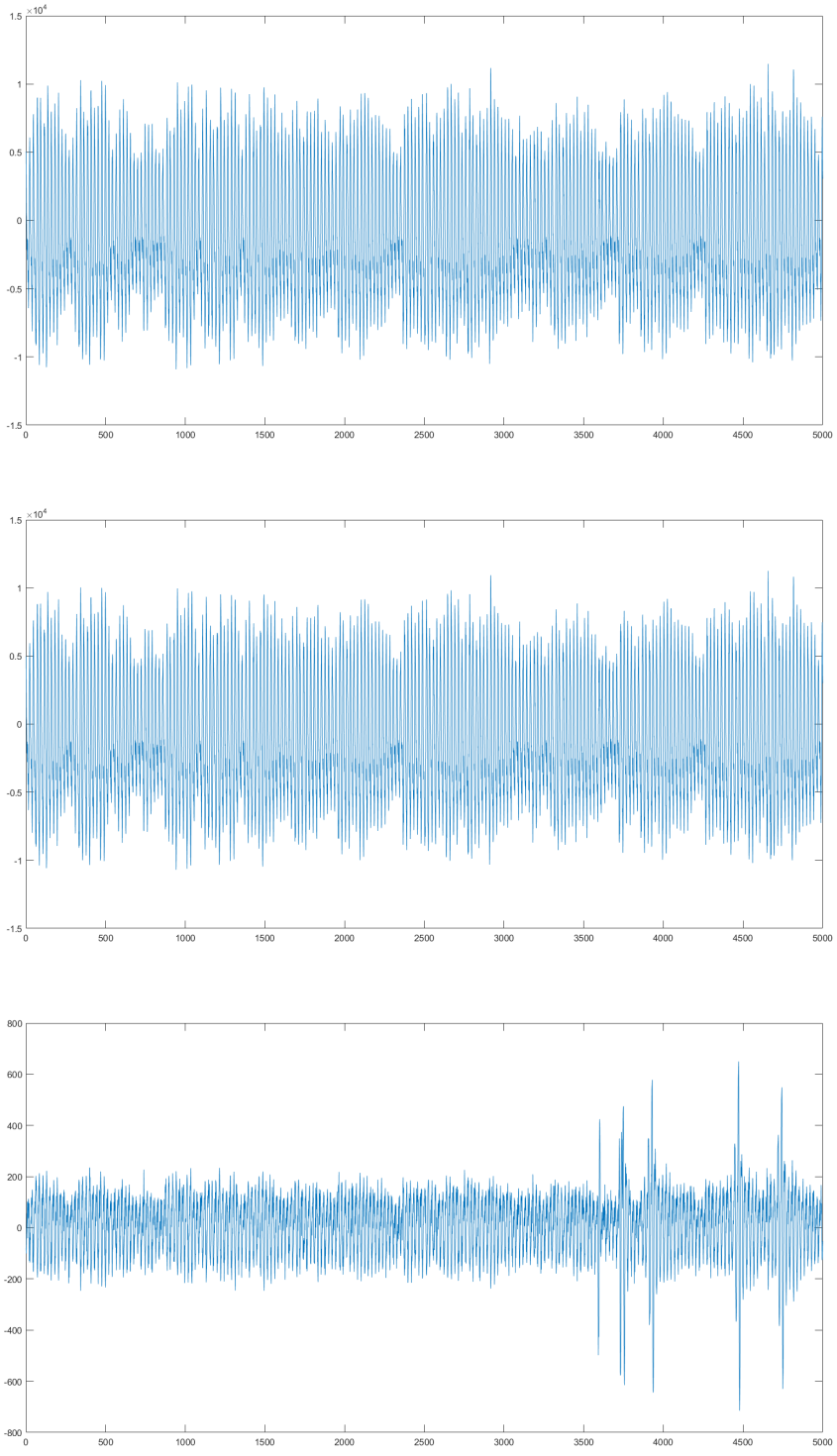



Figure 4.1: Figure showing 3 traces. Going from top to bottom, the first and second one are the average traces of two subsets, while the third one is the difference trace.

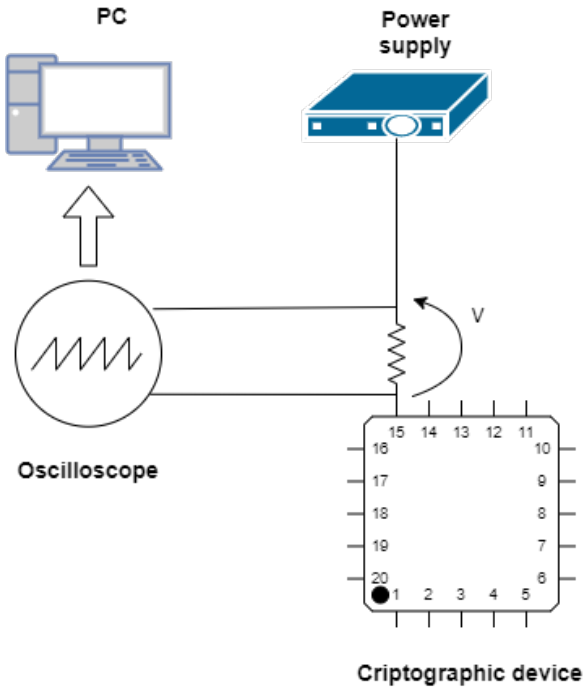



Figure 4.2: Typical setup for a DPA attack.

### 4.2.2 Measurement

The measurement stage is the step in which the power consumption of the cryptographic device is sampled by the oscilloscope and sent to the device where the DPA algorithm is implemented. It is important to reduce as much as possible the noise superposed to the signal by adding analog filters, adjusting the oscilloscope bandwidth and sampling rate and reducing as much as possible the distance between the various instruments.

### 4.2.3 Signal processing

Since each trace can be seen as a signal, a technique that allows to obtain better measurements and, as a consequence, having more chances to recover the secret key, is applying some signal processing to the entire set of traces. The technique usually adopted is the time alignment of the traces against a time reference point or in alternative to perform the DPA attack in the frequency domain, which implies to compute the Fourier transform of the power traces [5]. The second common technique that is used measures multiple traces of the power consumption. These traces are obtained giving in input to the device the same data and then average them in order to reduce significantly the amount of noise superimposed to the signal. In general, it is common that only small portions of the trace contains useful information for the DPA attack, if the device can be characterize it is useful to remove the non interesting parts of the

trace. In this way, the execution time of the attack is minimized due to the fact that the amount of data to process is strongly reduced. Moreover, another useful technique is the *trace compression* where consecutive measurements are summed together and repetitive effects are eventually removed from the trace [5].

### 4.2.4 Prediction and selection function generation

DPA attacks exploit the fact that in the device some steps of the computation depend on the secret key value. The attacker has to develop a selection function that will be used to assign traces to subsets and they are usually based on the guess on the value assumed by an intermediate variable produced during the computation. The outcome of these functions can be binary (0 or 1) or can be non-binary. In the latter case the output of the function assumes the role of a weight when in the next step the traces will be averaged and each weight can be 0, positive or negative. Usually in DPA attacks binary functions are used as, for example, the value of the LSB of the result coming from an SBOX in an AES encryption.
After that, the selection function must be applied to all possible values that the key portion which the attacker wants to guess can assume. So if the target of the DPA is the value of one byte of the key, the traces will be divided in 2 subsets for each of the 256 possible values that the key byte can assume. A scheme of the entire process is shown in Figure 4.3.

### 4.2.5 Differential traces production

This is the phase where the most of the DPA calculation is performed since at this point the average of the traces contained in the same subset is computed. The complexity of this task increase proportionally with the number of traces, the length of each trace and the number of selection functions used (which is equal to the number of possible values that the target portion of the key can assume). That is why usually some optimization techniques are used such as the compression and the removal of the non interesting portions of the traces in order to reduce the trace length and as a consequence the total number of points that need to be averaged. Since this task is highly parallelizable, another possible way to reduce the bottleneck is to work in parallel with different threads and machines.

Once the averages have been computed, a differential trace is produced for each possible key value by subtracting the average trace of one subset to the average trace of the other one. A scheme of this phase is shown in Figure 4.4.

### 4.2.6 Results evaluation

In most of the cases the result evaluation consists in the visual inspection of the differential traces previously obtained. The correct key value is the one whose differential

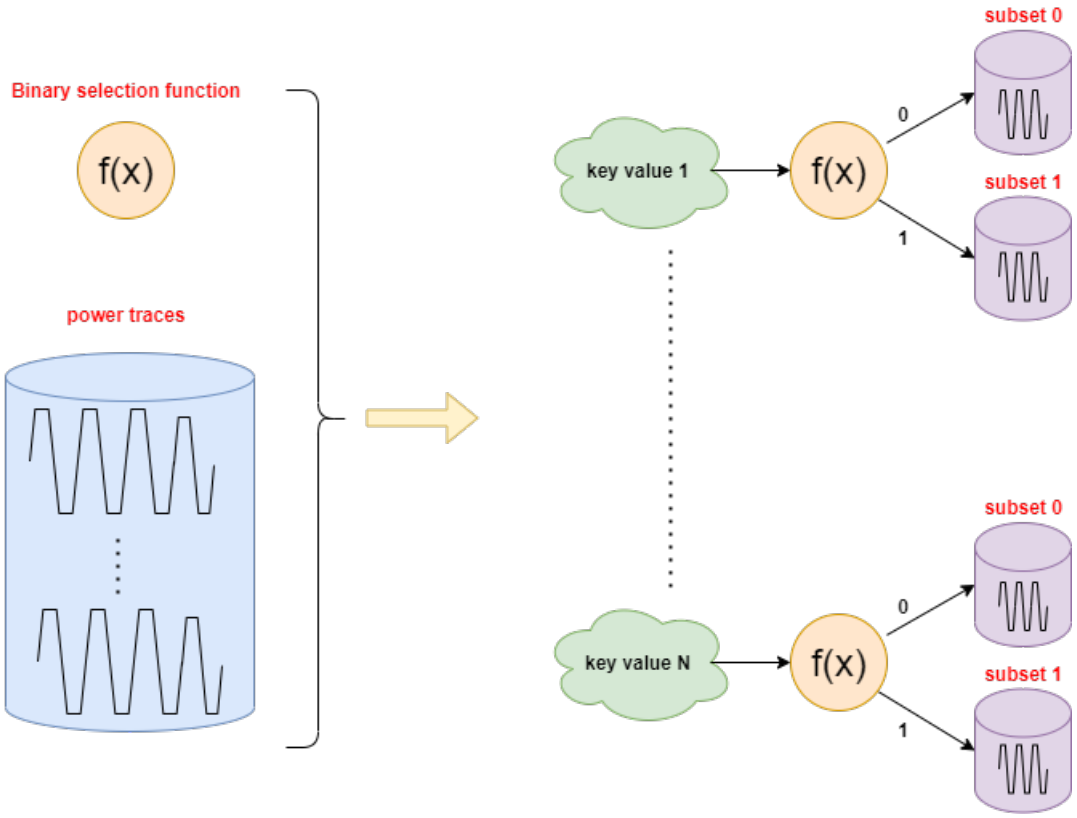



Figure 4.3: Scheme of the trace division into subsets based on the value assumed by the selection function for each possible key value.

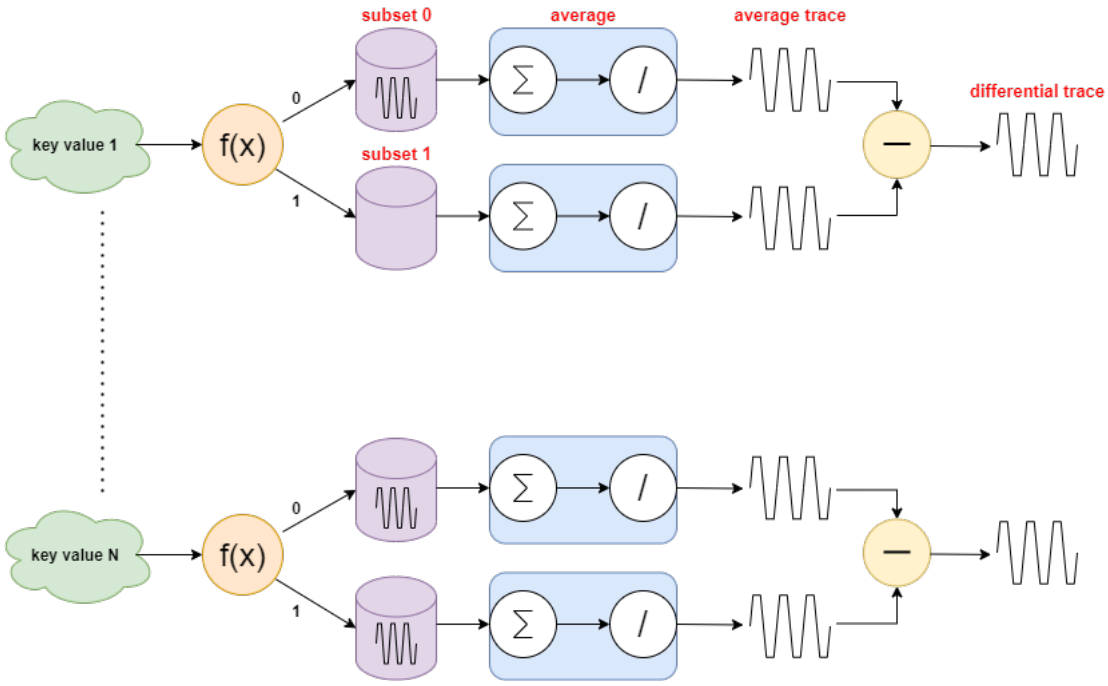
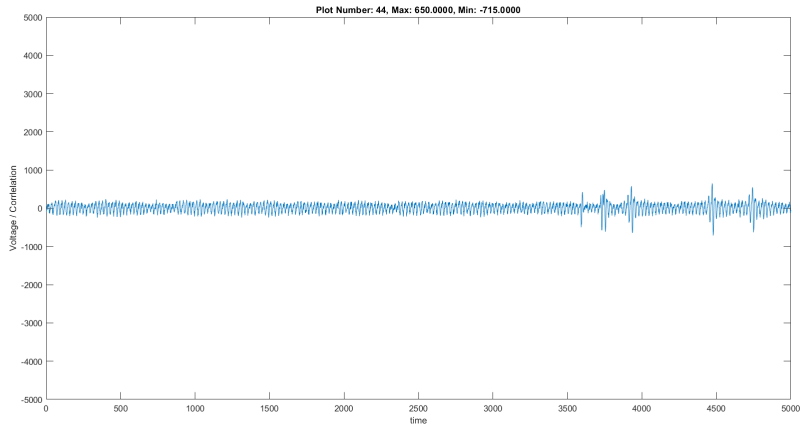
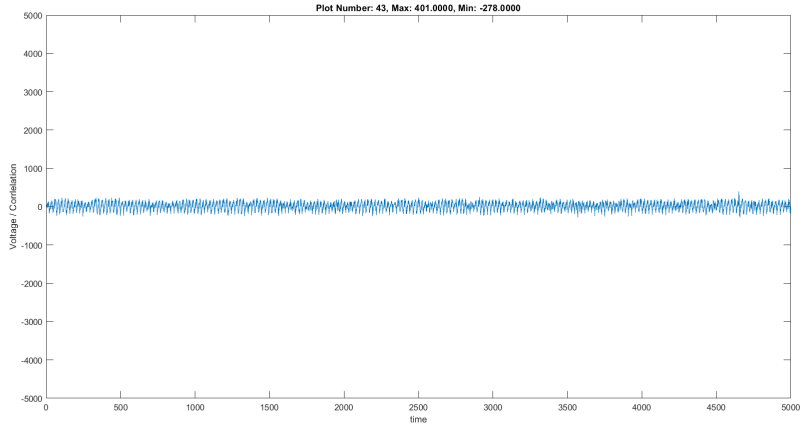



Figure 4.4: Scheme representing how the differential trace for each key value is produced starting from the trace subsets.

trace presents the highest peaks; while the traces related to incorrect values will have smallest peaks or even none. Some results may be misleading since regions with unusually high noise can show spurious peaks, but in order to compensate to these effects each point of the trace can be divided by the standard deviation of all the traces at the same point [5]. Moreover, for certain types of DPA attacks and algorithm, differential traces different from the correct one can have a significant correlation with the target leak and so they can show huge spikes as well. The key values related to those traces are called *harmonics* and, although initially they can make some confusion on the guessing of the correct value, they can provide useful information to the attacker. In fact, by knowing the selection function used and the cryptographic algorithm he can predict the harmonic pattern that would be generated and check if it is compliant with the observed one [5]. In Figure 4.5 is shown the difference between the differential trace of a correct key value and the one of an incorrect value.




(a) *Correct key differential trace.*




(b) *Incorrect key differential trace.*

Figure 4.5: Difference between the differential trace of a correct key value and an incorrect one.

## 4.3 Correlation Power Analysis attack

As for DPA attacks, also Correlation Power analysis (CPA) attacks [2] require the use of multiple traces. The difference between them is in the way in which the traces are analyzed to compute which is the correct key used in the cryptographic operation. The general idea of this type of attack is to compute the correlation between the real power measured from the device and the predicted power consumption during the generation of a specific intermediate value. Usually the key which gives the highest correlation is the correct one, but sometimes due to the presence of harmonics there can be more than a single value which produce strong correlation values. In that case by reasoning on the algorithm properties all the harmonics can be discarded and the correct key value is found.
The attack is composed of the following steps:

1. **Power consumption measurement**;
2. **Power model choice**;
3. **Intermediate value choice**;
4. **Intermediate value computation**;
5. **Power consumption prediction**;
6. **Correlation traces generation**;
7. **Results evaluation**.

A block diagram illustrating the steps of the CPA attack is shown in Figure 4.8.

### 4.3.1 Power consumption measurement

The setup used to measure the power consumption of the cryptographic device in a CPA attack is exactly the same described in section 4.2.2 for a DPA attack since the power traces needed to perform this attack are the same.
That is why also some signal processing is applied after the collection of the traces. The most used technique in this attack are the averaging to reduce the noise superimposed to the measurement and the cut of useless parts to lighten the amount of data to be processed in the next steps. All the power traces must be recorded while the device is using the same secret key, but a different message for each trace. The messages can be random or chosen appropriately if the attacker can in some way have access to the inputs of the device. A crucial point in this part is assessing the needed number of power traces to guess the correct key value, but this will be described in section 4.3.7 since the concept of Pearson correlation coefficient has not been treated yet.
The whole recorded power traces can be grouped together in order to form a matrix

that we call T[8]. Its dimension will be N x K where N represents the number of collected power traces and K the number of samples that compose each trace. In this way each row of the matrix T is a different power trace.

### 4.3.2 Power model choice

This step consists in the choice of a suitable power model to be used to model the power consumed by the device in a certain instant of time. The most important power models are:

- **Bit model**;
- **Hamming weight model**;
- **Hamming distance model**;
- **Zero value model**;

#### Bit model

The bit model is the simplest of the models listed above. It consists in modeling the power consumed by the device to 0 or 1 based on the value of a single bit of an intermediate value produced during the device computation. The power consumed is assumed to be 1 if the chosen bit is 1 and 0 otherwise. This power model, being very simple, can be used only for software implementations of the cryptographic algorithm (which means that the algorithm is run in a microprocessor) and the processor leakage is pretty high.

#### Hamming weight model

The Hamming weight model (HW) can be seen as en extension on multiple bits of the bit model since it models the power consumed by the device with a scalar number equal to the number of bits set to 1 of an intermediate value. This type of model is suitable for software implementations of the cryptographic algorithm because, for example, it can model well the power consumed by the device when the intermediate value is transferred using a bus which is precharged to 0. In that case the power consumed by the device is equal to the number of 0 to 1 transitions that occur in the bus, which is equal to the Hamming weight of the value transferred. Giving a more general definition, this model can give very good results if, considering a transition from a value $v_0$ to a value $v_1$ where $v_1$ is the target intermediate value, the value of $v_0$ is constant.

### Hamming distance model

The Hamming distance model is used to predict the power consumed during a transition that occurs inside the device. When there is a transition from a value $v_0$ to a value $v_1$, the power consumed is modeled with a scalar value equal to the number of bits in the same position that differs from the 2 values. For example, if $v_0$ is equal to 0110 and $v_1$ is equal to 1010, the Hamming distance corresponds to 2.
This power model is used especially for hardware implementation of a cryptographic algorithm, which means that the algorithm is described in hardware using an ASIC, since it allows to predict in a precise way the dynamic power consumed by a logic cell, especially a CMOS one. In fact the power consumption of a CMOS cell can be seen as the sum of 2 components as shown in equation 4.8: the leakage power and the dynamic power.
The leakage power is a contribution which is always present when the cell is supplied and its value is almost insignificant with the respect of the dynamic power. The leakage power is due to a small current that flows across the silicon substrate even if the cell is in idle state. The major part of the whole power consumed by a CMOS cell consists in the dynamic power, which is the power consumed when there is a transition at the output pins of the cell and it is due to the current that flows across it in the little amount of time in which, during the transition, both the pull-up and pull-down networks are conducting, and the current necessary to charge internal nodes.

$$P_{tot} = P_{leak} + P_{dyn} \tag{4.8}$$

$$P_{dyn} \gg P_{leak} \tag{4.9}$$

$$P_{dyn} = \frac{1}{2} \cdot f_{clk} \cdot C_{load} \cdot V_{dd}^2 \cdot \propto \tag{4.10}$$

### Zero value model

The zero-value model (ZV) assumes that the power consumed by the device when it receives in input values equal to 0 is much less than the power consumed in all the other cases. In particular it models the power consumption equal to 0 when the inputs are 0 and 1 otherwise as shown in equation 4.11.

$$P_{i,j} = ZV(v_{i,j}) = \begin{cases} 0 \text{ for } v_{i,j} = 0 \\ 1 \text{ for } v_{i,j} \neq 0 \end{cases} \tag{4.11}$$

This model can be applied in fewer cases than all the other model described before, but can be very efficient when it is used in the right context, for example when modeling the power consumption of a multiplier.

### 4.3.3 Intermediate value choice

In CPA attacks, as for DPA attacks, it is not important to model the power consumption of the device over its entire computation, but it is sufficient to predict it in a single instant of time, usually during the computation or saving of an intermediate value.
The properties that the intermediate value must have in order to be a good candidate are the following:

- **message dependency**;
- **key dependency**.

Those two conditions are fundamental, otherwise it is impossible to correlate the power consumption to the secret key used during the cryptographic operation. Moreover, if the Hamming distance model is used, two consecutive intermediate values must be chosen since it requires to have a transition, while if the choice fell on one of the other power models only one intermediate value is sufficient.

### 4.3.4 Intermediate value computation

Once the target intermediate value is chosen, the next step consists in computing its value for every possible couple (message,key). Before doing that the attacker needs to understand more deeply the key dependency of the intermediate value since he must know exactly which portion of the key it depends on. Then, for each message used during the power trace recording and for each possible value that can be assumed by the selected key portion, the intermediate value is computed. The result of this operation is a matrix V whose dimension is N x P[8]. N represents the number of possible messages and P is the possible values that the key portion can have. In this way the element $v_{i,j}$ is the intermediate value produced by the device when it receives in input the $i^{th}$ message and the key value $j$. If the attacker wants to use the Hamming distance model, two matrices V has to be produced, one for each intermediate value, otherwise a unique matrix is sufficient.

### 4.3.5 Power consumption prediction

This phase consists in predicting the power consumption of the device during the production or saving of the chosen intermediate value for every possible couple (message,key). This can be translated in the production of a matrix H where the element $h_{i,j}$ of the matrix H is obtained by applying the chosen power model to the intermediate value $v_{i,j}$ of the matrix V[8]. As a consequence, the dimension of the matrix H is the same of the matrix V.

$$h_{i,j} = PowerModel(v_{i,j}) \ \forall i = 1...N, \forall j = 1...P \tag{4.12}$$

### 4.3.6 Correlation trace generation

This is the step demanding the most quantity of time and computing power since the correlations between the predicted power consumptions and the measured ones are calculated. Starting from the power trace matrix T and the predicted power consumption matrix H, the correlation matrix R is built where each element $r_{i,j}$ is computed using the formula shown in equation 4.13 [8].

$$r_{i,j} = \frac{\sum_{n=1}^{N}(h_{n,i} - \bar{h}_i) \cdot (t_{n,j} - \bar{t}_j)}{\sqrt{\sum_{n=1}^{N}(h_{n,i} - \bar{h}_i)^2 \cdot \sum_{n=1}^{N}(t_{n,j} - \bar{t}_j)^2}} \tag{4.13}$$

This is the application of the Pearson correlation coefficient formula shown in equation 4.15 where X and Y are two random variables, $x_i$ and $y_i$ are single measurements of those variable and $\bar{x}$ and $\bar{y}$ stand for their mean.

$$\rho(X,Y) = \frac{Cov(X,Y)}{\sqrt{Var(X) \cdot Var(Y)}} \tag{4.14}$$

$$\rho(X,Y) = \frac{\sum_{n=1}^{N}(x_n - \bar{x}) \cdot (y_n - \bar{y})}{\sqrt{\sum_{n=1}^{N}(x_n - \bar{x})^2 \cdot \sum_{n=1}^{N}(y_n - \bar{y})^2}} \tag{4.15}$$

This coefficient is an adimensional number whose value is between -1 and 1 that represents the linear relationship between the two variables. When the correlation coefficient is positive, the datasets of the 2 variables are directly proportional, which means that if the value of X increase, also the value of Y increase; instead if the coefficient is negative, the datasets of the two variables are inversely proportional. The remaining option is when the correlation is equal to 0 and that means that there no correlation between them. A graphical example about the meaning of the correlation coefficient is shown in Figure 4.6.
In the case of the CPA attack, the variable X is a column of the matrix H which is the power consumption prediction with a fixed key and a variable message, while the variable Y is a column of the matrix T which represents the measured power consumption at a certain instant of time $t$ in every power trace.
Each row of the matrix R is a *correlation trace* and it is referred to the key value equal to the index of the row. In particular the $k^{th}$ correlation trace represents the correlation in time between the measured traces and the predicted power consumption computed assuming that the key value is $k$.

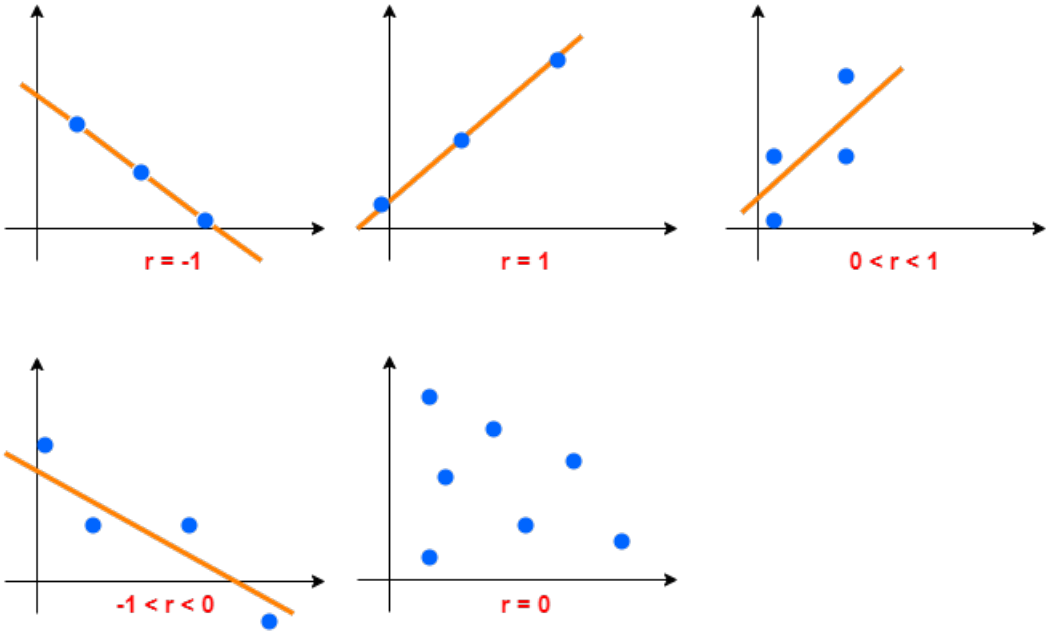
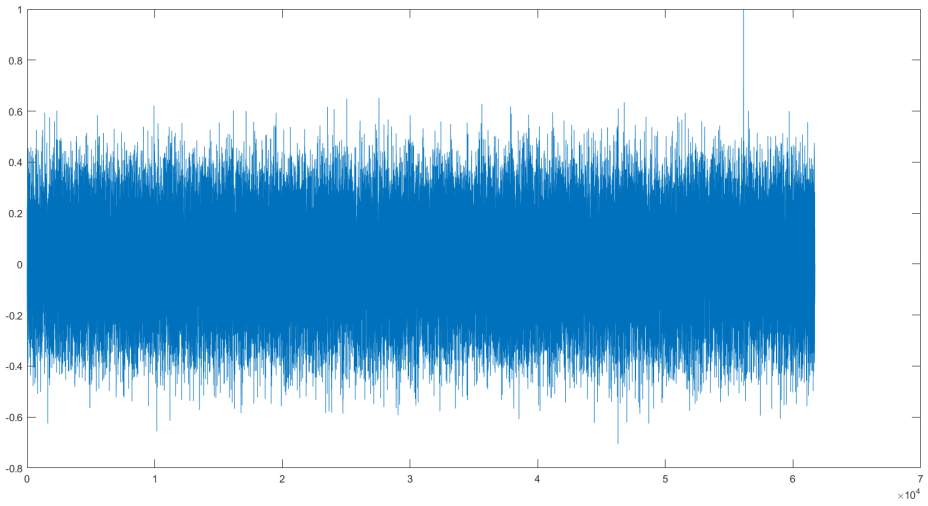


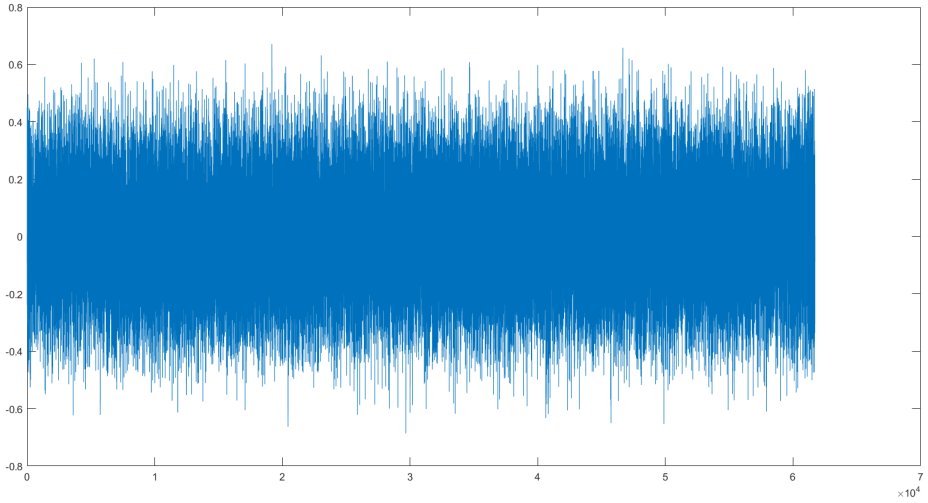

Figure 4.6: Graphical representation of the correlation coefficient meaning.

### 4.3.7 Results evaluation

Once the matrix R has been produced, the guess on the correct key can be done by plotting all the correlation traces. The one related to the correct key value will show a strong correlation peak in a certain instant of time, while all the others will be almost flat or just noise as shown in Figure 4.7. The correlation peak gives to the attacker also another important information: the time in which the target intermediate value has been produced or used inside the device. In some particular cases, depending on the choice of the intermediate value, the structure of the architecture and the cryptographic algorithm, more than one correlation trace can show a spike. When this happens, it's up to the attacker to identify which is the correct value based on the information that he has on the architecture of the device and on the algorithm.
Moreover, another fundamental parameter which determines the outcome of the attack is the number of traces needed to perform it successfully. In fact, the robustness of a device from a cryptographic point of view is directly proportional to the number of traces that are needed to successfully attack it. This is due to the fact that by increasing the number of traces, the variance of the noise superimposed to the measurement is decreased of a factor $\sqrt{N}$.




(a) *Correct key correlation trace.*




(b) *Incorrect key differential trace.*

Figure 4.7: Difference between the correlation trace of a correct key value and an incorrect one.K:rifare

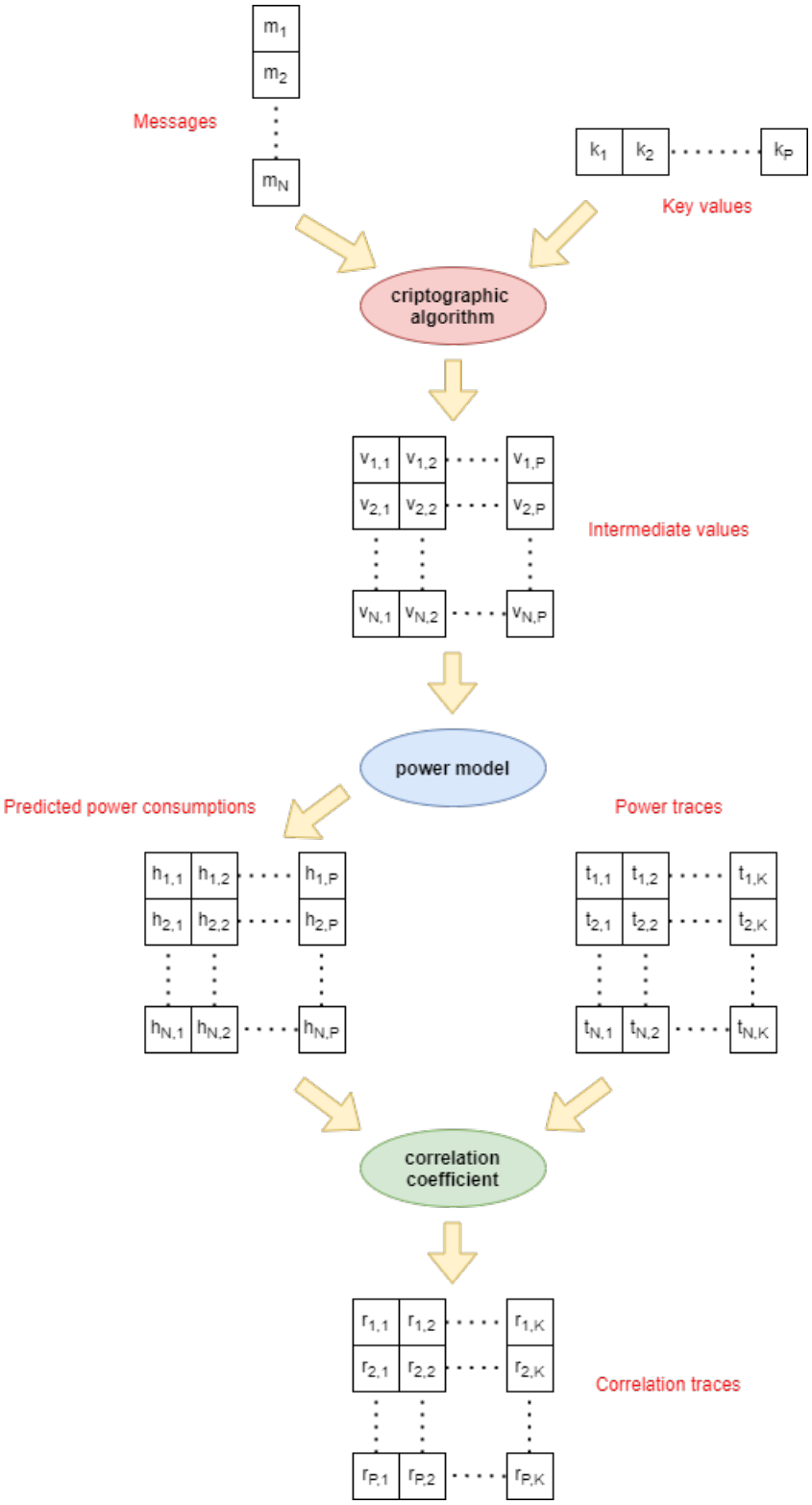



Figure 4.8: CPA attack steps.

# Chapter 5

# Methodology

The goal of this work is to create a methodology that could be used by designers who work in the cryptographic field to test if their design is safe against Power Analysis Attacks before the physical production of the device. This could save a huge amount of time and money to manufacture a new and safe product.
The method has been thought in order to be integrated within the ASIC design flow visible in Figure 5.1, in particular it should be applied during the *Logical Verification and Testing* phase since it requires the use of the post-synthesis netlist. In order to test its validity, it has been adopted to assess the side-channel leakage of the Vector By Circulant multiplier described in Chapter 3, which means verify if it is possible to retrieve one of its inputs by analyzing the power consumed during the multiplication. Since the inputs are an encrypted message and a private key, the device can be considered vulnerable if it is possible to guess the entire private key.
This methodology consists of 2 phases:

1. **Power traces generation**

2. **Simulation of a Power Analysis Attack**

## 5.1 Power Traces generation

The first phase consists in the generation of the power traces of the device, which means collecting its power consumption over the time in which it is performing some cryptographic operation.
The number of power traces that must be produced depends on the attack that the designer wants to simulate next: for a Simple Power Analysis attack only a power trace is needed while Differential Power Analysis attacks and Correlation Power Analysis attacks require multiple traces. To test the side-channel leakage of the multiplier it has been decided to perform a Correlation Power Analysis attack, so multiple traces were needed. The minimum number that allows to obtain valid results has been discovered

to be 30 and so the attack has been executed using 30 different traces to minimize its execution time.
Each power trace is referred to a multiplication executed with the same private key and a different message. The messages have been generated randomly, while the private key is still random but has all the characteristics necessary to be considered a possible real private key. Since the device has not been physically produced, the power measured in this step is computed with the help of two tools: **QuestaSim** developed by Mentor Graphics and **PrimeTime** developed by Synopsys. The generation of the power traces has been automatized by the use of a script written in Python which launches in sequence firstly QuestaSim and immediately after Primetime, as shown in Figure 5.2. The number of traces that will be generated is a parameter of the script so it can be easily modified depending on the needs of the designer.

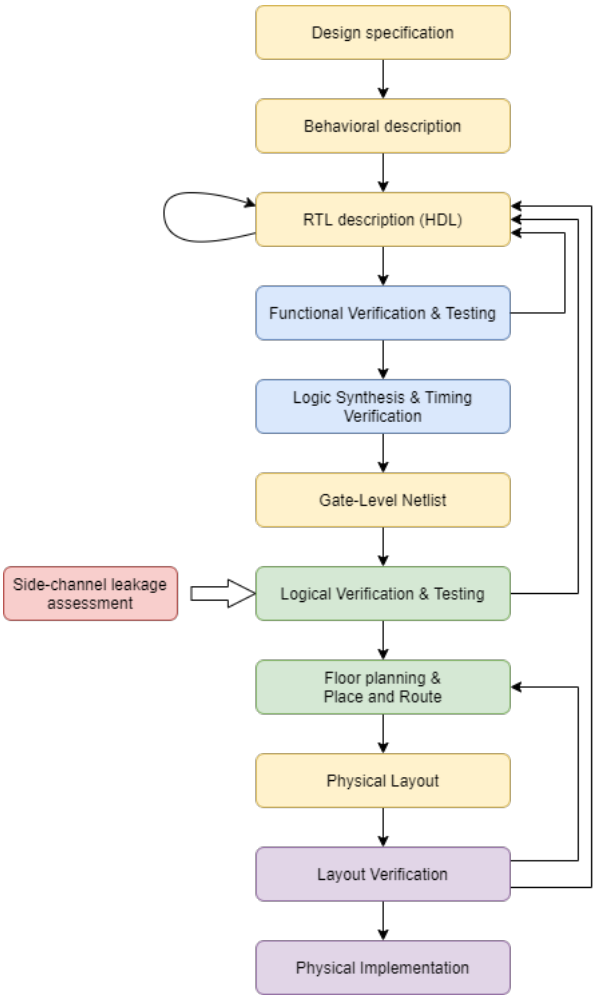



Figure 5.1: ASIC design flow.

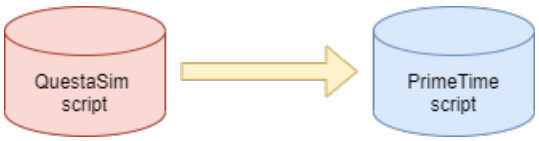



Figure 5.2: Sequence of tools used to produce the power traces.

### 5.1.1 QuestaSim script

QuestaSim is used to simulate the post-synthesis netlist and to produce in output the Value Change Dump file. This file contains the time in which each signal present in the post-synthesis netlist changes during the simulation and the new value that it assumes. Those information will be used by PrimeTime to compute the instantaneous power consumption of the device.
The inputs of this script are:

- the **post-synthesis netlist**;
- the **inputs of the device** to be used during the simulation. For the multiplier they consists in an encrypted message and a private key. Those inputs are stored in separate files that will be read at the beginning of the simulation and their content will be stored into the appropriate memories. In particular the message is composed of 14939 bits divided in word of 8 bits equal to the length of each memory cell, while the key is composed of 11 values each of them representing the position of a 1 in the first row of the key matrix. For this reason the value that each position can assume goes from 0 to 14938 and they are all different from the others;
- the **testbench**. It consists of the vhdl files that are used to provide stimulus to the netlist during the simulation;
- the **technological library** used for the synthesis. It contains all the basic logic cells that have been adopted by the synthesizer to produce the post-synthesis netlist. The library used in this test is composed of cells developed using a 65 nm technology;
- the **Standard Delay Format file** produced during the synthesis. It contains all timing information about the netlist which means path delays, timing constraint values, interconnection delays and high level technology parameters.

The script executes the following steps, visible also in Figure 5.3:

1. compilation of the testbench files;
2. compilation of the post-synthesis netlist;

3. compilation of the technological library;
4. simulation initialization;
5. simulation run until the files containing the inputs of the netlist are fully read and their content loaded into the appropriate memories;
6. creation of the VCD file and its initialization with all the needed signals;
7. simulation run until the multiplication is done;
8. simulation end.

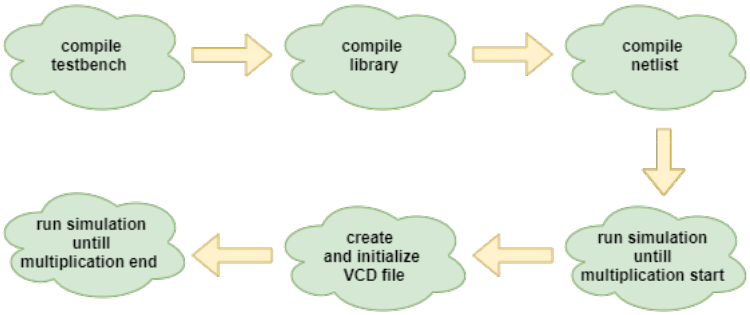



Figure 5.3: Sequence of operations executed by the QuestaSim script.

### 5.1.2 PrimeTime script

The PrimeTime script is launched immediately after the QuestaSim one terminates. This tool is employed in order to compute the power consumed by the netlist during the simulation time, which is also called *Time Based Power Analysis*. The inputs of this script are:

- the **post-synthesis netlist**;
- the **technological library**;
- the **VCD file** produced by the QuestaSim script;
- the **SDF file** created during the synthesis;
- the **Synopsys Design Constraint file** generated during the synthesis. This is an ASCII file containing design constraints and timing assignments given in input to the synthesizer.

The script perform the following steps, visible also in Figure :

1. read the technological library;

2. enable the power analysis feature;
3. set the power analysis mode to time based. In this way the tool calculates the power for every event present in the switching activity file, which is the VCD file, to generate power waveforms over time;
4. activate the *Delay-Aware Peak Power Analysis*. This setting allows to obtain more accurate peak power numbers since in this way the tool shifts all the events by a delay factor computed by looking at the cell or net delay stored in the technological library. If this feature is not enabled, the instantaneous peak powers are lowered because the leakage and dynamic energy associated to any event is distributed evenly over the clock period. This is due to the fact that in the switching activity files the signal transitions occurs only during the clock edges and so the events are all aligned and considered instantaneous;
5. set the effort level of delay shifting to high. In this way PrimeTime chooses the cell-arc, and as a consequence the shifting delay, based on the transition that occurs in the cell and this allows to obtain the most accurate power consumption values. Instead, if the effort level is set to low, the tool always chooses the cell arc with the worst delay;
6. read the post-synthesis netlist;
7. create the reference clock;
8. read the SDF file;
9. read the SDC file;
10. read the VCD file;
11. perform the power analysis;
12. create the power waveform file. If the resolution is not specified, PrimeTime computes the power at the same resolution of the switching activity file, otherwise it sums up the energy of all the events that occur inside the resolution time interval and distributes the energy evenly within it. The resolution has been set equal to the clock period in order to obtain reliable power values and reduce as much as possible the file dimension.

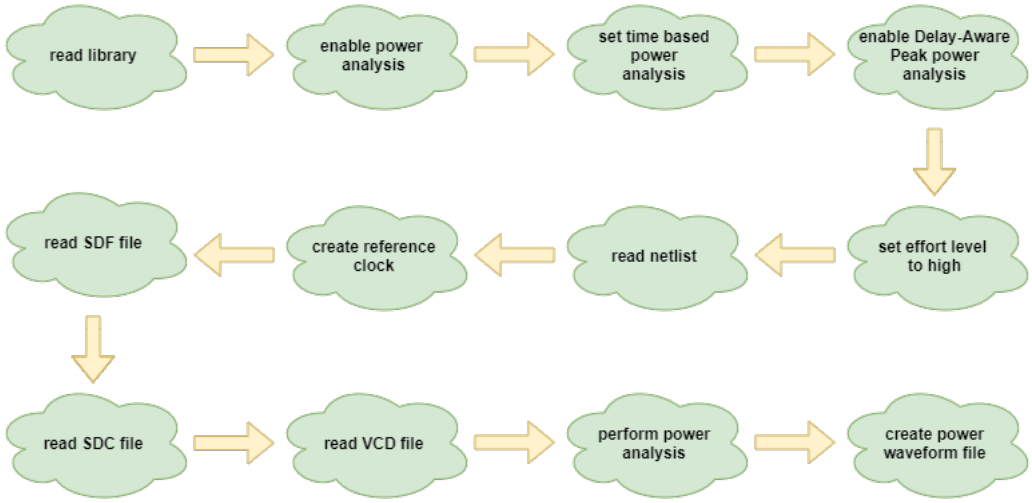



Figure 5.4: Sequence of operations executed by the PrimeTime script.

Figure 5.5 and Figure 5.6 shows one of the multiplier power waveforms produced by PrimeTime plotted using Matlab. The former represents the power consumption over the entire multiplication, while the latter is the zoom over a small portion of Figure 5.5 to better show each sample of power consumption computed by the tool.

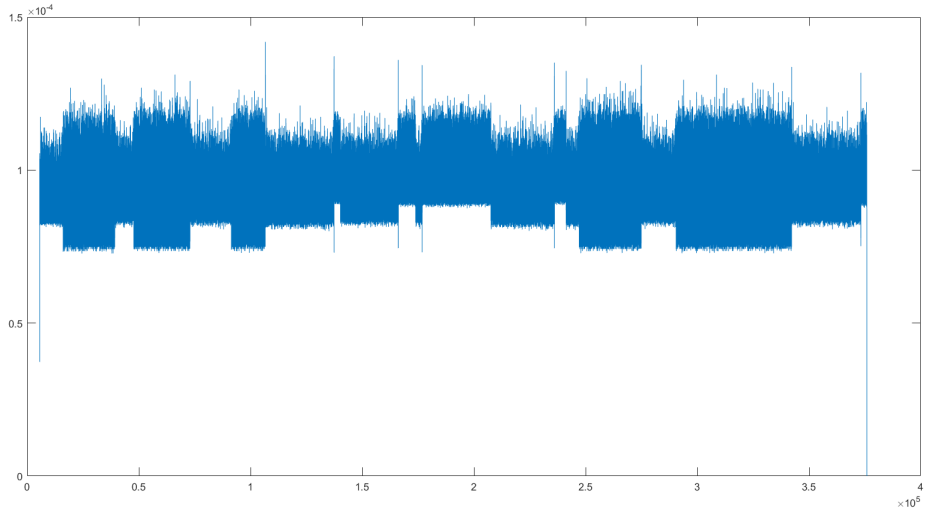



Figure 5.5: Power trace produced by PrimeTime representing the power consumption of the polynomial multiplier during the whole multiplication.

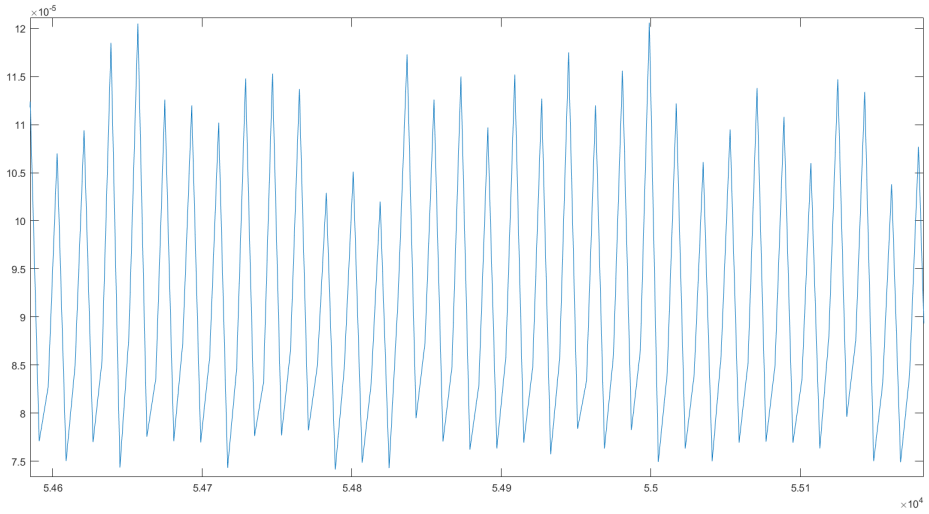



Figure 5.6: Small portion of the multiplier power trace shown in Figure 5.5.

## 5.2 Simulation of a Power Analysis Attack

After the generation of the needed power traces the next step is to simulate a Power Analysis Attack on the netlist using the power waveforms previously produced. In this way it is possible to understand if the architecture firstly designed and then sinthetized using a specific technological library leaks or not confidential informations, such as in the case of the multiplier, the private key of the code.
As described in Chapter 4, there are several types of attacks that exploits the power consumption behaviour of the device over time, but the one that has been chosen is the ***C****orrelation* ***P****ower* ***A****nalysis* attack. The attack has been developed in an iterative way, which means that for each iteration a single key position is guessed. Since the key is composed by 11 positions, 11 iterations are necessary to guess the entire key. The attack si composed of the following steps and they are executed following the flow shown in Figure 5.7

1. **choice of a power model**;
2. **choice of an intermediate value**;
3. **computation of the intermediate value**
4. **power consumption prediction**;
5. **correlation traces generation**;
6. **results evaluation**.

In the following it is described how each step of the attack has been performed and the reasons behind the decisions that have been made.

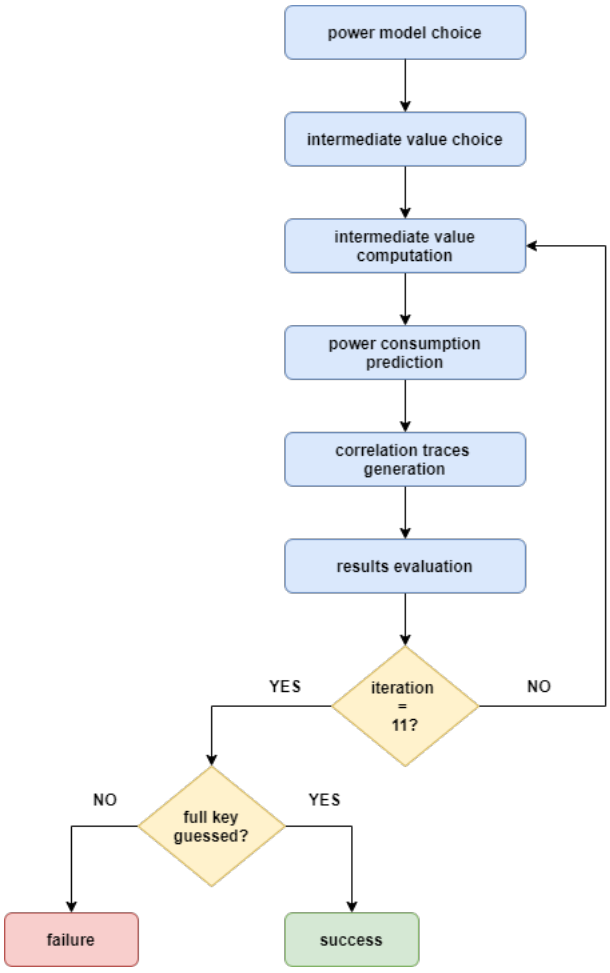



Figure 5.7: Execution order of the attack steps done on the multiplier.

### 5.2.1 Power model choice

The first step consists in the choice of a suitable power model that approximates in the most realistic way the instantaneous power consumed by the device. When a CPA or DPA attack has to be performed on a hardware implementation of an algorithm, the power model that guarantees the best results is the *Hamming distance* because it allows to model in a very precise way the power consumed by logic cells. Since the Hamming distance model expresses the power as a scalar number corresponding to the number of bits that changes between 2 binary numbers; if those two numbers are 2 consecutive values that are stored in the same register, this model becomes very accurate in representing the dynamic power consumed by the register when switching from a value to another one.

### 5.2.2 Intermediate value choice

Since the chosen power model is the Hamming distance, two intermediate values need to be selected. The ones that have been chosen for this attack are the first and second values stored in the *SynNew* register in each key position cycle, as shown in Figure 5.8. They are the result of the bit-xor between the value produced by the collapse unit and the value coming from the Syndrome memory, which is the one where every partial product is stored until the final result is produced.
The choice fell on those values because they have both a message and key dependency. The message dependency is justified by the fact that one of the values given in input to the xor gates is the result produced by the collapse unit which is always a message word. Instead, the key dependency comes from the fact that the portion of the message generated by the collapse unit depends on one of the key positions. These dependencies are even more emphasized if also the other input of the xor gates is taken into consideration because its value, being a partial product, has a dependency with both the message and previous positions of the key (with the exception for the first key position in which the content of the Syndrome memory is set to 0).

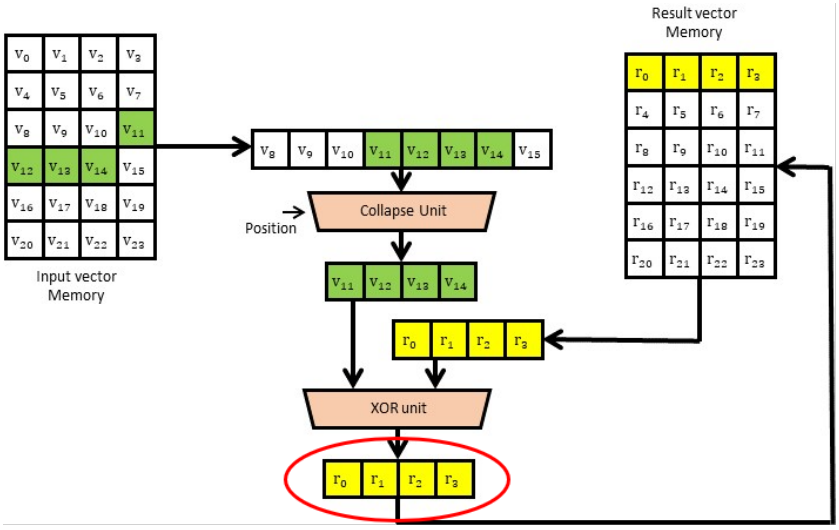



Figure 5.8: Architecture of the polynomial multiplier where it is shown the register whose content has been chosen as intermediate value to be attacked.

### 5.2.3 Intermediate value computation

This step consists in the computation of the chosen intermediate values for each possible couple message-key position. Since the messages are 30 and each key position can assume 14939 values, at each iteration of the attack 2 V matrices of size 30x14939 are computed as shown in Figure 5.9.

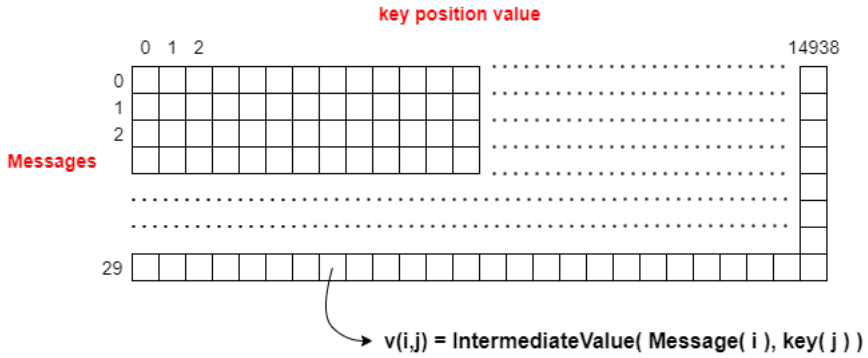



Figure 5.9: Structure of one of the V matrices

### 5.2.4 Power consumption prediction

This step consists in generating a unique matrix H starting from the two intermediate values matrices V. Each element in the H matrix is obtained by computing the Hamming distance between the elements in the same position in the two V matrices as shown in Figure 5.10.

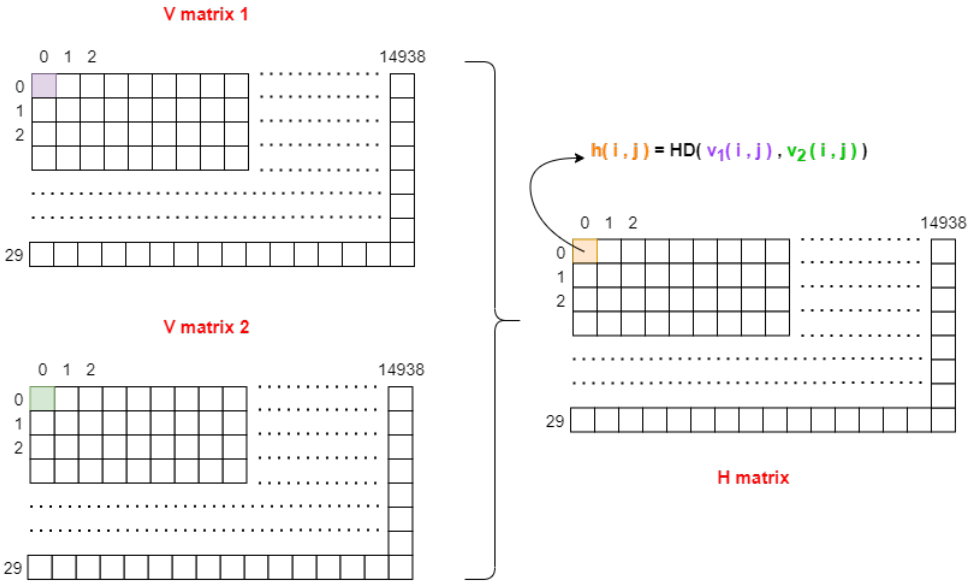



Figure 5.10: Creation of the H matrix starting from the two V matrices.

### 5.2.5 Generation of correlation traces

This step consists in the generation of the correlation matrix R starting from the power prediction matrix H and the power traces following the procedure described in section 4.3.6. The power traces can be grouped together in order to form a unique matrix called T where each row is one of them. Since the dimension of the matrix H is 30 x

14939 and the power trace matrix T is 30 x 61719, the correlation matrix dimension is 14939 x 61719 as shown in Figure 5.11.

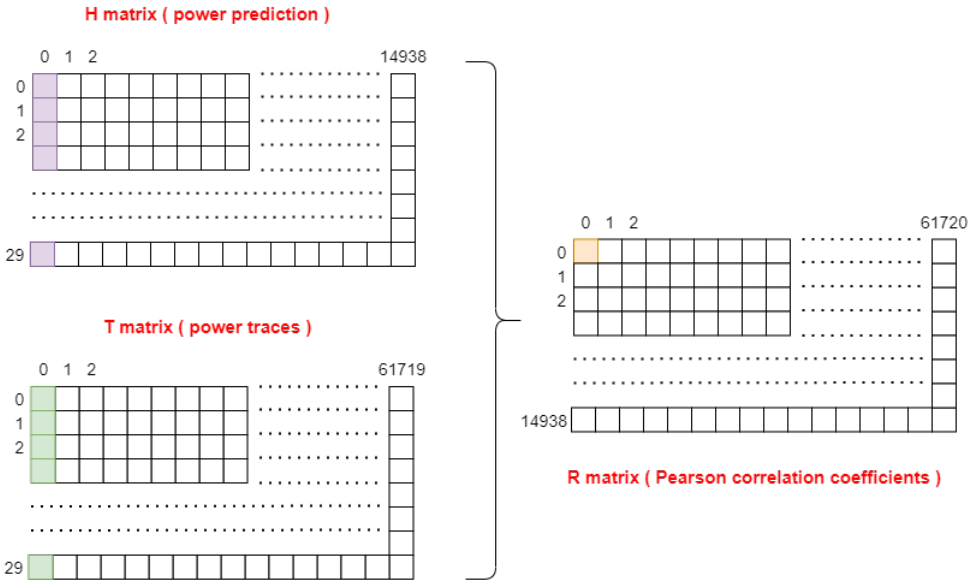



Figure 5.11: Creation of the R matrix starting from the H matrix and T matrix

### 5.2.6 Results evaluation

The results that has been obtained for the first key position are different from the ones obtained for the successive positions, but at the end the attack was successful since the entire private key has been correctly guessed. Further details about the obtained results is given in the following.

#### 1st position

For the first key position there are several traces with a correlation peak very close to 1 (only one in each trace) and so the corresponding key values are all candidates to be the correct guess.

In Figure 5.12 are shown in a unique plot all the previously mentioned correlation traces. It is clearly visible that all those traces have a correlation peak immediately at the beginning of the simulation and this is correct since the target key position is the first one and the chosen intermediate values are the first two words of the first partial product produced by the architecture. Figure 5.13 has been obtained by zooming in the zone where all the correlation peaks are grouped. It can be noticed that they are all distanced by 3 clock cycles and those clock cycles represents the time needed by the architecture to produce a word of the partial product (a micro-cycle). In fact all the peaks corresponds to the moments in which the value stored in the SynNew register is overwritten by the new one just produced.

The peak corresponding to the correct key is the first one of the series, all the other ones must be discarded. The reason behind this statement and the strange results obtained can be explained by the fact that the multiplier has a cyclic architecture and that the partial product words produced during the first macro-cycle are always a rotation of the input message. This is because, since it is the first cycle, the Syndrome memory is initialized to 0 and all its contributions are irrelevant, so the xor gates can be seen just as buffers that propagates the result coming from the collapse unit. In particular: let's assume that the correct key value is $k$. The first value to be written into the SynNew reg will be the Message bits whose index are between $k$ and $k+7$ that for seek of practice are indicated as $Message[k, k+7]$. After that, the value that will overwrite it will be $Message[k+8, k+15]$ and so on. The exclusive or between $Message[k, k+7]$ and $Message[k+8, k+15]$ will produce the target intermediate value of the attack and the CPA algorithm will find a strong correlation in the clock cycle $t$. If the correct key value would have been $k+8$, the value that will be searched by the CPA algorithm will be the exclusive or between $Message[k+8, k+15]$ and $Message[k+16, k+23]$. In the power trace, where the used key is $k$, it is be produced in the clock cycle $t+3$ and, as a consequence, also the correlation peak will be in that position.
In conclusion, all the potential correct keys found by the algorithm are:

- the correct key value;
- all key values obtained by summing multiples of 8 (the parallelism of the architecture) to the correct one until reaching the maximum value.

As said before the correct guess can be made by considering only the correlation trace that shows a strong correlation the closest to the beginning of the simulation. To confirm this statement an additional verification has been done by checking in the QuestaSim simulation that the moment in time of the correlation peak corresponds to the instant in which the SynNew register is overwritten with the second value.

### 2nd position

For the second key position the result is univocal since only one correlation trace presents a strong correlation value(almost 1) and the key value associated to that trace is the correct one. An example of the resulting correlation trace is shown in Figure .

### 3rd position

For what concerns the third key position guess, the number of correlation traces showing a correlation peak very close to 1 can be:

- **2** for most of the cases;
- **3** in rare situations.

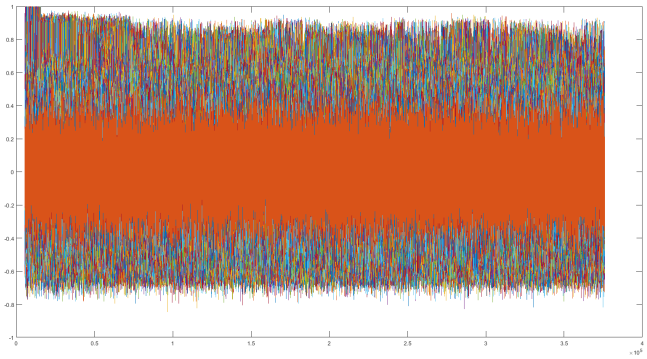


Figure 5.12: Plot showing all the correlation traces containing a strong correlation value for the guessing of the key position 1.

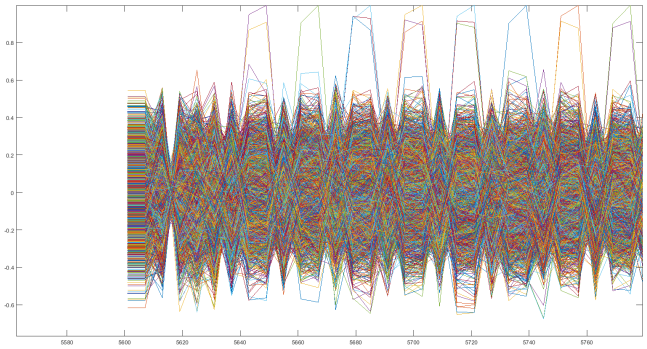


Figure 5.13: Zoom of Figure 5.12 in the part containing the highest correlation values.

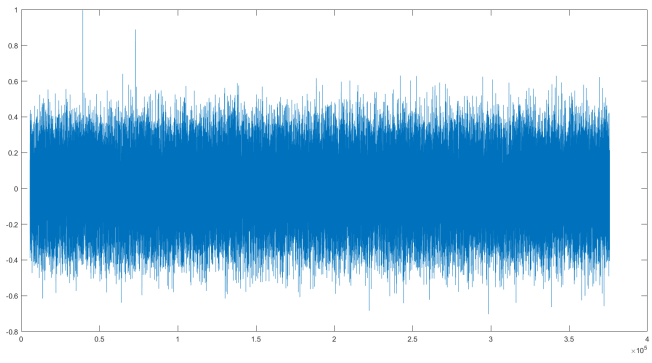


Figure 5.14: Plot showing the correlation traces having a strong correlation peak during the guessing of the key position 2.

In the first scenario the key values related to the relevant traces are the correct one and the one guessed for the second position. In the former trace the peak appears in the right position, while the one related to the incorrect value is in the instant of time where the intermediate values in the 1st macro-cycle are saved in the SynNew register. In order to explain the reason behind this behavior let's assume that the correct key values in the first 3 macro-cycles are respectively $k$, $j$ and $n$.
Following this assumption, the first 2 values that are written in the SynNew register in each cycle are the one shown in Table and Table .

| **Position** | **Key value** | **SynNew value 1** |
|---|---|---|
| 1 | $k$ | $Message[k, k+7]$ |
| 2 | $j$ | $Message[j, j+7] \oplus Message[k, k+7]$ |
| 3 | $n$ | $Message[n, n+7] \oplus Message[j, j+7] \oplus Message[k, k+7]$ |

Table 5.1: First value written in the SynNew register during the first 3 cycles.

| **Position** | **Key Value** | **SynNew value 2** |
|---|---|---|
| 1 | $k$ | $Message[k+8, k+15]$ |
| 2 | $j$ | $Message[j+8, j+15] \oplus Message[k+8, k+15]$ |
| 3 | $n$ | $Message[n+8, n+15] \oplus Message[j+8, j+15]$ $\oplus Message[k+8, k+15]$ |

Table 5.2: Second value written in the SynNew register during the first 3 cycles.

When the algorithm computes the correlation trace for a certain key value, it assumes that the correct value is the one under analysis. So, when it is evaluating the correlation trace for the key value $j$, the first two values written in the SynNew register become:

$$\begin{cases} n = j \\ SynNew\_value\_1 = Message[n, n+7] \oplus Message[j, j+7] \oplus Message[k, k+7] \\ SynNew\_value\_2 = Message[n+8, n+15] \oplus Message[j+8, j+15] \\ \qquad\qquad\qquad \oplus Message[k+8, k+15] \end{cases}$$

$$\Downarrow$$

$$\begin{cases} m = n \\ SynNew\_value\_1 = Message[j, j+7] \oplus Message[j, j+7] \oplus Message[k, k+7] \\ SynNew\_value\_2 = Message[j+8, j+15] \oplus Message[j+8, j+15] \\ \qquad\qquad\qquad \oplus Message[k+8, k+15] \end{cases}$$

$$\Downarrow$$

$$\begin{cases} m = n \\ SynNew\_value\_1 = Message[k, k+7] \\ SynNew\_Value\_2 = Message[k+8, k+15] \end{cases}$$

Those values are equal to the ones produced during the first key position cycle and that is why the algorithm finds a strong correlation in the clock period when the first SynNew value is overwritten by the second one in the first macro-cycle. In the case in which the algorithm produce in output 3 possible candidates for the correct key value, the additional value is the correct 1st key position. This is due to the fact that, maintaining valid the previous assumption for what concerns the correct values in the first 3 macro-cycles, when the algorithm computes the correlation trace for the key value $k$ it assumes that the two values written in the SynNew register are:

$$\begin{cases} n = k \\ SynNew\_value\_1 = Message[n, n+7] \oplus Message[j, j+7] \oplus Message[k, k+7] \\ SynNew\_value\_2 = Message[n+8, n+15] \oplus Message[j+8, j+15] \\ \qquad\qquad\qquad \oplus Message[k+8, k+15] \end{cases}$$

$$\Downarrow$$

$$\begin{cases} m = n \\ SynNew\_value\_1 = Message[k, k+7] \oplus Message[j, j+7] \oplus Message[k, k+7] \\ SynNew\_value\_2 = Message[k+8, k+15] \oplus Message[j+8, j+15] \\ \qquad\qquad\qquad \oplus Message[k+8, k+15] \end{cases}$$

$$\Downarrow$$

$$\begin{cases} m = n \\ SynNew\_value\_1 = Message[j, j+7] \\ SynNew\_Value\_2 = Message[j+8, j+15] \end{cases}$$

If those two values are really saved in the register during the multiplication the algorithm will find a strong correlation and identify the key value $k$ as a possible candidate. This correlation can only be found in the simulation time in which the first key position is managed since the content of the SynNew register is composed by a single contribution. Although this situation can happen only if two conditions are satisfied:

- the difference between the 1st ($k$) and 2nd ($j$) position values is a multiple of the parallelism of the architecture.
- the 2nd key value must be greater than the 1st one.

If those conditions are not satisfied, the bits that compose $Message[j, j+7]$ are splitted and inserted in two consecutive values produced during the macro-cycle managing the $1^{st}$ key position. The same consideration can be done for $Message[j+8, j+15]$. Independently on the number of possible candidates found by the algorithm, the correct value can simply be found by exclusion since the other values have already been guessed and there cannot be two identical values in the key.

### Other positions

For the remaining positions there are always 2 correlation traces showing a strong peak close to 1. The key values associated to those traces are the correct one and the correct value of the previous position. In the former trace the peak appears in the right position, while the one related to the incorrect value is in the instant of time where the intermediate values of two macro-cycles ago are saved in the SynNew register. In order to better understand why an example is reported.
Let's assume that the key position to be guessed is the $4^{th}$ one whose value is $m$ and the previously guessed values are: $k$ for the first position, $j$ for the second one and $n$ for the third one. In Table 5.3 and Table 5.4 are reported the first 2 values that will be stored in the SynNew register during the 4 macro-cycles.

| **Position** | **Key value** | **SynNew value 1** |
|---|---|---|
| 1 | $k$ | $Message[k, k+7]$ |
| 2 | $j$ | $Message[j, j+7] \oplus Message[k, k+7]$ |
| 3 | $n$ | $Message[n, n+7] \oplus Message[j, j+7] \oplus Message[k, k+7]$ |
| 4 | $m$ | $Message[m, m+7] \oplus Message[n, n+7] \oplus Message[j, j+7]$ $\oplus Message[k, k+7]$ |

Table 5.3: First value written in the SynNew register during the first 4 cycles.

| **Position** | **Key Value** | **SynNew value 2** |
|---|---|---|
| 1 | $k$ | $Message[k+8, k+15]$ |
| 2 | $j$ | $Message[j+8, j+15] \oplus Message[k+8, k+15]$ |
| 3 | $n$ | $Message[n+8, n+15] \oplus Message[j+8, j+15]$ $\oplus Message[k+8, k+15]$ |
| 4 | $m$ | $Message[m+8, m+15] \oplus Message[n+8, n+15]$ $\oplus Message[j+8, j+15] \oplus Message[k+8, k+15]$ |

Table 5.4: Second value written in the SynNew register during the first 4 cycles.

So when the algorithm is evaluating the correlation trace for the key value $n$, the

first two values that it expects to be written in the SynNew register are:

$$\begin{cases} m = n \\ SynNew\_value\_1 = Message[m, m+7] \oplus Message[n, n+7] \oplus Message[j, j+7] \\ \qquad\qquad\qquad \oplus Message[k, k+7] \\ SynNew\_value\_2 = Message[m+8, m+15] \oplus Message[n+8, n+15] \\ \qquad\qquad\qquad \oplus Message[j+8, j+15] \oplus Message[k+8, k+15] \end{cases}$$

$$\Downarrow$$

$$\begin{cases} m = n \\ SynNew\_value\_1 = Message[n, n+7] \oplus Message[n, n+7] \oplus Message[j, j+7] \\ \qquad\qquad\qquad \oplus Message[k, k+7] \\ SynNew\_value\_2 = Message[n+8, n+15] \oplus Message[n+8, n+15] \\ \qquad\qquad\qquad \oplus Message[j+8, j+15] \oplus Message[k+8, k+15] \end{cases}$$

$$\Downarrow$$

$$\begin{cases} m = n \\ SynNew\_value\_1 = Message[j, j+7] \oplus Message[k, k+7] \\ SynNew\_Value\_2 = Message[j+8, j+15] \oplus Message[k+8, k+15] \end{cases}$$

It can be noticed that those two values are equal to the one present in the second row of Table 5.3 and Table 5.4 because they are the first two numbers written in the SynNew register during the $2^{nd}$ macro-cycle. That is why the algorithm finds strong correlation also in the clock period in which in the simulations the first number is overwritten by the second one. By the way this additional peak doesn't affect the final guess since the correct value can be found by simply excluding the values guessed during the previous cycles because there cannot be 2 identical values in the key.
An example of two relevant correlation traces found during the $11^{th}$ position guess is shown in Figure 5.15

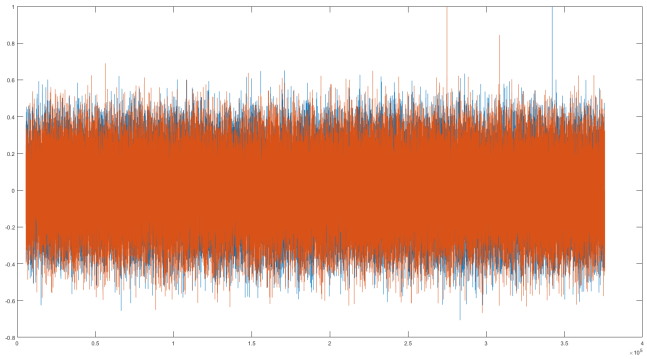


Figure 5.15: Plot showing the correlation traces having a strong correlation peak during the guessing of the key position 11.

## 5.3 Threshold frequency detection

A study has been conducted in order to find which is the minimum working frequency of the multiplier that allows to recover the secret key using the CPA attack described in the previous sections. In this case, instead of guessing the entire key, the target is only its second position. This choice has been made in order to replicate the attack conducted on a real device described in chapter 6 where also the motivations behind this choice are detailed. As shown in chapter 4 the dynamic power consumed by a digital system is directly proportional to its working frequency. Since the power contribution exploited by the CPA attack is the dynamic one it is reasonable to expect that the higher is the dynamic power, the higher will be the accuracy of the attack result. For accuracy it is meant the difference between the correlation peak of the correct key and the highest correlation peak coming from all the other key guesses which are wrong.
To find this threshold the same CPA attack has been reproduced by only changing the working frequency of the architecture, which means that:

- the post-synthesis netlist is the same and it has been produced giving the same clock constraint of the attack described in chapter 5, which is 200 MHz. It is important to not modify the clock frequency in the synthesis constraints because it would almost surely lead to some changes in the synthesized architecture;
- the clock frequency in the testbench files has been changed and has assumed a different value for each test conducted. This is the working frequency of the architecture during the simulations;
- the VCD files produced by QuestaSim and, as a consequence, also the power traces coming from PrimeTime will be different because of the different clock frequency set in the testbench files;

- the number of power traces used for the CPA attack has remained unchanged to 30, which is the minimum number that allows to obtain reasonable results.

### 5.3.1 Results

The results obtained are shown in Figure 5.16 and in Table 5.5.

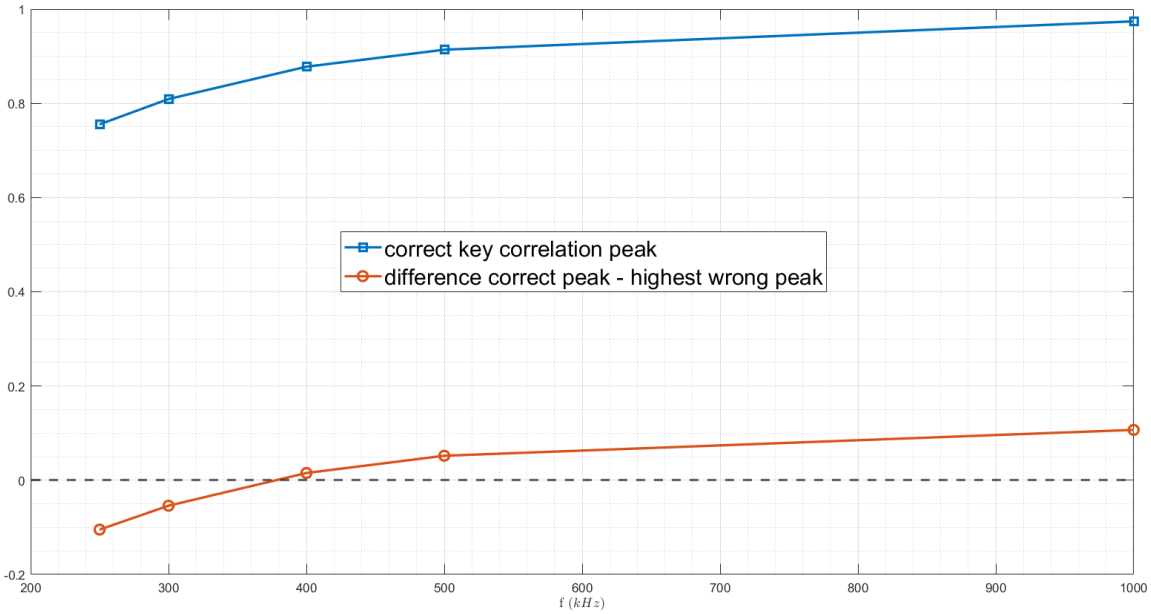



Figure 5.16: Graph representing the behaviour of the correlation peak of the correct key value and an indication on the accuracy of the attack depending on the working frequency of the multiplier.

| **Frequency** (kHz) | **Correct peak** | **Highest wrong peak** | **Accuracy** |
|---|---|---|---|
| 250 | 0.75543 | 0.8602 | -0.10477 |
| 300 | 0.80914 | 0.8632 | -0.05406 |
| 400 | 0.87777 | 0.862417 | 0.015353 |
| 500 | 0.91381 | 0.861798 | 0.052012 |
| 1000 | 0.97411 | 0.86723 | 0.10688 |

Table 5.5: Values of the correct key peak, the highest wrong key peak and the accuracy of the result depending on the working frequency of the multiplier.

The blue line represents the behaviour of the correlation peak of the correct key value, while the orange line corresponds to the accuracy of the attack. As expected, the higher is the frequency, the higher is the value of the peak. From the graph it can be noticed that the minimum working frequency that allows to recognize the correct

key value from the wrong ones is around 400 kHz where the difference between the peaks starts to be greater than 0 which means that the correct key has the highest peak. The behaviour of the two graphs is very similar because the value of the highest wrong peak is always around 0.86 and does not vary significantly with the frequency. Those results have to be intended as ideal since they come from simulations, so if the same experiment has to be repeated on a real device, the frequency threshold will surely be higher than 400 kHz due to the noise and various errors superimposed to the power trace signal coming from different sources like the instruments used for the measuring setup and from the environment. All those unwanted components will lower the correlation peak value and as a consequence more traces will be needed to make the CPA attack successful.

# Chapter 6

# CPA attack on a multiplier FPGA implementation

The last step of this work has been to apply the same CPA attack done in simulation to a real implementation of the multiplier. The only difference is that instead of recovering the entire key, the target has been only the second position since it is the one requiring the lower number of samples to be recorded from the power trace of the device allowing us to reduce as much as possible the required storage memory and the execution time of the attack.
The goal is to verify if the results obtained from the simulations are compliant with the one obtained in a real scenario. Both the measuring setup and the device under test used for this experiment are included into a unique board which has been developed by the Politecnico di Torino named *VirtLab*.
In the following sections will be described the architecture of the board [14], how the traces have been recorded and the results obtained.

## 6.1 VirtLab architecture

The board is divided in two sides:

- the **Master side** containing the test equipment and the logic to program the user side;
- the **User side** containing the device under test.

Both the User and Master sides contain an FPGA and a microcontroller together with some peripherals and LEDs. A block diagram representing the architecture of the board is shown in Figure 6.1.

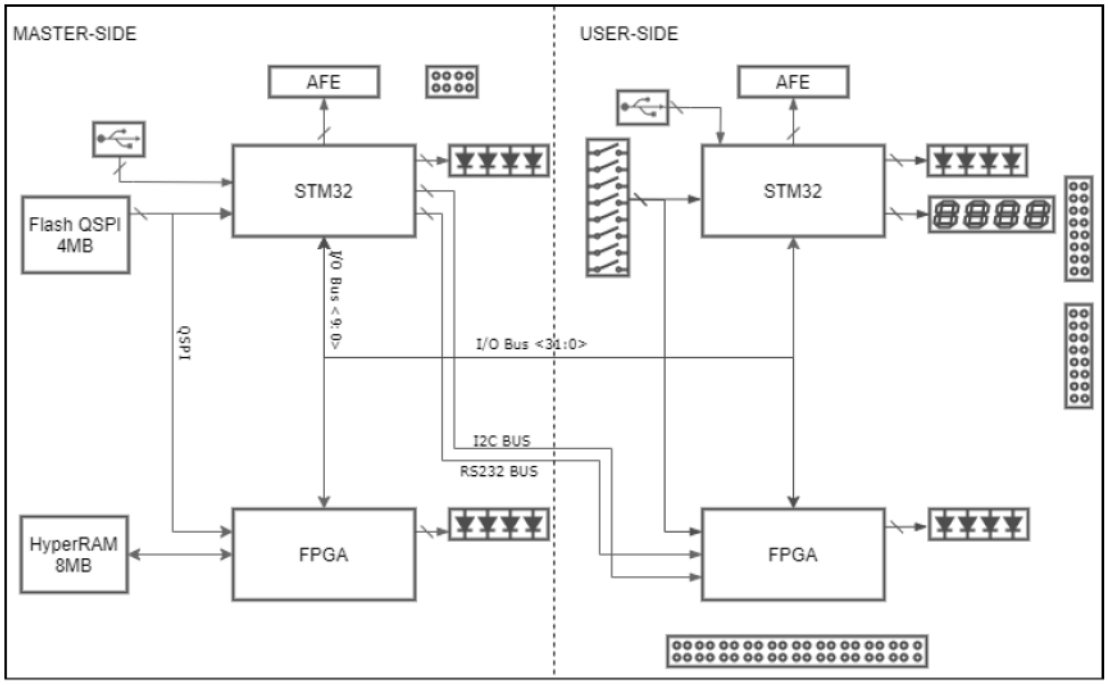



Figure 6.1: Block diagram of the VirtLab architecture.

### 6.1.1 Master side

The master side contains all the components that act as test equipment and control the User side elements. The measuring instruments integrated in the board are:

- a **Digital Storage Oscilloscope**;
- a **Signal generator**;
- a **Multimeter**;
- a **Logic Analyzer**.

The main components that are included in this side are:

- a 32 bit **microcontroller** that is in charge of programming the User FPGA by using an $I^2C$ bus or a RS232 bus and the Master FPGA thanks to a QSPI channel. Moreover it can be used to configure and send control commands to the components and its peripherals that act as test equipment and retrieve from them signals that can be sent to a PC using an USB connection;
- an **FPGA** connected to the main and only general purpose I/O bus of the board that connects together all the most important components in both the Master and User sides;

- an **Analog Front End** that is formed by the components in charge of converting digital signals into analog output and viceversa such as ADCs and DACs. In this way it is possible to emulate the functionalities of both a digital oscilloscope and of a signal generator.
- a **Hyper-RAM** which is used to store samples from both the oscilloscope and the signal analyzer;
- a **Flash QSPI** that contains the configuration file (.RBF) of the FPGA master and its content will be transferred to the FPGA at the startup of the board thanks to the Master microcontroller.

### 6.1.2 User side

The User side is the one where the system under test is implemented and its main components are:

- a **microcontroller** which is freely programmable and it is connected both to the User FPGA and to some useful peripherals like a 7-segment display and 4 LEDS;
- a **FPGA** that, as the microcontroller, is fully programmable and it is connected to 4 LEDS.

Both the microcontroller and the FPGA are connected to the 32 bits IO bus in order to communicate also with the components in the Master side.

## 6.2 Power trace recording

Since only the second value of the key had to be guessed, it has been recorded only the time frame where that value was used during the multiplication. The setup was organized as shown in Figure 6.2. The VirtLab was connected to the pc through a USB port which has been used to program both the microcontroller and the FPGA on the User side and to send commands to the Master microcontroller to setup correctly the oscilloscope [13]. The testbench components and the multiplier with its memories have been implemented as follows:

- the User MCU (*MicroController Unit*) contains the testbench. Its role is to send the control signals to the multiplier in order to start and stop correctly the operation;
- the User FPGA has been used to implement the multiplier together with all the memories that it needs to retrieve the input values and store the result.

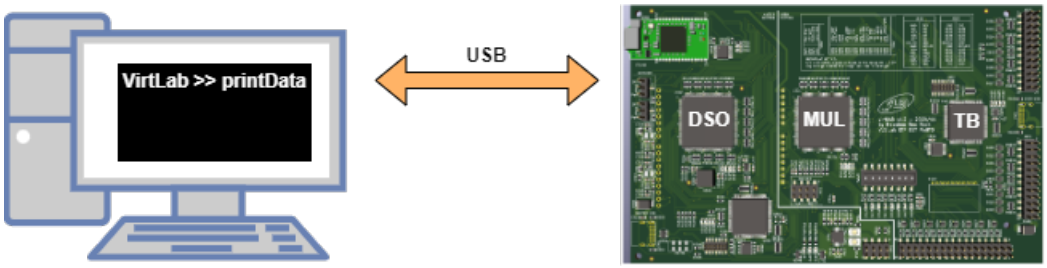



Figure 6.2: Measuring setup used to record power traces using the VirtLab board.

For what concerns the measuring equipment, the oscilloscope has two channel that can be connected to six possible sources:

- two general purpose analog channels;
- a signal representing the power consumption of the user FPGA on the 3.3V power line. This voltage is used to feed the I/O cells of the FPGA;
- a signal representing the power consumption of the user FPGA on the 2.5V power line. This line feeds the analog part of the PLLs present in the device;
- a signal representing the power consumption of the user FPGA on the 1.2V power line that feeds the core cells.

For our purpose the relevant sources are the 3.3V and the 1.2V that have been connected to the 2 channels of the oscilloscope. Then, the measuring procedure has been the following:

1. the User MCU has been programmed;
2. the User FPGA has been configured in order to have both the multiplier and the two input memories containing respectively the secret key and a message to decrypt;
3. the oscilloscope has been set up: the two channels were connected to the 3.3V and 1.2V power line of the FPGA and the sampling frequency was set to the double of the working frequency of the multiplier in order to obtain 2 samples for each clock cycle.
4. the testbench has been launched;
5. when the testbench is concluded the samples stored by the oscilloscope in both the channels were retrieved by transferring them through the USB port to the pc where they have been saved into text files.

This procedure has been repeated for each power trace collected and, when all the measures have been done, for each couple of traces (3.3V and 1.2V) related to the same message, they were summed up into a unique trace that has been used to perform the CPA attack.

## 6.3 Results

Due to the fact that the maximum sampling frequency that could be reached using the VirtLab board is 500 kSa/s, the working frequency of the multiplier has been set to 250 kHz in order to obtain two samples for each clock cycle. As expected from the results obtained in chapter 5.3, the working frequency of the architecture was not enough to make the attack successful. In this experiment 100 different messages has been used and for each of them 10 traces have been collected. The attack has been conducted firstly using for each message one of the 10 traces collected and, with the second try, the traces related to the same message have been previously mediated and the mean-trace produced was used for the attack. This procedure has been done to reduce the amount of noise superimposed to the signal. Since those attacks did not lead to good results, another approach was used for the last attempt. The idea was to delete from the power trace the static contribution or at least to reduce it a lot in order to maintain only the dynamic part. This was done by firstly computing a mean trace from the 100 messages and them subtract this contribute, that should be very close to the static power, to each of the 100 traces. The power profiles obtained from this operation should represent the dynamic power consumed by the architecture and have been used to perform the attack. However also this attempt was not successful.
As already stated in chapter 5.3, due to the fact that the threshold frequency in simulations is 400 kHz, if a real CPA attack has to be conducted on this architecture, its working frequency must be much higher in order to make the dynamic power significant with the respect to the sum of the static contribution and the noise superimposed to the signal, but with the hardware used in this experiment this was not possible.

# Bibliography


[1] Marco Baldi. *QC-LDPC code-based cryptography*. Springer Science & Business, 2014.

[2] Eric Brier, Christophe Clavier, and Francis Olivier. "Correlation power analysis with a leakage model". In: *International workshop on cryptographic hardware and embedded systems*. Springer. 2004, pp. 16–29.

[3] Jintai Ding and Bo-Yin Yang. "Multivariate public key cryptography". In: *Post-quantum cryptography*. Springer, 2009, pp. 193–241.

[4] Olaf Grote, Andreas Ahrens, and César Benavente-Peces. "A Review of Post-quantum Cryptography and Crypto-agility Strategies". In: *2019 International Interdisciplinary PhD Workshop (IIPhDW)*. 2019, pp. 115–120. DOI: 10.1109/IIPHDW.2019.8755433.

[5] Paul Kocher et al. "Introduction to differential power analysis". In: *Journal of Cryptographic Engineering* 1.1 (2011), pp. 5–27.

[6] Kristjane Koleci. "VLSI QC-LDPC Decoder for Post-Quantum Cryptography". MA thesis. Politecnico di Torino, 2019.

[7] Kristjane Koleci et al. "Efficient Hardware Implementation of the LEDAcrypt Decoder". In: *IEEE Access* 9 (2021), pp. 66223–66240.

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

[9] Baldi Marco et al. *LEDAcrypt: Low-dEnsity parity-check coDe-bAsed cryptographic systems*. 2019. URL: https://www.ledacrypt.org/documents/LEDAcrypt_v3.pdf.

[10] Robert J McEliece. "A public-key cryptosystem based on algebraic". In: *Coding Thv* 4244 (1978), pp. 114–116.

[11] Daniele Micciancio and Oded Regev. "Lattice-based cryptography". In: *Post-quantum cryptography*. Springer, 2009, pp. 147–191.

[12] Raphael Overbeck and Nicolas Sendrier. "Code-based cryptography". In: *Post-quantum cryptography*. Springer, 2009, pp. 95–145.

[13] Massimo Ruo Roch. “Tutorial: Setting up and using VirtLab”. VirtLab tutorial. Nov. 2021.

[14] Massimo Ruo Roch. “VirtLab 1.2”. VirtLab documentation. Apr. 2021.

[15] Katsuyuki Takashima. “Efficient algorithms for isogeny sequences and their cryptographic applications”. In: *Mathematical modelling for next-generation cryptography*. Springer, 2018, pp. 97–114.