POLITECNICO DI TORINO

Master Degree Course in Electronic Engineering

Master Degree Thesis

Evaluation of Encryption Algorithm Security in Heterogeneous Platform against Differential Power Analysis Attack

Supervisor:

Prof. Stefano DI CARLO

Candidate:

Fiammetta VOLPE
Student ID: 235145

A.A. 2017/2018

Summary

An embedded system security can be violated at different levels of abstraction: the vulnerability is not only present from software point of view, but also the hardware can be attacked.

This thesis is focused on an hardware attack at logic/microelectronic level called Differential Power Analysis (DPA), included in the larger categories of the Power Analysis (PA) and Side-Channel Analysis, catalogued like a passive and non-invasive attack, since it includes the observation of the normal behaviour of the device without any physical alteration. As a consequence, this kind of attack could be extremely dangerous and it doesn't leave any trace.

A DPA attack is essentially based on the principle that the power consumption is correlated to the activity of the device during data encryption, so also to the used encryption key. Thus, using statistical analysis on a sufficiently large number of power traces, it is possible to detect the correct hypothesis for the key.

Due to the improvement of FPGAs in terms of capacity and performance and the significant increment of the value of handled data, it is essential to do an analysis of the level of vulnerability of the device. For this reason, the MachXO2-7000 FPGA included, together with a STM32F4 CPU and a SLJ52G SECURITY CONTROLLER-SMART CARD, inside the BGA chip SEcube^{TM}, appositely designed for security goals, is the chosen target for this thesis.

The FPGA was programmed with a VHDL code of AES-256/AES-128 (both in ECB and CBC Block Cipher Modes), among the most used encryption algorithms for symmetric key cryptography. The verification of the correct behaviour and the simulations related to the used area and power consumption were reported.

Later, all the possible power supply accesses of the given board were searched and power traces were acquired from a LeCroy waveRunner 6030 Digital Oscilloscope, making comparisons between them. For the acquisitions also a Tektronix TM502A Current Probe and an handmade USB-USB/BNC Connector were used.

Exploiting `Jlsca` DPA Tool, different versions of DPA were applied on the collected data, then the results were elaborated. To underline the most relevant results, a simple MATLAB code that compared every key hypothesis with the used one was written, calculating the difference function and the number of hit bits and plotting all the results and some bar diagrams to compare in particular the four different versions of the encryption (AES-256 CBC, AES-256 ECB, AES-128 CBC, AES-128 ECB) with all the different instrumentation used, the variations obtained changing the working frequency and the number of samples considered.

The main problem identified in the application of the DPA attack was the presence of decoupling capacitors in correspondence to the FPGA power supply accesses. Indeed, they are considered like a "natural" countermeasure against this kind of attack.

Finally these results were compared with the ones reported in literature, some conclusions were deducted and possible future works and improvements were proposed.

Acknowledgements

I want to express my biggest gratitude to all my family, near and far, for the unlimited patient and the unconditioned support.
I specially thank Gianfranco Albis for the precious help and for generously sharing his experiences and all the laboratory stuff, both from DET (Department of Electronic and Telecommunications) and from DAUIN (Department of Automation and Information).
I thank Prof. Stefano Di Carlo and Alberto Carelli for the guidance and the opportunity to work on this thesis.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 General Overview and Objectives

The main target of this thesis was getting an analysis of the level of vulnerability of the FPGA contained inside SEcube^{TM} board against a DPA attack.
First of all, due to the complexity and the variety of topics included by the thesis, a deep research related to the theoretical part and the state of the art, both for the DPA attack and the cryptography algorithms, was necessary to get the right coordinates for moving inside the theme.
Subsequently there was the familiarization with new tools, programming languages, SEcube^{TM} board and laboratory instrumentation.
In particular, Lattice Diamond and Active-HDL Lattice Edition were used for writing VHDL code, simulating its behaviour, the used area, the timing and the power consumption. For the creation of the files used for programming the FPGA, Lattice Diamond Run Manager and Lattice Diamond Programmer were exploited. Then, the final file was used by Diamond Deployment Tool for the generation of a vector of bitstreams, exploited for the C code in System Workbench for STM32 environment.
The power traces acquisition was executed automatically by LeCroy waveRunner 6030 digital oscilloscope, exploiting a previously defined trigger signal that indicated the encryption interval. A careful examination of SEcube^{TM} schematics was necessary to individuate the possible power supply accesses and some tips were used to improve the quality of the measurements. The traces were acquired in different conditions, using different power supply accesses, four different versions of the encryption (AES-256 CBC, AES-256 ECB, AES-128 CBC, AES-128 ECB), different instrumentation (in addition to the classic voltage probe, the Tektronix TM502A current probe and an handmade USB-USB/BNC connector), three different working frequencies (66.50 MHz, 38.00 MHz and 2.08 MHz) and finally varying also the number of samples.
Exploiting `Jlsca` DPA Tool, written in Julia programming language (specifically designed for high-performance numerical analysis), different versions of DPA were applied on the collected data, then the results were elaborated. MATLAB was used for the plots and the data comparisons.
Finally, the conclusions concerning the obtained results were elaborated starting from the graph specifically plotted for the different comparisons and the analysed literature.

1.2 Thesis Overview

The thesis was organized in seven chapters:

Chapter One - Introduction:
General overview of topics, tools and instrumentation related to the thesis and of its main goals, followed by a brief summary of all the chapters.

Chapter Two - Differential Power Analysis (DPA) Key Concepts:
Summary of the main topics related to DPA, including a brief overview of the possible attacks at the different levels of abstraction, focusing on the Power Analysis (PA), the differences between Simple Power Analysis (SPA) and Differential Power Analysis (DPA), reporting the advantages and the disadvantages and the necessary conditions for the success of the DPA attack.
It follows a detailed description of all the stages of the attack, that are: the choice of the instrumentation and of the settings (including precautions for improving the quality of the traces), the acquisition of the measurements, the processing of the obtained traces (alignment and decimation), the selection function generation, the averaging and the evaluation.
Since the thesis concerned the Advanced Encryption Standard (AES) executed on a FPGA, the related technical insights were described.
Finally, overviews concerning the statistical methods, the possible variants of DPA and the applicable countermeasures were reported and in conclusion it was added the state of the art, containing the overview and summaries of some of the most interesting articles present in literature concerning the Differential Power Analysis (DPA).

Chapter Three - Cryptography Overview:
A brief description of cryptography main targets was reported, together with an explanation of the used notation. It follows the classification of the kinds of encryption algorithms and the description of their characteristics. The possible modalities of security, the properties of a good encryption and the main types of attack were described.
Afterwards, a section was dedicated to the Private-Key Algorithms, divided in Stream Ciphers (synchronous, self-synchronizing) and Block Ciphers (in particular, DES, 3DES, AES, Blowfish and Twofish) and another section was dedicated to Public-Key Algorithms, with a focus on Diffie-Hellman Key Exchange and RSA.
The main strategies for the management of plaintexts longer than 128 bits were described with a focus on the peculiarities, the advantage and the disadvantages, in particular: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Propagating Cipher Block Chaining (PCBC), Cipher Feedback (CFB), Output Feedback (OFB), Counter (CTR).
In conclusion, as also done for DPA, overview and summaries of some of the most interesting articles concerning encryption algorithms were described.

Chapter Four - AES VHDL Code and Board Programming:
A brief overview of SEcube^{TM} board and in particular the schematics used for writing the `.lpf` file were reported. After, all the passages and the details related to VHDL code for AES-256 (firstly in ECB mode, then also in CBC mode), reporting the block scheme of the FSM, the results of the testbenches, the verifications, the design summary, the timing report, some RTL views and the comparisons between the two proposed architectures. Finally some modifications were added, in order to allow the acquisitions in laboratory and the DPA application, together with the comparisons related to the changes in the architecture.
All the passages related to the exportation of the VHDL code from Lattice Diamond to System Workbench for STM32 environment, using Diamond Deployment Tool and the preparation of the C code were described in detail.

Chapter Five - Measurements in Laboratory:
Some figures acquired from the digital oscilloscope related to the verification of the correct behaviour of the board through some testing signals previously defined were inserted, followed by pictures and plots related to the different tried situations. In particular, the traces were acquired in the following cases: running frequency of 66.50 MHz and handmade USB-USB/BNC connector, running frequency of 38.00 MHz and current probe, running frequency of 2.08 MHz and current probe, handmade USB-USB/BNC connector, J3000 power supply, J3001 power supply, J4001 power supply and J4002 power supply.
Subsequently, some consideration (taking into account also the results form Lattice Diamond Power Calculator Tool) and a comparison with traces from a microcontroller STM32F401 were done, referring also to SEcube^{TM} package and power supplies characteristics.
It follows a description of all the necessary changes in the VHDL code for passing from AES-256 to AES-128 and comparisons with the previous cases. Then, plots and comparisons of the traces with the previous ones.

Chapter Six - DPA Attack and Results:
The steps for the application of different variations of DPA attack using `Jlsca` Tool were precisely described, including the pre-elaboration of the traces, subdivided in sets and averaged, and the generation of the plaintext/ciphertext files. The obtained results were included in some tables and plotted on different graphs in MATLAB in order to underline the principal variations in the analysed cases: in particular, every key hypothesis was compared with the used one, then the difference function and the number of hit bits were calculated. Some bar diagrams were used specifically for making comparisons related to the four different versions of the encryption (AES-256 CBC, AES-256 ECB, AES-128 CBC, AES-128 ECB) with all the different instrumentation used, the variations obtained changing the working frequency and the number of samples considered.

Chapter Seven - Conclusions:
Here all the considerations drawn from the comparisons of the results and the literature analysed in the first chapters were described. To sum up, the main problems identified in the application of the DPA attack were the presence of decoupling capacitors in correspondence to the FPGA power supply accesses, considered like a "natural" countermeasure against this kind of attack, the presence of more then one module concurrently running on the FPGA and the inaccessibility given by the BGA package.
The proposed future works and improvements were the reduction of noise, the increment of the power consumption due to the encryption, the utilization of more advanced tools for the compensation of decoupling capacitors effect and the application of other kinds of attacks like Electromagnetic Analysis (EMA), Temperature Side-Channel Attacks or Differential Fault Analysis (DFA).

CHAPTER 2

Differential Power Analysis (DPA) Key Concepts

2.1 Introduction and motivations

As defined in [12], *"a cryptographic device uses a secret key to process input information and/or produce output information and protocol designs typically assume that input and output messages are available to attackers, but that other information about the keys is not available"*.
Since integrated circuits are composed by individual transistors acting as voltage-controlled switches, when a charge is applied to the gates or remove a charge from the gates of other transistors, interconnect wires and other circuit loads, the current flows across the transistor substrate. Due to this, the motion of electric charges consumes power and produces electromagnetic radiation, that are both externally detectable. Therefore, individual transistors produce externally observable electrical behaviour [3].
In this way, since microprocessor logic units exhibit regular transistor switching patterns, it is possible to easily identify macro-characteristics simply monitoring the power consumption of the device [3].
In the following picture, you can observe a general overview of the main attacks and countermeasures at the different levels of the system:

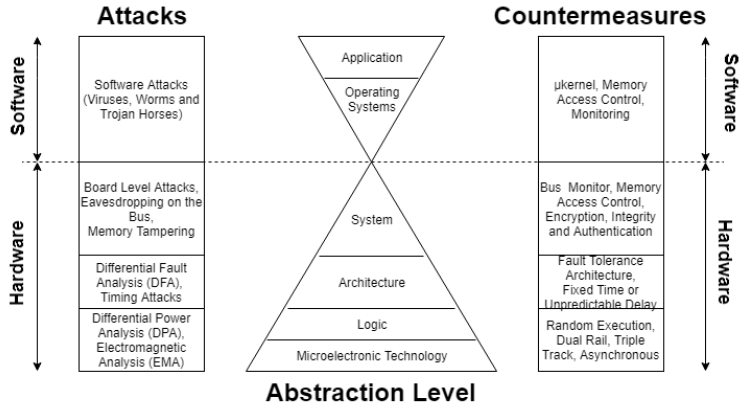

Figure 2.1: Embedded System Security Design Pyramid [1]

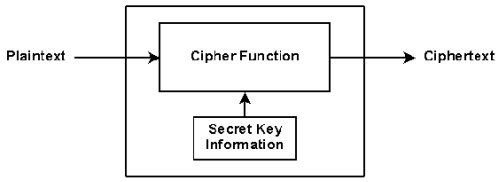

Figure 2.2: Traditional Cryptographic Assumptions [2]

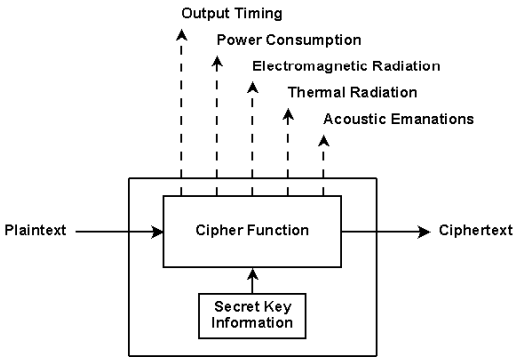

Figure 2.3: Actual information available [2]

As exposed in [13], cryptanalysis attacks can be distinguished in four categories:

- ⋆ **active**: the malicious user acts directly on the device trying to alter its behaviour (e.g., injecting faults)
- ⋆ **passive**: the attacker does nothing other than to observe the normal behaviour of the device
- ⋆ **invasive**: the attack alters physically the device, for example exposing access to its internal components and connections
- ⋆ **non-invasive**: the attack don't require physical alteration of the device because they exploit external information, i.e., physical quantities

Referring to the definitions above, side-channel analysis is a passive and non-invasive type of attack.

Power Analysis (PA) attacks target is to determine the sensitive information of the cryptographic devices from measured power consumption, like: a part of the secret key, the Hamming weight of the secret key or some information identifying the part or the whole secret key [4].

As reference, you can define the *power analysis* as processing and evaluation of information, while the subsequently application of obtained information to abuse the cryptographic device is called *attack* [4].

Power analysis (PA) measures the power consumption of the cryptographic device depending on its activity and was introduced in 1998 by Paul Kocher, Joshua Jaffe and Benjamin Jun and it can be essentially decomposed in the following steps [2]:

1. identification of a relationship between secret key information and the instantaneous power consumption
2. determination of the required inputs to the system, the output values to be measured, and when to capture them
3. extraction of the state by measuring and recording the items identified earlier including the power consumption (the collection of measurements also known as traces can be made in a non-invasive manner while a system performs a cryptographic operation)
4. by processing the extracted information, evaluation of the relationship between the measured items

There are two basic methods of Power Analysis (PA):

- **Simple Power Analysis (SPA)**:
  This method use primarily visual inspection to identify relevant power fluctuations. It has the advantages of the more or less direct determination of the secret key from the measured power traces and of being computationally cheap and simple and feasible in a relatively short time. However, unfortunately it is quite challenging in practice and it requires a detailed knowledge about the implementation of the cryptographic algorithm. Moreover, in most cases, it is necessary to use statistical methods to extract useful signals and it is feasible only in the case that the secret key have a significant impact on the power consumption [2].

- **Differential Power Analysis (DPA)**:
  DPA attacks use statistical analysis and error correction techniques to extract the information related to the key.Indeed, while the effects of a single transistor switching is normally impossible to identify from direct observations of a device's power consumption, you can use statistical operations in order to make the DPA able to reliably identify extraordinarily small differences in power measure [12].
  With respect SPA, the main advantage is that in most cases you don't need a detailed knowledge about the device and encryption algorithm, but just the kind of the algorithm and a simple power simulation model. Indeed, the DPA observes how the power consumption at fixed time moments depends on the processed data and the quality of the traces is relatively not important, since statistical methods are used. Anyway, there are also important disadvantages, that are: you need to acquire a large number of traces for the statistical analysis, you have to consider the extreme importance of the mathematical approach and power consumption models so you have to carefully take care of them in detail and finally the DPA will obviously need more time to be done, always due to the usage of the statistical methods and the necessity of a larger number of traces [2].

Nowadays DPA is the most widely used and best known method for the power analysis, since in general is more powerful and more difficult to prevent [2].
Anyway, you can observe that attacks that involve multiple parts of a security system are difficult to predict and model. If cipher designers, software developers, and hardware engineers do not understand or review each other's work, security assumptions for any level of a system's design will probably be incomplete and/or unrealistic. In order to avoid this problem, security faults often involve unanticipated interactions between components designed by different people [14].
Indeed, performing a Differential Power Analysis (DPA) attack requires knowledge in several fields, in particular: statistics, cryptography and programming (for the attack itself), electronic instrumentation and related tools (for the automatic measurement system) and electronic in general (for the improvement of results) [15].
In general, you have also to consider that, as underlined in [16], a successful DPA attack is subject to two conditions: existence of an intermediate variable in the implementation that is correlated with the power consumption and the exclusively dependence of the intermediate variable on the plaintext (or ciphertext) and small part of the key. If you want to protect devices against DPA, you can get rid of the second condition by data randomization or masking. However, also in this case, higher-order DPA attacks (HO-DPA) are already been introduced and they could also overcome the so-called 1st order masking [16].

2.2 Stages of DPA attack

2.2.1 Device instrumentation

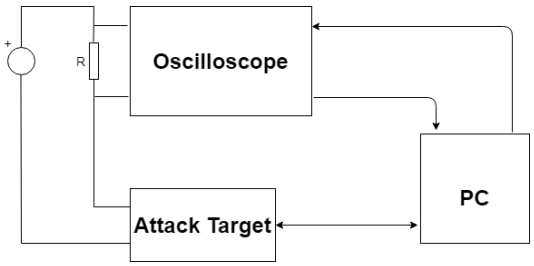

Figure 2.4: Block Diagram of a DPA laboratory setup [3]

This preliminary stage involves the develop of the means to communicate with the device to invoke cryptographic operations and the record of the device responses [6]. The measurement apparatus is basically composed by a digital oscilloscope, the PC and the target device. Depending on the device and the access available, you can use a resistor (otherwise you can generally use the internal resistance) or a current probe connected in series with power supply and ground. In order to get a good quality measurements, you have to take it close to the cryptographic component. In alternative you can use a larger number of lower-quality traces. For triggering, in general you connect the measurement system to the device's I/O lines [6].

2.2.2 Measurement

At this stage, the data are collected: in practice, the power traces are recorded while the target device performs cryptographic operations and as a result, each captured trace is stored on a PC with the associated cryptographic data (plaintext or ciphertext). This is often the most time-consuming step of DPA [6].
In case of necessity, trace quality and capture efficiency may be improved by adding analog filters, adjusting bandwidth or sampling rates and exploring SPA signal characteristics to remove irrelevant regions [6].
On large ASICs/ SoCs with multiple power and ground connections, inputs that power the circuitry implementing cryptographic computations provides better data. Similarly, at the board level, removal of decoupling capacitors or use of an external bench-grade power supply can reduce noise [6].
In some cases, signal quality is affected by a device operating parameters, such as voltage, temperature and clock rate/waveform. In addition, you have also to consider some phenomena, that can be extremely useful for an efficient collection of the data; for example: a) lowering the input voltage may stress voltage regulators and increase leakage; b) a bench-grade clock can reduce timing jitter, especially when synchronized to the sampling clock of the measurement apparatus; c) using a sine-wave clock may reduce high-frequency noise in measurements; d) commands that perform more cryptographic operations are often preferred, since increasing the number of cryptographic operations per trace usually speeds up the data collection stage; e) while many DPA attacks work with random or arbitrary input messages (such as typical ciphertext), chosen input messages can sometimes reveal additional leakage or simplify the analysis; f) depending on the algorithm being analyzed and the attack strategy, it may be most efficient to use a sequence of adaptively-chosen input sets, where each set of inputs depend on the results of a DPA analysis done on the prior set; g) in some instances, when inputs are not fully controllable by an attacker, fault or glitching attacks may be used to influence the cryptographic operations being performed [6].

Finally, data collection is performed with a high-speed analog-to-digital conversion system (digital storage oscilloscopes), that can be selected starting from these conditions: 1) reasonably deep memory (for capturing longer traces); 2) trigger flexibility (to help start trace capturing at the appropriate time); 3) rapid trigger re-arming time and fast transfer rates (to speed up the data collection phase) [6].

Signal fidelity and calibration are less important, since the types of distortions introduced by cheaper laboratory equipment generally do not interfere with the leakage signals exploited by DPA, while it might appear that sampling resolution would matter significantly when dealing with very small correlations, the primary concern for DPA is the signal-to-noise ratio, and sampling errors are usually significantly smaller than other noise sources [6].
Here there is a list of some possible precautions: a) analog pre-processing of the raw signal prior to A/D conversion; b) simple bandwidth limiting (a feature in many oscilloscopes), that can help to remove unwanted high-frequency artifacts; c) minimization of the amount of extraneous data collected (examination of any SPA signals to help narrow down when the cryptographic process occurs); d) implementation of a preliminary DPA test using the known input and output bits as selection functions (it can highlight the precise location where the cryptographic operation is performed); e) DPA tests using selection functions based on the expected intermediate values within the cryptographic operation (more general technique for characterizing the leakage), but this "known-key" analysis is only possible if the attacker can obtain a device with a known key [6]; f) using active differential probe, you can get a good signal to noise ratio, allowing the decrement of the number of power measurements during encryptions; g) in order to get galvanic insulation, you can use a USB-UART cable for serial communication for preventing of grounding loops; h) in the case you implement in FPGAs, high sampling frequency and internal memory of the oscilloscope are strictly necessary [3].
Additionally, you can use also these noise reduction techniques [17]:

- **RC filter:**
  Thank to this, you can get great results if the noise frequency is known, since $f_{cutoff} = \frac{1}{2\pi RC}$ in a cheap way, with a low sensitivity to other components, but the voltage drop over the series resistor can cause stability problems of the digital circuit

- **LC filter:**
  This filter is much better suited for filtering a wider noise spectrum (no voltage drop over its components), with $f_{cutoff} = \frac{1}{2\pi\sqrt{LC}}$, it has an high efficiency and it works bests in case of an external power supply, but the disadvantage is the sensitivity to other components (high inductivity)

In short, in order to optimize the measurement you should execute these preliminary steps [15]:

1. Performing simple operations on the device and play around with the various possibilities for the measurement setup with the aim of optimizing the measurement setup of the board.

2. Reducing as much as possible the noise on the power supply, because it damages the precision of measurements seriously, considering the observations written above.

3. Choosing the right value for resistor (optional) between global supply and the supply pin of the controller across which we measured the current profile, considering that bigger resistance would mean higher voltage swing across the resistor, so the measure will be easier and precise, while smaller resistance avoids reducing the actual supply voltage of the controller, leading to the desired effect of higher voltage swing.
   Indeed, you should avoid reducing the supply voltage, since power consumption itself also depends heavily on the amount of it. Moreover another effects for the reduction of supply voltage

is created by the presence of clamp diodes in input protection circuits of CMOS pads: they turn on when voltage on the input pad is higher than the circuit's supply voltage and, as a consequence, introducing the resistance makes these diodes conductive when the input value is high and voltage drop on the resistance is big, so the internal circuit is supplied by input current from the input pads, which is not measured.

4. Managing the overshoot on the resistance (given by clock transitions), connecting a small fast capacitor in parallel to the resistance. The best way to find out the optimal value of this capacitor is to try different devices and decide after some measurements if the desired effect has happened. It's recommended to use an active differential probe with high input resistance and very low input capacitance.

2.2.3 Signal processing

This stage involves processing traces in software in order to remove alignment errors, isolate features of interest, highlight signals and reduce noise.
In general, you need only a simple temporal alignment or you can omit this step. However, an additional digital signal processing, applied after traces collection in digital form, can significantly improve the efficiency of the rest of the DPA process and improve its outcome [6].
Trace alignment is the simplest signal processing technique and it consist in identifying a reference time location in each trace. To align a trace T_i against a reference trace T_0, a simple correlation test is employed to find the time shift d that minimizes the differences or the square of them between $T_0[j]$ and $T_i[j+d]$ [6].
However, you could need more complex alignment methods in some specific cases like the clock drifts across traces, that are mainly: a) shuffling of the operations order, b) inserting of random no-ops or clock skips and c) using of desynchronized clocks [6].
As an alternative, you could also perform the alignment in the frequency domain, exploiting a Fourier analysis on the relevant regions of each trace.
To sum up, although correct alignment is not strictly necessary for the DPA success, a good alignment could be very useful and could allow the reduction of the number of traces required to extract a key [6].
The two main used approaches are :

- **Static Alignment**: it finds a reference sample in measured power consumption traces and matches the same references by shifting the traces [18].
- **Elastic Alignment**: it uses linear re-sampling of the traces and is more efficient than static alignment, but it requires also more computational power [18].

Additionally another extremely useful technique in the case of management of high number of traces is the **decimation**: indeed, generally using a high decimation factor speeds up the computation algorithm even when using more traces [3].
In real scenario of DPA application, it could happen that the trigger signal cannot be used. In this case, the misalignment of traces measured can be caused by: a) different start time of the measurement (it can be triggered only by sending input data); b) jitter of the oscillator which provides clock for the cryptographic device; c) various execution time of processor instructions which depend on input data; d) intentional DPA countermeasure like inserting random delay while executing cryptographic algorithm [18].

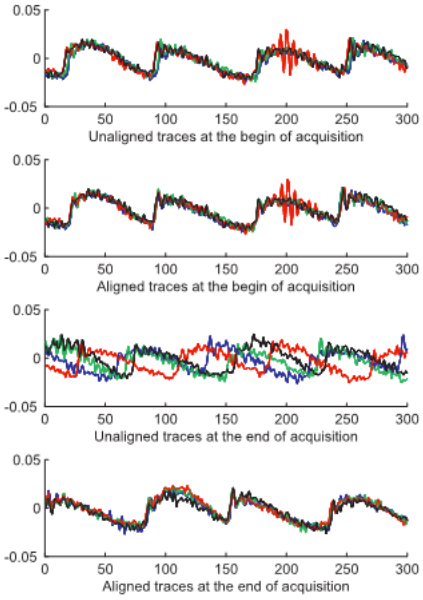

Figure 2.5: Power Traces Alignement [3]

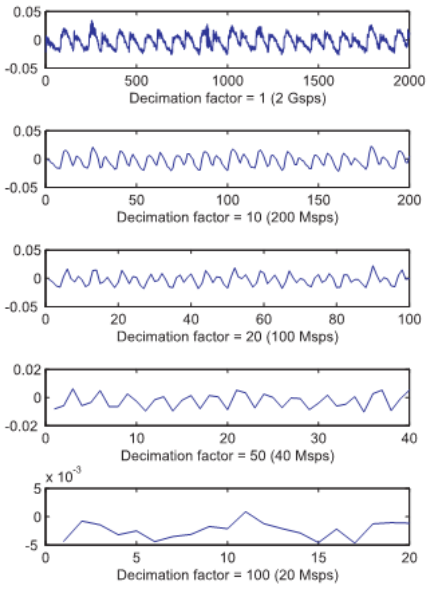

Figure 2.6: Power Traces Decimation [3]

2.2.4 Prediction and selection function generation

As explained in [6], this stage manages different hypotheses about a portion of the key in order to define selection functions for analysis.
At this stage you still can apply another digital filtering for the reduction of noise and focusing on the parts of the spectrum where the leakage signal is present :

- **Trace compression**: it can be performed by adding together successive measurements; in this way you can reduce high-frequency noise and amplify signal resolution while reducing the amount of data that requires processing in subsequent steps, this because if unwanted repetitive effects are present, these can be detected and subtracted from the traces [6].
- **Trace repetition**: it is the collection of traces while an identical operation is repeated; making this operation you can reduce extraneous and measurement noise from traces averaging these together [6].

Furthermore, in [6], they observe also that DPA testing is a highly data intensive task, especially when you have a very large number of traces each containing a large number of measurement points, but much of the information present is not useful. Once the device is characterized, traces can be greatly compressed by discarding all points except the few that matter.
At this stage, in the case of implementation of one variant of DPA, traces have also to be prepared for analysis.

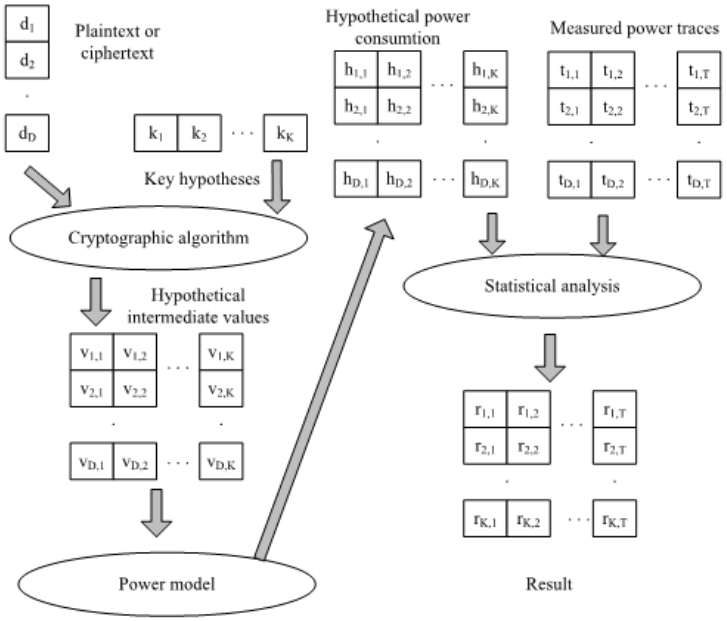

Figure 2.7: DPA General Diagram [4]

DPA attacks exploit leaked information doing a prediction about some aspect of the computation that varies in a key-dependant manner, but real leaks can be complicated [6]. For example, in some devices, transitions from 0 to 1 leak differently than transitions from 1 to 0 or they can have word-oriented leaks which may be correlated to flag bits such as sign, zero, overflow, or carry. Furthermore, Hamming weight and Hamming distance models treat bits independently, but the amplitude of leaks varies significantly for different bits (multi-bit effects) and in the case of FPGAs, it can be used significantly less power than average when a byte being written into the round register has the value 0 and even less power when a byte written into the round register is the same value as the overwritten byte [6].
Starting from the knowledge of these effects, you can formulate different types of selection functions (for PA and PB notation reference, see the figure 2.9):

- **Rising transition**:
  The set PA contains the input transitions of the target gate from 0 to 1 and the set PB contains all the other input transitions. This function models in an accurate way a CMOS circuit (power consumption is measured by means of a small resistor) [5].

- **Hamming distance**:
  The set PA contains the input transitions that make the target gate to commute from 0 to 1 and from 1 to 0, while the set PB contains all the other input transitions. This function models in an accurate way a CMOS circuit (measurement of electromagnetic field) [5].

- **Hamming weight**:
  The set PA contains the input transitions that make the target gate to commute from 0 to 1 and from 1 to 1 and the set PB contains all the other input transitions. This function is used when it's not possible to calculate the transition on the target gate, but only the final value [5].

The secret constant targeted by a DPA attack can be a recognizable portion of a round key or a constant that is a function of multiple secret constants. For this reason, it is necessary an iteration of

the attack for a number of rounds sufficient to recover enough secret materials for all keys.
Here there is a list of some main cases:

- **Known Key Analysis**: usage of specific leakages to guess a large number of bits of the key or sensitive parameter [6].
- **Practical Attack Against the Leak**: if an attacker can select the cipher input, the analysis complexity can often be reduced [6].
- **Chosen Message Analysis**: it can be used for public key algorithms and it consists in guessing only part of the key and predict intermediates using this value, so the best prediction will provide the closest approximate of the key, even if it is still deviate from the actual intermediate state; this approach can be applied iteratively, to successively obtain better approximations until the key is known [6].

Selection functions are normally 0/1 valued, but a selection function can also have a non-binary output (that will be used as a weight in the final averaging step, where the weights can be zero, positive or negative), especially when leakage characteristics of a device are well known. In general, **non-binary selection functions** and **multi-bit selection functions** can improve the efficiency of attacks [6].
As soon as you have defined your selection function, you can compute its outputs for each trace.

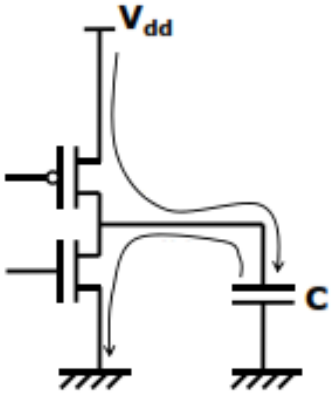

Figure 2.8: Power Consumption in a CMOS Circuit [5]

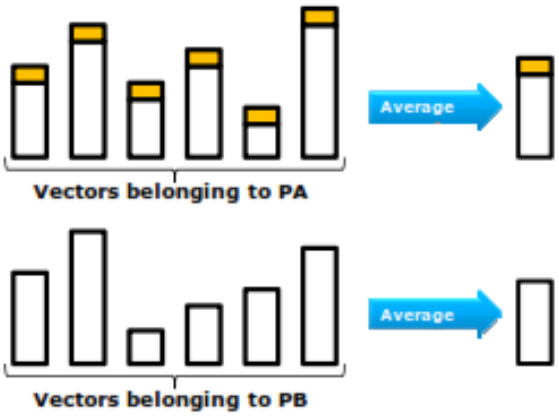

Figure 2.9: Power Consumption After Pattern Partitioning [5]

2.2.5 Averaging

Now you can compute the averages of the input trace subsets defined by the selection function outputs. Normally this stage is the computational bottleneck; indeed, you generally compute: 1) the averages of various subsets of the traces; 2) the average of all traces; 3) the variance at each point across all traces [6].
The performances for the averaging step are determined by *processing power* and *storage throughput*. In terms of number of traces (N), length of traces (L), and number of selection functions (M), the naive complexity of this task is $O(N \cdot M \cdot L)$ and the memory use is $O(M \cdot L)$ [6].
The principal optimizations that can be used to simplify the problem are synthesized below:

1. If you assume to have to compare the average of the subset where a selection function is 1 (A_1) to the one of the subset where the selection function is 0 (A_0) and that the average of all traces (A_{all}) is known, then A_0 can be calculated from A_1 and A_{all} all using the number of selection

functions and the total number of traces.
In particular, this approach is very convenient in the case you have to calculate many selection functions over the same set of traces, because you have only to calculate once for all A_{all} and the $A_{1,m}$ subsets and you get $A_0 \approx A_{all} - A_1$.
This results in just a factor of two improvements, so the complexity is still $O(N \cdot M \cdot L)$, but you get the important advantage of halving CPU and memory use [6].

2. If you have a cache with size of $2^8 - 1$ that works with 8 traces at a time, computing the sums of 255 possible combinations of these traces, then these traces can be added into the m-th averaging task using one addition out of the cache instead of up to 8 additions of the individual traces.
A cache of size c requires $O(c \cdot L)$ memory and takes $O(2^c \cdot L)$ operations to set up, so the performance improvement is in the order of $O\left(\frac{N}{c} \cdot M \cdot L + 2^c\right)$.
In general, if there is a statistical bottleneck in the selection function outputs and only B unique sequences of selection functions are generated over our L traces and you assume $B << L$, then using an "input bins" cache with B entries you can improve performance to $O\left(L + \frac{B}{c} \cdot M \cdot N + 2^c\right)$ [6].

3. Since bottlenecks can also be introduced by repeating each message multiple times, any decrease in the trace size produces a proportional improvement in averaging time, so you can use compression methods in order to greatly reduce the number of points [6].

Although DPA tests normally compare the difference of averages, this test is not effective in all cases, especially in the presence of countermeasures, so it could be necessary a preprocessing step before averaging.
The task is also parallelizable, since data can be distributed over many drives to eliminate I/O bottlenecks and computation can be distributed over multiple threads or machines [6].

2.2.6 Evaluation

In this step you have to analyse DPA test results to determine the most likely candidate key guesses [6].
This process could be done visually, but in general it is better using automated tools for logistic reasons.
Regions with high noise can show spurious spikes in a differential trace, so you have to divide each point in the differential trace by the standard deviation of all traces at that point, in order to correct these spurious spikes effects and to assess the statistical significance of the results [6].
Doing this, the obtained result will be a normalized trace giving the polarity and significance of the difference (measured in standard deviations) at each point in time [6].
In reality, the classic "difference of averages" is just one way that two distributions of measurements can differ, while a more general statistical test can compare at the distribution of measurements at each point in the subsets of traces, and calculates the significance of differences observed between them. For such analysis, the "averaging" stage actually computes the distributions of measurements rather than compressing the distributions down into their averages [6].
For certain types of DPA attacks, the evaluation process is more complex, since there is the presence of *harmonics of the correct key*, that are defined as multiple guesses besides the correct key which have significant correlation to the target leak and thus may show spikes in the differential trace [6].
Indeed, the assumption that when the guessed byte is incorrect the output from the prediction is uncorrelated is an approximation: smaller positive or negative correlations exist, but they are ignored in a basic DPA analysis, although they can provide additional information for identifying the correct key guess [6].

Harmonics are a function of the target algorithm and the selection function strategy employed, so it is possible to: 1) analyze the selection function process; 2) determine the pattern of harmonics it would generate (the main problem is that the pattern of harmonics varies); 3) match this pattern against the amplitudes of the observed harmonics [6].
For each bit, the fundamental signal corresponds to the correct key guess and you can use alternate multi-bit evaluation strategies, that are sensitive to harmonic peaks [6].

In the case of iterative application (for example AES-256 and triple-DES, where multiple round subkeys or multiple encryption keys must be found), new information about the key enables the generation of new selection functions, so obviously the DPA evaluation stage is not the final one. The iteration process usually restarts back at the selection function generation stage, but for adaptive chosen message attacks, the evaluation result guides the next sets of inputs for the data collection stage [6].

2.3 Using DPA to attack AES

As observed by the author of [6], "computational intermediates, such as the outputs of the AES S-box, have a small statistical influence on power consumption measurements, but these small correlations can be used to reveal the secret key".

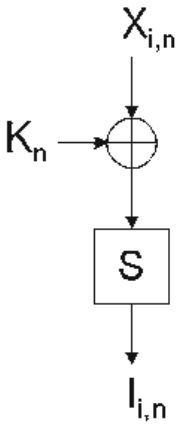

Figure 2.10: AES S-box lookup during first round [6]

To have an overview of what will be explained with additional details in "Chapter Three - Cryptography Overview", here are reported the steps of the first round of AES encryption [6]:

1. **Initialization:** The initial 16-byte state of the cipher, organized as a 4x4 byte matrix, is initialized to the 16 bytes of the plaintext.
2. **AddRoundKey:** The 16-byte secret key is exclusiveORed with the 16 bytes of the plaintext state.
3. **SubBytes:** Each byte of the state is replaced by another using the S-box, which is an invertible lookup table.
4. **ShiftRows:** Bytes in each row of the state are shuffled.
5. **MixColumns:** Each column of bytes of the state is mixed using a linear operation.

The DPA attack will target the output of **AddRoundKey** and **SubBytes** in AES.
Here is reported the notation for AES:

- I_i denote the 16-byte intermediate state of the cipher just after the SubBytes step in round 1
- $I_{i,n}$ denote the n-th byte of this state ($n \in [0, ..., 15]$)
- K is the first round key
- K_n is n-th byte of the first round key
- $X_{i,n}$ denote the n-th byte of plaintext X_i used for the i-th trace
- S is the AES substitution table defined in the AES standard

$I_{i,n}$ only depends on one byte $X_{i,n}$ of the input and one byte K_n of the key:

$$I_{i,n} = S[X_{i,n} \oplus K_n] \tag{2.1}$$

In this equation: $X_{i,n}$ is a *known variable* (one byte of plaintext), K_n is a *secret constant* and $I_{i,n}$ is an unknown variable which depends on a 1-byte secret constant and other known quantities.
AES can be broken easily if there is an efficient test that reveals whether a given candidate for K_n is correct [6].
In particular, K_n is an 8-bit value, so at most 256 queries of this test would be required to confirm the correct K_n.
The 16 K_n bytes that make up the entire AES-128 key could be found by simply solving for each byte separately.
DPA provides a practical way to test if a candidate value of K_n is correct: the candidate K_n is substituted in the equation to derive the value of $I_{i,n}$ for each $X_{i,n}$
A selection function can be based on the calculated $I_{i,n}$; for example you could use bit 0 (the LSB) of $I_{i,n}$ as selection function output. Each trace is assigned to one of two subsets, depending on whether the selection function result is 0 or 1 for the candidate K_n and the plaintext being encrypted when the trace was captured [6].
The difference of the subsets averages is then examined: if the candidate K_n is correct, the value of the S-box output bit predicted by the selection function has even a tiny correlation to the power traces and the DPA test will show spikes; if the candidate K_n wrong: the predicted values of $I_{i,n}$ will be (largely) unrelated to any data being processed by the target device, and the DPA test will not be (or will be much less) statistically significant [6].
When you repeat the analysis for all the 16 bytes of the state to recover the entire 128-bit AES secret key, you can reuse the same traces; moreover it is not necessary to collect separate data, since each test is checking for different correlations in the data set.
Notation for DPA:

- T is the set of traces that are collected
- T_i denote the i-th trace
- $T_i[j]$ denote power measurement or sample at the j-th time offset within the trace T_i
- C is the set of known inputs or outputs for the traces
- C_i corresponding to the i-th trace
- $D(C_i, K_n)$ denote a binary valued selection function with input C_i and guess K_n of a part of a key

Each point j in the differential trace Δ_D for the guess K_n is computed as follows:

$$\Delta_D[j] = \frac{\sum_{i=1}^{m} D(C_i, K_n)T_i[j]}{\sum_{i=1}^{m} D(C_i, K_n)} - \frac{\sum_{i=1}^{m}(1 - D(C_i, K_n))T_i[j]}{\sum_{i=1}^{m}(1 - D(C_i, K_n))} \tag{2.2}$$

For a typical DPA analysis, the guess for K_n that produces the largest spikes in the differential trace is considered to be the most likely candidate for the correct value.
The attack can be adjusted easily for other cipher modes and target devices [6].

2.4 Observations

The final three steps (prediction, averaging, and evaluation) are often iterated.
For example, with AES-256, the first round key is typically found before the attack can begin on the

second round key.
In other cases, additional steps may also be repeated, e.g., if adaptively chosen input messages are being used. Indeed, attackers and product evaluators are motivated to obtain keys as quickly and easily as possible and if basic DPA is not immediately successful, there are many ways to adapt the attack to compensate for countermeasures or to reduce data collection or processing time.
Improvements in the initial data collection can reduce the time to recover keys by improving signal quality [6].

2.5 DPA application on FPGAs

FPGAs are powerful tools to design products that require hardware performance without having the costs and delays of ASICs; furthermore, FPGAs are also more flexible and they can be updated in the field [19].
Reconfigurable computing systems, such as FPGAs, are a very promising platform for designing high-performance cryptographic because they are characterized by high throughput rates and inherent design flexibility. As a consequence of the growing popularity of FPGAs as a cryptographic processing element, research is focusing into their susceptibility to power analysis attacks [20].
As illustrated in [21], FPGA security is complicated by the environment in which the FPGA is expected to perform, since the design of FPGA security features assumes no physical barrier and no communication network. This environmental assumption is one of the main differences between FPGA security and internet security, where servers may physically reside in a trusted environment and can verify identity through name servers.
To sum up, since there is physical access to the device, it can mounted any electrical, physical or side channel attack. As a consequence, the target is the negation of this to the adversary: in other words, the containing system have to ensure the security of the FPGA by controlling all the accesses [21].
Although military systems may employ physical security, the cost is impractical in commercial systems, where it is balanced against the value of the information being protected; in other words, FPGA security is designed to make the cost of breaking the security greater than the adversary's expected economic gain: this decision is ultimately in the hands of the application developer, not the FPGA manufacturer [21].
Today we find FPGAs deployed in a security hostile environment, protecting data of great commercial value, so the value of the IP of the application designs has grown, motivating significant investment in built-in security functions and the value of the data handled by the FPGA has also increased significantly, including such information as personal-data databases. [21]
During power-up, an SRAM-based FPGA reads its configuration from an external non-volatile memory: this includes all functional design as well as the I/O configuration for the pins and the exact placement and routing of all used components, so copying a configuration to use it for multiple FPGAs makes all devices behave in exactly the same way. The whole design of an FPGA application is encoded within the configuration file, that as a role similar to the one of the software for microcontrollers. This nature implies that, as advantage, you are provided of a means to update the configuration file of an FPGA to adapt its behaviour to new requirements or to fix early design flaws, but, as disadvantage, copying of a design and thus stealing of IP (Intellectual Property) is simplified [19].

Here there is a simple synthesis of the principal threats for FPGAs, that are described in [21]:

Cloning:
An adversary copies the FPGA programming, then uses it in an identical device, selling it as his own. It may apply to an entire design or may apply to a subset of the design and the adversary does not require detailed knowledge of the design [21].

Overbuilding:
The adversary, such as a contract manufacturer, builds additional systems, inserting the legitimate bitstream into those systems and selling them without the designer's approval and it doesn't require detailed knowledge of the design [21].

Reverse Engineering:
Someone may reverse engineer the bitstream to recover the circuit design that it implements. The main targets are: understanding and duplicating the functionality of the application, using it as part of an attack on other aspects of the system and tampering with the application to insert malware. However, there are relevant application problems, like the not-standardized bitstream, so that every new FPGA device requires a new bitstream reverse engineering effort. A more insidious problem is dealing with the size of the application. To sum up, although reverse engineering may divulge the netlist of the application, transforming a multimillion gate netlist into an understandable design that can be modified is problematic. The complexity of the application increases its value, making theft attractive, but the consequent size makes theft difficult [21].

Tampering:
There's a modify of an application, that can be employed to: a) add logic that leaks information from an application (in this case, tampering must control the application to set values in the bitstream, so reverse engineering may also be required); b) disable parts of the application, potentially defeating other security measures (in this case, merely scrambling parts of the bitstream may be sufficient) [21].

Spoofing:
The adversary replaces the FPGA bitstream with his own: this may or may not include components derived from cloning or reverse engineering. The disadvantage is that a spoofed application may compromise the system in which it operates [21].

Denial of Service, Destruction of the FPGA and Substitution:
Since it is assumed that the FPGA is in the hands of an adversary, denial of service and malicious destruction of the FPGA device are somewhat irrelevant: rather than mount a clever attack on the design to prevent the system from operating, an adversary could simply smash the FPGA with a hammer. If a system requires an FPGA containing a unique key, an adversary may choose to circumvent security measures by replacing the FPGA in a system with another identically manufactured device from the FPGA vendor without the key or with his own key: in many cases, this substitution is simpler than attempting to break the FPGA device security. Since these physical attacks are so simple, FPGAs typically do not defend against these types of threats [21].

Referring to [19], in order to counter reverse-engineering threats, it is possible to implement an encryption mechanism called *bitstream encryption*: instead of saving a plain bitstream file within the configuration ROM feeding an FPGA, the designer stores an encrypted configuration. The used key is chosen by the designing engineer and is programmed into the FPGA and the part of the FGPA memory storing this secret key is battery powered so that the key will immediately be erased on power

loss of the battery support: this feature is designed to hinder invasive attacks to recover or reverse engineer a device configuration. With the known encryption key inside the FPGA and the encrypted bitstream stored within a ROM, the FPGAs can be securely configured, because only encrypted data pass the channel between ROM and FPGA. The FPGA has to contain a dedicated hardware to decrypt the bitstream, that isn't accessible for other purposes [19] .

2.6 Power Simulation Models

As exposed in [4], an important process is the mapping of hypothetical intrinsic values to hypothetical values of power consumption during the prediction step of DPA attack.
For this purpose, easy power simulations models are used, considering that the attacker does not have any detailed knowledge about the device under attack. A schematic description of the main ones is following [4]:

- **Hamming Weight Model (HW):**
  It's the basic power simulation model and it is usually used when the attacker does not have any information about the netlist of the device and about the processing data. The attacker expects that the power consumption is directly proportional to the number of non-zero bits in the processed data. An hardware model is not suitable for simulation of the CMOS circuits but the experimental results from practice show that Hamming weight of currently processed data is dependent on power consumption of CMOS circuits and it can be used.

- **Hamming Distance Model (HD):**
  The attacker expects that the power consumption is directly proportional to the the number of changed data values in the processed data and he can map the data which are transmitted via data bus to the value of power consumption without the knowledge of the device netlist. The power consumption, which is caused by a change of the data bus value from v_0 to v_1 is proportional to $HD(v_0, v_1) = HW(v_0 \oplus v_1)$.

2.7 Statistical Methods

Here is described a list of the main statistical methods illustrated in [4]:

- **Correlation Coefficient:**
  It's one of the best known methods to determine the linear relationship between two random variables.
  The correlation coefficient is defined by the covariance as follows:

$$\rho(X, Y) = \frac{Cov(X, Y)}{\sqrt{\sigma^2(X) \cdot \sigma^2(Y)}} \tag{2.3}$$

  It's a dimensionless quantity and it can only take values in the range: $-1 \leq \rho \leq +1$.

  - if $\rho = -1$, there is a indirect dependence (change in one group is accompanied by an opposite change in the second group)
  - if $\rho = 0$, there isn't a detectable statistic dependence between values of the two groups
  - if $\rho = +1$, there is a direct dependence or a perfect correlation between the values of two groups

Also ρ is typically unknown and must be estimated with an estimator r defined by the following equation:

$$r = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \overline{x})^2 \cdot \sum_{i=1}^{n}(y_i - \overline{y})^2}} \tag{2.4}$$

In DPA attacks, the correlation coefficient is used to determine the linear dependence between the columns h_i and t_j were $i = 1, ..., K$ and $j = 1, ..., T$ (fig.2.7). The result is a matrix R containing correlation coefficients and each value $r_{i,j}$ based on elements of D from columns h_i and t_j, so we can write:

$$r = \frac{\sum_{d=1}^{D}(h_{d,i} - \overline{h_i})(t_{d,j} - \overline{t_j})}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \overline{h_i})^2 \cdot \sum_{d=1}^{D}(t_{d,j} - \overline{t_j})^2}} \tag{2.5}$$

- **Difference of Means:**
  The basis is a comparison of two measured groups by calculating the difference of the mean values of these groups.
  The attacker creates a binary matrix H which divides the measured power traces into two groups. Sequence of zeros and ones in each column H is a function of the input data d and estimates key value k_i. In order to determine if estimate key k_i is correct, attacker can divide the matrix T into two sets of lines (two sets of power consumption by h_i).
  The first data file contains the lines T, where index corresponds to the position of zeros in the vector h_i.
  The second data file contains the remaining rows of T.
  Subsequently, the attacker calculates the mean of the rows. Vector m'_{0i} denotes the average of the rows in the first file and m'_{1i} denotes the mean of the second file.
  Estimation key k_i is correct, if there is a marked difference between the m'_{0i} and m'_{1i}. This difference indicates the relationship between h_{ck} and some of the columns T.
  As in the previous case, this difference indicates the point at time, where the intrinsic values corresponding to h_{ck} are processed.
  In other moments, the differences between mean vectors are zero. The result of the attack is the matrix R, where each row corresponds to the difference between vectors m'_{0i} and m'_{1i} one estimate key value. Equations to calculate R according to the difference of means method are given:

$$m'_{1i,j} = \frac{1}{n_{1i}} \cdot \sum_{l=1}^{n} h_{l,i} \cdot t_{l,j} \tag{2.6}$$

$$m'_{0i,j} = \frac{1}{n_{0i}} \cdot \sum_{l=1}^{n} (1 - h_{l,i}) \cdot t_{l,j} \tag{2.7}$$

$$n_{1,i} = \sum_{l=1}^{n} h_{l,i} \tag{2.8}$$

$$n_{0,i} = \sum_{l=1}^{n} (1 - h_{l,i}) \tag{2.9}$$

$$R = M_1 - M_0 \tag{2.10}$$

  where n is the number of rows of the matrix H; in other words, this parameter represents the number of measured power consumption [4].

- **Distance of Means:**
  This method is based on the distance of mean values and improves previous method considering

the standard deviation.
It uses a commonly known hypothesis test to compare the equality of the mean values of two different distribution.
The attack divides the matrix T into two sets of rows for each key hypothesis as well as explained before.
Compared with the previous method, the difference is that the mean values are compared by test of distance mean.
Elements of the matrix R are calculated according to the following equation:

$$r_{i,j} = \frac{m_{1i,j} - m_{0i,j}}{s_{i,j}} \tag{2.11}$$

where $s_{i,j}$ is the standard deviation of the distribution of the two groups [4].

2.8 Variants of DPA

In addition to SPA and DPA, there are variants of the basic attack that are better suited to exploit information leakage in some settings [6]:

Correlation Power Analysis (CPA):
It involves evaluating the degree of correlation between variations within the set of measurements and a model of device leakage that depends on the value of one or more intermediates in the cryptographic calculation. If the number of traces available is limited, CPA can help make the maximum use of the data. It is most effective in white-box analysis where the device leakage model is known and it can also be used for black-box evaluations if there is some correlation between the actual leakages of the device and the leakage model used for CPA [6].
The possible results are: 1) if CPA does not find leakage, it is ambiguous whether this indicates low leakage in the device or a large gap between actual leakage and the model; 2) if CPA is successful at recovering the key, additional leakage modes may also be present [6].
The common examples are always: Hamming weight (power consumption directly proportional to the number of non-zero bits in the processed data) of a multi-bit value in a register or on a bus and Hamming distance (power consumption directly proportional to the the number of changed data values in the processed data) between a value and the value it overwrites [6].
Generally, Hamming weight or Hamming distance models can be reasonable approximation for leaks from software running on some 8-bit microcontrollers, but for larger ASICs and CPUs, these methods are less effective at modelling the leakage found so countermeasures developed solely on the basis of these simplistic models are generally insufficient to prevent DPA attacks [6].

Probability Distribution Analysis or Zero-Offset Second-Order DPA Attack (ZO2-DPA):
Given any differing probability distribution, it is possible to define a trace preprocessing operation that will make DPA work; alternatively, the DPA analysis can be performed by comparing probability distributions instead of averages [6] .

High-Order DPA:
It combines multiple samples from a trace and it targets a known or hypothesized relationship between parameters contributing to a side channel; the "order" of a high-order attack is the number of parameters involved in the target relationship [6].
It can help to analyse relationships such as: a) *similarity/difference*, since the calculations at different points in a power trace (or in a pair of traces) may involve a common data parameter; b) *masked shares of a secret (traditional second-order DPA)*, since, in a masked implementation,

a sensitive intermediate may be manipulated as two parts, each part by itself is random, but the exclusive-or of the parts would give the intermediate (individual measurements are correlated to the parts, but uncorrelated to the variable); c) *unknown input*, since, measurements at a first location in a set of power traces can be used to estimate the value of otherwise unknown inputs and outputs to cryptographic functions, then these estimates can be used to prepare probabilistic predictions for plaintext/ciphertext portions needed for a standard DPA attack which is then evaluated at a second location in the set of power traces [6] .

Template Attacks:
Following the explanation in [6], they seek to maximize the use of a small number of traces from a target device: the analyst constructs a model of the target device (in contrast to CPA, template attacks build a model from actual power measurements or simulations). This model is typically represented by a set of statistical parameters for the expected power traces corresponding to various states that the device may enter. Once these templates have been created, they can be used to determine the most likely state of a target device from a small number of actual traces. In order to construct the model (set of template), you can use: a) the actual device that is the target of the attack; b) a device from the same family as the target; c) simulations of the target device. Although template attacks may require a large amount of initial effort, they can achieve theoretically an optimal use of the signal in the target traces [6].

Reverse Engineering Unknown S-Boxes and Algorithms:
In cases where the custom cipher is based on a well-known design, but with different set of constants such as S-boxes, DPA can be used to first reverse engineer the values of these customized constants and then attack the cipher [6].
A general approach is to iteratively evaluate a broad range of selection functions, yielding correlations that are stronger or later in time may correspond to the device's intermediates deeper in the algorithm. The better selection functions are kept and their outputs are used as candidate inputs for subsequent iterations of the analysis [6].

Machine Learning:
In [22], the authors demonstrate that, given a proper tuning, even in an unrestricted scenario, using a sufficient number of profiling traces, ML techniques are able to be more efficient than template attacks. A measure called the *Data Confusion Factor* provides a measure of successfulness for machine learning techniques and can offer additional level of confidence in the performance of ML techniques. Moreover, this measure fills the gap since it clearly quantifies the difficulty of the problem with regards to the level of noise and the number of classes. It has two contributing parts: the first part penalizes the data sets that have classes with a larger number of instances than the average class in the set as these are more difficult for the classifiers to handle, while the second part deals with instances that are wrongly classified [22].

Other Side-Channels Attacks:
SPA and DPA techniques applied to EM measurements are termed Simple Electromagnetic Attacks (SEMA) and Differential Electromagnetic Attacks (DEMA); such attacks can be highly effective, particularly if power measurements are unavailable. Data from other sources could potentially be used in DPA-style analysis; the areas for further research are: photon emissions from semiconductors and temperature measurements [4].
To extract keys from obfuscated cipher implementations, digital data dumped from repeated cryptographic operations can be used in lieu of power traces. Cache timing and other micro-architectural attacks also use similar statistical techniques, but make somewhat different assumptions about the type of access available to an attacker in order to break software implementations of cryptography [4].

2.9 Preventing DPA

Techniques for preventing DPA and related attacks fall roughly into three categories, as you can read in [14]:

1. **Reduction of Signal Sizes**:
   You can use strategies, such as: using constant execution path code, choosing operations that leak less information in their power consumption, balancing Hamming Weights, balancing state transitions and physically shielding the device.
   Unfortunately such signal size reduction generally cannot reduce the signal size to zero, as an attacker with an infinite number of samples will still be able to perform DPA on the (heavily-degraded) signal. In practice, aggressive shielding can make attacks infeasible but adds significantly to a device cost and size [14].

2. **Introduction of Noise into Power Consumption Measurements**:
   Like signal size reductions, adding noise increases the number of samples required for an attack, possibly to an infeasibly-large number. In addition, execution timing and order can be randomized. Designers and reviewers must approach temporal obfuscation with great caution, however, as many techniques can be used to bypass or compensate for these effects and for safety, it should be possible to disable temporal obfuscation methods during review and certification testing [14].

3. **Design of Cryptosystems with Realistic Assumptions about the Underlying Hardware**:
   Non-linear key update procedures can be employed to ensure that power traces cannot be correlated between transactions; similarly, aggressive use of exponent and modulus modification processes in public key schemes can be used to prevent attackers from accumulating data across large numbers of operations.
   Using a leak-tolerant design methodology, a cryptosystem designer must define leakage rates and functions that can survive from cryptography. Leakage functions can be analysed as oracles providing information about computational processes and data, where the leakage rate is the upper bound on the amount of information provided by the leakage function. Implementers can then use leak reduction and leak masking techniques as needed to meet the specified parameters. Finally, reviewers must verify that the design assumptions are appropriate and correspond to the physical characteristics of the completed device [14].

To sum up, as exposed in [23], the principals countermeasures will be: 1) masking, that uses a random value called "mask" which is mixed (XOR, additions, multiplications, etc.) with the sensitive data; 2) shuffling, that plays on randomizing the order of operations of the cipher; 3) hiding, that makes the leakage uniform and independent of the data processed. At the same time, the countermeasures drawbacks could be in terms of time, cost or both.

2.10 State of the Art

In "Analysis and Improvements of the DPA Contest v4 Implementation" [23], the authors implemented on an AES-256 protected with a low-entropy masking scheme (RMS), mostly written in C language, running on an ATMEL AVR-163 microcontroller. They proposed several attacks and an improved implementation, demonstrating that it is possible to resist the non-profiled attacks at an overhead of 27% in code size, 50% in memory and 1.5% in computation time.

In the paper "On the Portability of Side-Channel Attacks - An Analysis of the Xilinx Virtex 4, Virtex 5, and Spartan 6 Bitstream Encryption Mechanism" [19], they made, as described in the title,

a side-channel analysis of the bitstream encryption mechanism provided by Xilinx FPGAs - Virtex 4, Virtex 5, and Spartan 6 families. The results they got provide an overview of a practical real world analysis and can help in the implementation of side-channel countermeasures. They used sophisticated attacks on off-the-shelf FPGAs that go far beyond schoolbook attacks on 8-bit AES S-boxes and they were able to extract the key with the measurements of a single power-up. They used a bandpass filter on measured traces and improved the analysis method by removing all phase shifts from the measured traces using an additional FFT preprocessing step. They observed also that the attack on Virtex 5 and later Spartan 6 FPGAs required more power traces to be successful, which is mostly due to a worse signal-to-noise ratio due to a newer process technology.

Manfred Aigner and Elisabeth Oswald in their "Power Analysis Tutorial" [15] illustrated the attack on a 8052 compatible ATMEL 89S8252 microprocessor implementation (application of DPA on asymmetric cryptosystems), using a C++ model that can be useful also to verify some countermeasures against DPA attacks. They declared that following the hints they illustrated you can make a DPA attack on a software DES implementation with less than 200 measurements.

In "Revisiting Higher-Order DPA Attacks: Multivariate Mutual Information Analysis" [16], they made a Multivariate Mutual Information Analysis (MMIA) on 1st and 2nd order masked software implementations of a DES like mini-cipher, that is a novel and different approach to HO attacks, that allows to directly evaluate joint statistics without pre-processing more efficient and less affected by noise. However, MMIA's measurement cost grows sub-exponentially with the attack order, so the security provided by the masking countermeasure needs to be reconsidered as 3rd and higher order attacks.

Machine learning techniques compared with Template Attacks applied on AES with low cost masking protection (Weka suite v3.8) are illustrated in "Side-Channel Analysis and Machine Learning: A Practical Perspective" [22]. They introduced also a new measure called the Data Confusion Factor, that can be used to assess how well machine learning techniques will perform on a certain dataset. The profiled side-channel attacks illustrated are: 1) Template attacks (TAs), the most of the time based on Gaussian noise assumption (most used and powerful from an information theoretic point of view); 2) stochastic attack (SA), with a linear regression in the profiling phase.
They used practical scenarios with three levels of noise and two multi-class scenarios and they demonstrated that even in unrestricted (but still practical) scenarios machine learning techniques can be more successful than template attacks. The implementation has been done with datasets that represent the de-facto standard in the side-channel community and 20000 measurements have been taken. They considered the following cases: a) uniformly distributed classes; b) binomial distributed Hamming Weight classes.

C.L. Pitu and R.Câmpeanu in their paper [17] used a Power Analysis Toolkit (dedicated software framework) with noise reduction techniques and digital pre-filtering mechanisms on Arduino Uno board equipped with an ATmega328 microcontroller using AES library (Atmel). They presented acquisition method has the target of decreasing the effort of the post-processing algorithms in order to speed up the attack and reduce the needed computing resources. Moreover the presented technique doesn't require dedicated board or special components and all the used tools are open source or freeware. All the presented software mechanisms were implemented in Python that is easy to use, with medium performance and portability.

The paper "An Integrated Validation Environment for Differential Power Analysis" [5] is related to a new flexible integrated simulation-based environment that allows validating a digital circuit. It allows to set most of the parameters and features (e.g., power consumption models, observation mech-

anisms, and statistical analysis) in such a way that different forms of DPA can be experimented. In all the cases the overall execution time is affordable and compatible with an iterative design flow, in spite of the large number of samples and of the target precision. The solution implements most of the parameters and features of DPA and can be used in a general way, for any design and countermeasure; in particular, it deals with: a) real measures or simulated data; b) three different power consumption models; c) any number of target bits; d) different combinations of the results of each target bit; e) reports on packets statics; f) graphical output.
Since the simulation is performed at transistor level, the power consumption waveforms are very accurate and in agreement with the physics.

"FPGA Security: Motivations, Features and Applications" [21] describes FPGA security primitives from multiple FPGA vendors and gives examples of those primitives used in some applications.

Matteo Bollo and Paolo Maistri in "Composite Fields against Side Channel Analysis for the Advanced Encryption Standard" [24] present AES and RAS design fully implemented on composite fields robustness against Differential Power Analysis (Xilinx Spartan 3 XC3S1000). In particular, Advanced Encryption Standard targeted at resource constrained applications, countermeasure developed against side channel analysis and its robustness also from a testability point of view, i.e., how it affects the amount of detectable/undetectable fault of the design, and how it could be used itself to perform a Built-In Self-Test (BIST). Then also the robustness of the countermeasure against side channel analysis has been verified through simulation. Both architectures (the reference and the composite field implementations) have been synthesized in AMS c35 Corelib technology, and several simulations at the transistor level have been run in order to collect the power consumption traces. Due to excessive time required for such simulations and to the fact the simulations are not bothered by noise existing in real life, the attack was limited to a specific byte, thus only 256 traces per design. Moreover, they observed that area overhead wasn't negligible.

In "Differential Power Analysis Attack on ARM based AES Implementation without Explicit Synchronization" [18], the authors implemented a Differential Power Analysis (DPA) attack on microcontroller STM32F103 (STM32 family) with 32-bit ARM Cortex-M3 core (AES with 128-bit key). In this case, elastic alignment methods are used for the trace synchronization and Dynamic Time Warping (DTW) and its fast algorithm Fast-DTW are deployed in order to match similar patterns in reference and misaligned power consumption traces.

DPA applied to Xilinx Virtex-II Pro FPGA running a Data Encryption Standard (DES) core is described by Song Sun, Zijun Yan and Joseph Zambreno in "Experiments in Attacking FPGA-Based Embedded Systems using Differential Power Analysis" [25]; in particular, they used an automated data acquisition and analysis design for an FPGA-based implementation of the Data Encryption Standard (DES).

In "Side-channel analysis of SEcube^{TM} platform" [13], the authors implemented a DPA attack of AES-128 on SEcube^{TM} and on ST Microelectronics Nucleo board equipped with the same microprocessor. The encryption process consisted into cypher the block of data with the specified key and repeat it for 256 times, varying the plaintext altering the last byte for every possible configuration, from 0x00 to 0xFF for every cycle (this in order to control and force the changes into the plaintext for generating on purpose data-correlation among the sets of measurements). The measurements acquired with the oscilloscope were amount to 2.713 traces for Nucleo Board and to 15.872 for SEcube^{TM}. The encryption sessions were 16 for the Nucleo board and 62 for SEcube^{TM} and each power consumption trace contained 10.000 samples. In order to get a relaxing time for data acquisition, the attack was

limited to a specific byte, thus only 256 traces in every encryption session. They changed the following parameters: 1) Number of samples of each trace of 25%, 30%, 50% and 100% with respect to the full number of samples; 2) Selection Function based on Hamming weight (hweight), Hamming distance (hdistance) and Rising .
They observed that attacking only the Sbox of the AES, the results are slightly better, but, in most cases, just half of last byte of the key is correct and considering the whole AES, the number of wrong bits increases. Both the used platforms provide almost the same level of security. Authors assumed that the obtained results might lead to improvements when DPA is combined with a brute-force attack.

In "Introduction to Differential Power Analysis" [6], Paul Kocher, Joshua Jaffe, Benjamin Jun and Pankaj Rohatgi implemented an AES-128 encryption using the AES-128 smart card traces (SASEBO platform).

The authors of "General Scheme of Differential Power Analysis" [4] used a DPA applied with PICDEM 2+, programming device MPLABIC2D and cryptographic module PIC 16F84A: the AES algorithm was implemented into the cryptographic module and the attack was focused only on the performed non-linear transformation *SubBytes* and concretely on the first byte of the secret key.

Michal Varchola and Milos Drutarovsky described in [3] a DPA in order to obtain a 128-bit key which was stored in the internal Flash memory of a ADuC 834 microcontroller based on a standard 8051 core. One important result was that, using an higher decimation factor, the computation algorithm is speeded up even using more traces.

"A Practical Differential Power Analysis Attack against an FPGA Implementation of AES Cryptosystem" [26] paper contains the description of a DPA attack on SRAM-based Xilinx Spartan-II, with the plaintexts generated from a 128-bit LFSR on the FPGA. The experiment executed confirmed that power analysis has to be considered as a serious threat for FPGA security, since they were able to discover the key in almost two hours.

Lukáš Mazur and Martin Novotný illustrated in their paper "Differential Power Analysis on FPGA board: Boundaries of Success" [27] a DPA attack on FPGA board (Spartan 3E Starter Board) with the implementation of AES cipher. They focused on the last round and used Hamming distance as a power model, as attacking first round and/or using Hamming weight does not lead to success in case of FPGA implementation. They demonstrated that the success rate of the DPA attack against FPGA board running AES encryption strongly depends on the measurement setup: a) while using differential probe disables the attack, using differential amplifier enables it; b) removing decoupling capacitors plays major role in success rate of the DPA attack: surprisingly, their presence does not disable the attack, however, as expected, their removal enables the attack significantly; c) replacing standard switched-mode power supply with accumulators and linear stabilizers simplifies the attack, but its effect is not that significant: even the circuit powered from (noisy) switched-mode power supply can be successfully attacked, however, noiseless power supply (accumulator) simplifies the attack in terms of number of traces necessary for successful attack on the FPGA implementation.

In "Differential Power Analysis Countermeasures for the Configuration of SRAM FPGAs" [28], William Luis, G. Richard Newell and Kenneth Alexander proposed a SmartFusion2 flash based FPGA with a secure boot loader, giving data on the effectiveness of the underlying solution using a statistical characterization of side channel leakage using Test Vector Leakage Assessment (TVLA) methodology proposed by Cryptography Research. TVLA provides an objective result as to the vulnerability of unprotected cryptographic implementations contrasted to the effectiveness of the solution presented,

since a SmartFusion2 FPGA, layered with a solution integrating DPA-mitigated cryptographic IP, yields a strong and effective solution for addressing DPA vulnerable endpoints.

Jun Wu, Yiyu Shi and Minsu Choi described in their paper "FPGA-based Measurement and Evaluation of Power Analysis Attack Resistant Asynchronous S-Box" [29] a low-power side channel attack (SCA) on a FPGA board (SASEBO-GII), demonstrating that their asynchronous S-Box is resistant to DPA attacks and has a lower power consumption than the synchronous one.

The authors of "A DPA Resistant FPGA Implementation of AES Cryptosystem with Very Low Hardware Overhead" [30] presented a novel approach for the implementation of AES with a greater strength against DPA with minimal additional area overhead (algorithmic countermeasure based on mathematical properties of Rijndael algorithm), verifying it on a Xilinx Spartan-II FPGA.

In "DPA Resistant AES on FPGA using Partial DDL" [31], they illustrated an improvement in terms of area for the Dynamic Differential Logic (DDL), a countermeasure against DPA used for FPGAs, called Partial DDL (since DDL is applied only to a part of the cryptographic hardware implementation). The demonstration has been done in a lightweight architecture of AES in the Partial Separated Dynamic Differential Logic (Partial SDDL) for FPGAs.

"Correlation Power Analysis Attack of AES on FPGA Using Customized Communication Protocol" [32] contains the description of a correlation power analysis attack carried out on AES encryption algorithm implemented on a Xilinx FPGA on SASEBO (Side-Channel Attack Standard Evaluation Board) using a customized communication interface protocol and optimizing the number of traces.

An interesting point of view is exposed in "Experiments in Attacking FPGA-Based Embedded Systems using Differential Power Analysis" [25], where the authors detailed their experience in performing a DPA attack on a commercial FPGA development board (Virtex-II Pro), presenting an automated data acquisition and analysis design for an FPGA-based implementation of DES. The presented platform gives a systematic view on how to successfully perform the DPA attack in a practical sense. Getting to the point, the efficiency of analysis is critical to DPA if the attacker wants to break the FPGA cryptographic system, since: 1) most modern cryptographic algorithms are based on the fact that they can be broken in theory, but not in practice (ex: billions of years to break a 256-bit AES system in a brute-force search); 2) the key of cryptosystem like AES or DES is usually changed after a variable period of time; 3) DPA attacks are not as powerful as expected when facing the commercial FPGA platform due to the decoupling capacitors, that results being a natural countermeasure; 4) presence of more than one electrical device on an FPGA board, since the other on-board components may overwhelm the power consumed by the reconfigurable logic.

CHAPTER 3

Cryptography Overview

3.1 Introduction to Cryptography

As written in [33], *"If I take a letter, lock it in a safe, hid the safe somewhere in Karachi, then tell you to read the letter, that's not security. That is obscurity. On the other hand if I take a letter and lock it in a safe, and then give you the safe along with the design specifications of the safe and a hundred identical safes with their combinations so that you and the world's best safecrackers can study the locking mechanism - and you still cannot open the safe and read the letter - that's security."*
For many years, govermment and military had the exclusive for the cryptography, while nowadays is diffused in all the environments.
In addition to **Security** (defined above), cryptography targets are also:

Confidentiality: protection of the data from all the receivers

Authentication: possibility for the receiver of a message to ascertain its origin

Integrity: possibility for the receiver of a message to verify that it has not been modified

Non-Repudiation: avoiding that a sender falsifies deny later that he sent a message

As reported in [34], cryptography provide solutions to two major problems of data security: 1) **privacy problem**, preventing an opponent from extracting information from a communication channel; 2) **authentication problem**, preventing an opponent from injecting false data into the channel or altering messages so that their meaning is changed.
Here are reported some notations:

- *cryptographic algorithm/cipher*: mathematical function used for encryption and decryption
- *restricted algorithm*: algorithm based on keeping the way that algorithm works a secret
- *cryptography*: art and science of keeping messages secure
- *cryptanalysis*: art and science of breaking ciphertext without the knowledge of key
- *cryptanalyst*: practitioner of cryptanalysis
- *cryptology*: branch of mathematics encompassing both cryptography and cryptanalysis

· *plaintext*: information intended to be sent

· *ciphertext*: result of encrypting the plaintext using a predetermined key, that is sent over an insecure channel

· *cryptosystem*: five-tuple (P, C, K, E, D), defined as follows:

P: finite set of possible plaintexts

C: finite set of possible ciphertexts

K (*keyspace/entropy*): finite set of possible keys (with a n-bits key, the size is 2^n)

E: encryption rule for the key (e_K)

D: decryption rule for the key (d_K)

Notice that each $e_K : P \to C$ and $d_K : C \to P$ are functions such that $d_k(e_k(x)) = x$ for every plaintext x defined in P [33]

From the paper "Introduction to Cryptanalysis: Attacking Stream Ciphers" [35], the *strength of a cryptographic algorithm* (or *computational complexity of the cipher*) is expressed in the total amount of computations an adversary needs to perform to recover the secret key and for a perfectly secure cipher, the computational complexity is the same as the key space. However, with clever optimizations it is often possible to find the secret key by doing far fewer computations than the actual entropy would require and these are defined as the *actual attack complexity of a cipher*.

As defined in "Introduction to Cryptography" [33] by M. I. Aziz and S. Akbar, there are two types of key-based algorithms:

⋆ **Symmetric/Private-Key/Conventional Algorithms**:
In this case, encryption key can be calculated from the decryption key and vice versa and in most cases they are equal.
Encryption and decryption can be denoted as: $e_k(x) = y$ and $d_k(y) = x$ They can be divided into categories:

- **Stream Ciphers**:
  They manage a single bit at a time of the plaintext.
  As summarized in [36], the advantages are the speed of transformation, since algorithms are linear in time and constant in space and the low error propagation, since an error in encrypting one symbol will not affect subsequent symbols. In the meanwhile, the disadvantages are that all information of a plaintext symbol is contained in a single ciphertext symbol and the susceptibility to insertions/modifications, because an active interceptor who breaks the algorithm might insert spurious text that looks authentic.
- **Block Ciphers**:
  They operate on the plaintext in groups of bits, called blocks.
  As summarized in [36], the great advantages are the high diffusion, since information from one plaintext symbol is diffused into several ciphertext symbols and the immunity to tampering, since it is difficult to insert symbols without detection. The prices to pay are the slowness of encryption, because an entire block must be accumulated before encryption/decryption can begin and the error propagation, because an error in one symbol may corrupt the entire block.

⋆ **Asymmetric/Public-Key Algorithms**:
The key used for encryption is different from the one used for decryption and they cannot be calculate one from the other in a reasonable time. The encryption key can be made public [33].

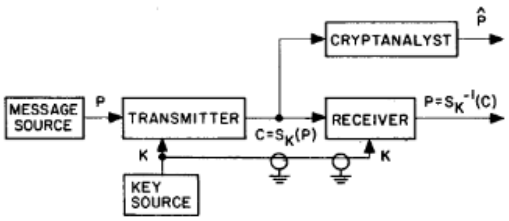

Figure 3.1: Private Key Cryptographic System [7]

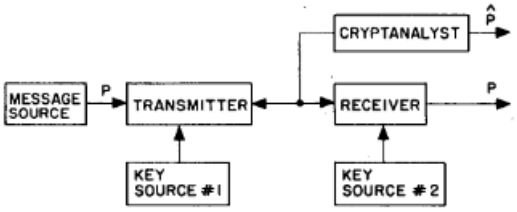

Figure 3.2: Public Key Cryptographic System [7]

One of the best-known algorithms today is the Advanced Encryption Standard (AES). It has been declared in 2011 the new federal government standard by the National Institute of Standards and Technology (NIST) and it is a symmetric key algorithm. However, AES is currently only known to be truly vulnerable to side-channel attacks, which are directed at weaknesses in the implementation of an algorithm rather than the algorithm itself [37].

There are two fundamentally different ways in which cryptographic systems may be secure (defined in the paper "Privacy and Authentication: an Introduction to Cryptography" [34]):

- **Unconditionally**:
  When the amount of information available to the cryptanalyst is actually insufficient to determine the enciphering and deciphering transformations, independently from the provided computing power.
- **Computationally**:
  When the intercepted material contains sufficient information to allow an unique solution to the cryptanalytic problem, but there is no guarantee that this solution can be found by a cryptanalyst with limited computational resources.

While you can demonstrate the security of an unconditionally secure systems, you can't can do the same with the computational infeasibility of any cryptanalytic problem. However the adopted strategy is the one of computationally security and the only unconditionally secure system in common use is the **One-Time Tape (OTT)**, in which the plaintext is combined with a totally random key of the same length: usually the plaintext is represented as an n-bit binary string which is xored with a totally random key of the same length, that, as suggested by the name, will never be reused. Even if the cryptanalyst could try deciphering under all possible keys, he would merely see all 2^n possible plaintexts, including not only the correct one, but also all other meaningful plaintexts of the same length: since intercepting the cryptogram does not allow the cryptanalyst to rule out any plaintext messages, he learns absolutely nothing except the length of the message [34].

As exposed in [33], a good cipher should has the following properties:

- **Semantic Security**:
  This means that: 1) there is no possibility of recovering encryption key using encryption scheme; 2) you can't recover the plaintext/ciphertext from the ciphertext/plaintext (x from $e_k(x)$); 3) the encryption of a string looks like a random garbage; 4) encryption schemes not only hide plaintext but also don't leak any partial information about it.
- **Indistinguishably**:
  The encryption of x is indistinguishable from encryption of an equal length string of nonsense.

Here a list of the main types of attacks (based on [33]) is reported:

Ciphertext-Only Attack: The cryptanalyst has the ciphertext of several messages, all of which have been encrypted using the same encryption algorithm and he has to cryptanalyst's job is to recover the plaintext of as many messages as possible, or better yet to deduce the key (or keys) used to encrypt the message.

Known-Plaintext Attack: The cryptanalyst has access not only to ciphertext of several messages, but also to the plaintext of those messages and he can deduce the key (or keys) used to encrypt the message or an algorithm to decrypt any new messages encrypted with the same key (or keys).

Chosen-Plaintext Attack: The cryptanalyst not only has access to the ciphertext and associated plaintext for several messages, but also chooses the plaintext that gets encrypted. This is more powerful than known-plaintext attack, because the cryptanalyst can chose specific plaintext blocks to encrypt end he can deduce the key (or keys) used to encrypt the message or an algorithm to decrypt any new messages encrypted with the same key (or keys).

Adaptive-Chosen-Plaintext Attack: This is a special case of chosen-plaintext attack. Not only can the cryptanalyst choose the plaintext that is encrypted, but he can also modify his choice based on the results of previous encryption.

Chosen-Ciphertext Attack: The cryptanalyst can choose different ciphertexts to be decrypted and has access to the decrypted plaintext. It can be primarily applicable to public-key algorithm.

Chosen-Key Attack: The cryptanalyst has some knowledge about the relationship between different keys.

Rubber-Hose Cryptanalysis: The cryptanalyst threatens, blackmails, or tortures someone until they give him the key.

To sum up: "Security of the algorithms depends on how hard they are to break. If the cost required to break an algorithm is greater than the value of the encrypted data, then you are probably safe. If the time required to break an algorithm is longer than the time the encrypted data must remain secret, then you are probably safe. If the amount of data encrypted with a single key is less than the amount of data necessary to break the algorithm, then you are probably safe" [33].

You can also define a classification for the breaking of the algorithm:

Total Break: a cryptanalyst finds the key, K, such that $d_K(y) = x$.

Global Deduction: a cryptanalyst finds an alternate algorithm, A, equivalent to $d_K(y)$, without knowing K.

Instance/Local Deduction: a cryptanalyst finds the plaintext of an intercepted ciphertext.

Information Deduction: a cryptanalyst gains some information about the key or plaintext.

In conclusion, it can be deduced the cryptography is a complex science and in order to get good results is fundamental having a productive cooperation between expertise from different environments, as summarized in the following figure 3.3:

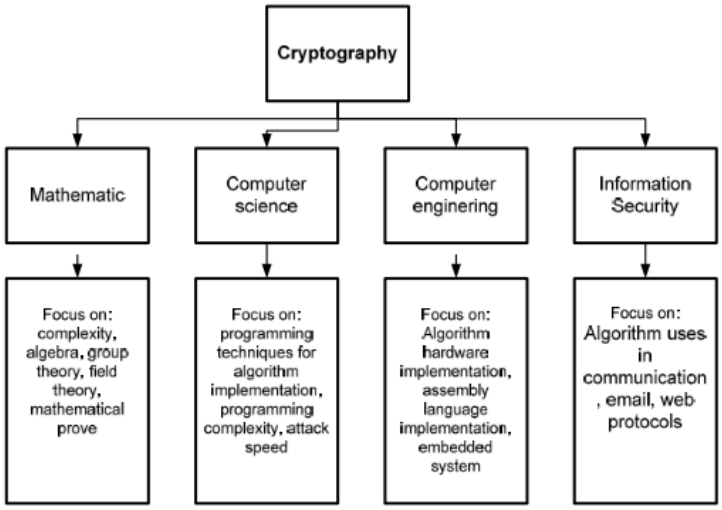

Figure 3.3: Scheme of Skills for Cryptography [8]

3.2 Private-Key Algorithms

3.2.1 Stream Ciphers

They produce a large chunk of secret, random looking data and combines it with the plaintext to produce ciphertext: without the exact same data chunk, the plaintext cannot be uncovered from the ciphertext. The *keystream* is defined as the random data representing a stream of bits which is derived from the secret key. In order to have a propagation to a successor state after each encryption step, a stream cipher contains some persistent memory (*internal cipher state*), which is initialized by the secret key [35].
The target for the output has to be comparable to the one of a Pseudo Random Number Generator (PRNG): the keystream varies, depending on the initialized secret key and the moment of encryption with respect to the propagation of the internal state and should consist of a unique bit string that contains uniformly distributed random bits. In general, encryption of plaintext and decryption of ciphertext are both performed by the exclusive-or (XOR) operation, that has the useful mathematical property of the possibility of inversion (this is why it can be used both for encryption and decryption) [35].
There are two types of stream ciphers:

- **Synchronous**:
  The encryption bits are computed independently from the plaintext. This could be very useful in situations when a communication channel is more prone to error (the speed of data-traffic is more important than the completeness and integrity of the data).

- **Self-synchronizing**:
  They compute the successor of its internal state with a function over the previous state and the ciphertext, so the internal state diverts from its original propagation path when a transmission error occurs.

The most widely used and best studied are the synchronous stream ciphers.
An important objective of a stream cipher is to avoid a direct relation between the input (secret key) and output (keystream) of the cipher: because the entropy of a synchronous stream cipher is limited to the size of the internal state, the produced keystream will eventually repeat itself, even if it is very insecure [35].
Keystream can be seen as an unique set of bits which must be as long as the plaintext, but continuing distribution of fresh keystream for long data sequences is undesirable and, with an increase in electronic transmissions, the need for alternative solutions grew and several stream cipher encryption techniques were introduced. For instance, they introduced *physical rotor machines* which operated mostly mechanically (like the well-known Enigma) and later the *non-linear binary sequence stream cipher* (after the computation of a new keystream bit, the successor function updates the internal state by a linear function to preserve as much entropy to the cipher and then the output component applies a non-linear filter function to compute the next keystream bit).
Moreover, in most cryptosystems it is important to link multiple encrypted messages in one cryptographic session (*chaining of encryption*), but stream ciphers inherently provide this feature since their ciphertext is produced incrementally [35].

3.2.1.0.1 RC4

Rivest Cipher 4, is the most widely used of all stream ciphers, particularly in software. It's also known as ARCFOUR or ARC4. RC4 has been used in various protocols like WEP and WPA (both security protocols for wireless networks) as well as in TLS. Recent studies have revealed vulnerabilities in RC4, prompting Mozilla and Microsoft to recommend that it be disabled where possible. In fact, RFC 7465 prohibits the use of RC4 in all versions of TLS. These recent findings will surely allow other stream ciphers (e.g. SALSA, SOSEMANUK, PANAMA, and many others, which already exist but never gained the same popularity as RC4) to emerge and possibly take its place.

3.2.2 Block Ciphers

Despite their advantages in flexibility and speed, stream ciphers are currently scarcely used, since typical stream cipher attacks aim to separate the plaintext from the encryption bits. For example, an attack can exploit a general and unavoidable weakness in traditional stream ciphers where the keystream is generated independently from the plaintext: small alterations (bitflips) to the ciphertext might be sufficient to perform the attack without actually recovering the secret key.
The security that is provided by the underlying building blocks of stream ciphers are well-studied, since the security implications of these separate components may not hold when they are combined and used together in one cryptographic algorithm.
The usual sizes of each block are 64 bits, 128 bits, and 256 bits [35].

3.2.2.0.1 DES

DES is a non-linear method of encrypting 8-byte blocks of data which was adopted by the National Institute of Standards and Technology as the algorithm to be used for protection of computer data. It has a complexity of 2^{56} for searching of the key.
DES is an iterated cryptosystem, in particular the same procedure is repeated 16 times (*rounds*) in a row in order to get greater confusion [9].
The DES key is 56 bits plus 8 checkbits (completely dependent on the 56 bits). Each round of the function, uses a 48 bit subkey, derived from the key, but different for each round. The input to each round in DES is an 8 byte quantity, initially the plaintext block to be encrypted [9].
As you can see in fig. 3.4, in any given round r, first the right 32 bits of the block are encrypted. The first 32 bits of output of round r of DES (left half), is just the right half of the input to round r. The

right half of the output is the result of the encryption process mentioned above, xored with the left half of the input to round r [9].

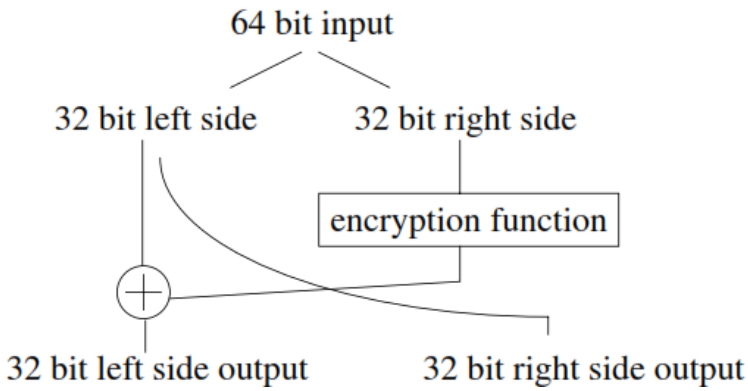

Figure 3.4: Scheme of a single round in DES [9]

Concerning the encryption function (fig.3.5), the right half of the data is first expanded from 32 bits to 48 bits using the "expansion permutation" that duplicates 16 particular bits of the 32 bits and then permutes the resulting 48 bit data using a fixed permutation algorithm. Then the permuted 48 bits are xored with the 48 bit subkey for the current round and divided into 8 groups of 6 consecutive bits each. Each 6 bit quantity is sent through an "S-box" (a sort of look-up table that turns a 6 bit quantity into a 4 bit quantity). There are 8 different look-up tables that correspond to the 8 different S-boxes in the encryption function. Finally the concatenated 32 bit output from the S-boxes is again permuted [9].

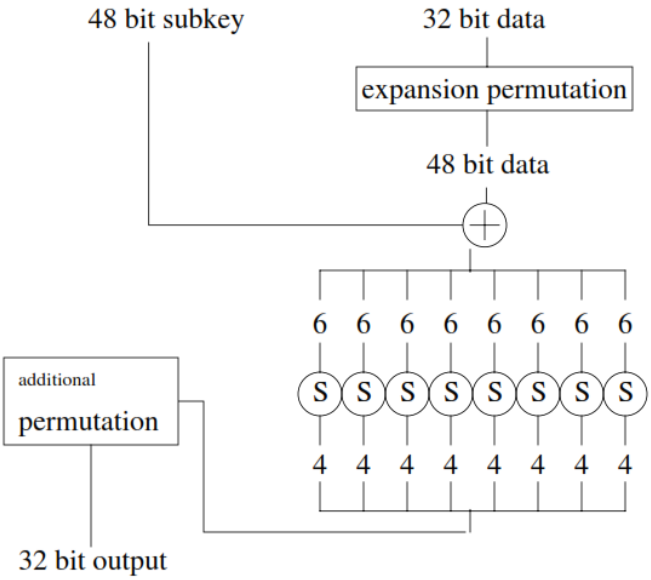

Figure 3.5: Scheme of Encryption Function in DES [9]

3.2.2.0.2 3DES

In practice, it is DES applied three times, applying different keys.
As a consequence, it is stronger than DES, but also much slower, so it never became the successor of DES.

3.2.2.0.3 AES

In 1998 the National Institute of Standards and Technology (NIST) announced a competition for a new block cipher to replace Data Encryption Standard (DES), due to the problem of the short key length. The AES candidates were evaluated according to three main criteria: security, cost (computational efficiency) and algorithm and implementation characteristics (flexibility and simplicity) [10]. In 2001 NIST announced their choice: the Rijndael algorithm. In 2002 AES became effective as a federal government standard and in 2003 U.S. government announced that AES could be used to protect classified information, and it soon became the default encryption algorithm for protecting classified information as well as the first publicly accessible and open cipher approved by the National Security Agency (NSA) for top-secret information [10].

AES is implemented as an iterated block cipher which has a block length of 128 bits and its design principle is the *Substitution Permutation Network (SPN)* which is a combination of both substitution and permutation (also known as confusion and diffusion).
There are three permissible key lengths:

- 128 bits, using 10 rounds (AES128)
- 192 bits, using 12 rounds (AES192)
- 256 bits, using 14 rounds (AES256)

Defining χ as the plaintext and Num as the number of rounds and γ as the ciphertext (State), the high-level description is the following:

1. Initialization of the State and AddRoundKey operation.
2. For Num-1 rounds, execution of the following operations:
   - SubBytes (substitution)
   - ShiftRows (permutation)
   - MixColumns
   - AddRoundKey
3. Execution of:
   - SubBytes
   - ShiftRows
   - AddRoundKey

The plaintext consists of 16 bytes and can be denoted $\chi = \chi_0, ..., \chi_{15}$ and represented in a 4x4 matrix:

$$\begin{pmatrix} \chi_0 & \chi_4 & \chi_8 & \chi_{12} \\ \chi_1 & \chi_5 & \chi_9 & \chi_{13} \\ \chi_2 & \chi_6 & \chi_{10} & \chi_{14} \\ \chi_3 & \chi_7 & \chi_{11} & \chi_{15} \end{pmatrix} \rightarrow \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix}$$

[10]

3.2.2.0.3.1 AddRoundKey

This operation simply consist in combining each byte of the state with a block of the round key using bitwise XOR (bitwise addition modulo 2) [10].

3.2.2.0.3.2 SubBytes

Each byte $a_{i,j}$ in the State Matrix is replaced by $SubByte(a_{i,j})$, using an 8-bit substitution box called the *Rijndael S-box*.
This operation provides the non-linearity in the cipher. Since if a block cipher is linear with respect to some field, then, given a few known plaintext-ciphertext pairs, it is possible to recover the key using a simple Gaussian elimination, the nonlinearity of S-boxes is very important.
In contrast to DES S-boxes, the AES Sbox can be defined algebraically with operations in a finite field GF (2^8), which is also known to have good non-linearity properties. S-box is constructed by combining the inverse function with an invertible affine transformation, to avoid attacks based on simple algebraic properties [10].
Indeed, Rijndael S-box takes 8 bits as an input, converts it to an element of a particular finite field $F_{2^8} = Z_2[x]/(x^8 + x^4 + x^3 + x + 1)$. Multiplication in a finite field is done modulo the irreducible polynomial used to define the finite field, so two elements of this field are multiplied and divided by $(x^8 + x^4 + x^3 + x + 1)$.
If the input byte is $a_{i,j} = a_7a_6a_5a_4a_3a_2a_1a_0$, it can be described as polynomial: $a_7x^7 + a_6x^6 + a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0$.
Then you have to find the multiplicative inverse element (in this finite field all non-zero elements have multiplicative inverse elements) and then again convert polynomial back to binary.
Finally you have to apply a specific affine transformation [10].
This passages are summarized by the following algorithm:

```
SubBytes(a){
    z ← binaryToField(a)
    if z ≠ 0 then z ← findInverse(z)
    a ← fieldToBinary(z)
    c ← 01100011
    for i = 0 to 7{
        b_i ← a_i ⊕ a_{i+4} ⊕ a_{i+5} ⊕ a_{i+6} ⊕ a_{i+7} ⊕ c_i
    }
    return b
}
```

Figure 3.6: Scheme of SubBytes Algorithm [10]

3.2.2.0.3.3 ShiftRows

This step is basically a transposition of the elements of the State Matrix. It cyclically shifts the bytes in each row by a certain offset:

1. first row: unchanged
2. second row: each byte is shifted one to the left
3. third row: each byte is shifted two to the left
4. fourth row: each byte is shifted three to the left

In this way, each column in output will be composed by the bytes from each column of the input, since input block is written column-wise [10].

$$\begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{pmatrix} \rightarrow \begin{pmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,1} & a_{1,2} & a_{1,3} & a_{1,0} \\ a_{2,2} & a_{2,3} & a_{2,0} & a_{2,1} \\ a_{3,3} & a_{3,0} & a_{3,1} & a_{3,2} \end{pmatrix}$$

3.2.2.0.3.4 MixColumns

The four bytes of each column of the state are combined using an invertible linear transformation: it takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. To sum up, ShiftRows and MixColumns provide diffusion in the cipher [10].
MixColumns step can be viewed as a multiplication by the particular *MDS Matrix (Maximum Distance Separable)* in the finite field defined. In MDS Matrix:

- 01 is 00000001 in binary (1 as a polynomial)
- 02 is 00000010 in binary (x as a polynomial)
- 03 is 00000011 in binary ($x + 1$ as a polynomial)

This step can be seen also as a multiplication of $a(x) = a_3x^3 + a_2x^2 + a_1x + a_0$ by the fixed polynomial $c(x) = 03x^3 + 01x^2 + 01x + 01$, taking the result modulo $x^4 + 1$ (with a_i denoting the elements of a column) [10].

3.2.2.0.4 Blowfish and Twofish

Blowfish has a block size of 64 bits and supports a variable-length key that can range from 32 to 448 bits.
One thing that makes blowfish so appealing is that it is unpatented and royalty-free.
Twofish is a 128-bit block cipher that supports key sizes up to 256 bits long.

3.3 Public-Key Algorithms

In this case, every client and server has a unique public key: anyone can encrypt a message using this key, but only the owner of the key can decrypt the message again. So there is one special server (SS) (that knows the public key of everyone) and every other client and server knows SS public key and, when it creates or changes its public key, asks the SS for a certificate, that is a plaintext certificate "decrypted" with the SS secret key (so anyone can encrypt the certificate using the SS public key to read the plaintext version, and know that only the SS could have generated that message) [9].
If two machines want to communicate without knowing their public keys, they have to send each other a plaintext with their certificates, that they can encrypt using SS public key. At this point, they knows their public keys and that they are genuine, so they can carry on a private conversation.
From this mechanism, you can deduce that the critical point is the protection of SS public key [9].

Indeed, the difficulty of distributing keys has been one of the major limitations on the use of conventional cryptographic technology: in order for the sender and receiver to make use of a physically secure channel for key distribution, they must be prepared to wait while the keys are sent, or have made prior preparation for cryptographic communication, so the cost of distributing many keys could be prohibitive [34].

As explained by P. K. Mohapatra in [38], theoretically a public-key system can be constructed by a special case of the one-way function ($f(x)$ is easy to compute for any input x, while $f^{-1}(y)$ is hard to compute) known as a *trapdoor one-way function*: in this last case, $f(x) = y$ (y known) becomes easy to solve if additional information (the *trapdoor*) is available.

3.3.1 Diffie-Hellman Key Exchange

The protocol is based on the difficulty of determining discrete log in group Z_p^*.
Assuming that p and g are publicly known, it follows this algorithm:

1. Alice chooses a at random, $0 \leq a < p-1$
2. Bob chooses b at random, $0 \leq b < p-1$
3. Alice computes $u = g^a \ mod \ p$ and sends to Bob.
4. Bob computes $v = g^b \ mod \ p$ and sends to Alice.
5. Alice computes $K = (g^b)^a \ mod \ p$ upon receipt of message from Bob.
6. Bob computes $K = (g^a)^b \ mod \ p$ upon receipt of message from Alice.
7. Alice and Bob compute the same value $g^{ab} \ mod \ p$ and they can use it as the secret key for the next communication.

An eavesdropper, Eve, tapping onto the transaction gets the values of $g^a \ mod \ p$ and $g^b \ mod \ p$, but not $g^{ab} \ mod \ p$: if Eve could determine a from $g^a \ mod \ p$ or b from $g^b \ mod \ p$, then she could compute $K = g^{ab} \ mod \ p$, but both these are instances of the *discrete log problem* [38].

3.3.2 RSA

The system is based on the hardness of the integer factorization problem in group Z_n. First of all, you have to set the RSA in order to generate public/private key pair:

1. Bob generates two large primes, p and q.
2. Bob computes $n = pq$ (that will be used as the modulus) and $phi(n) = (p-1)(q-1)$
3. Bob chooses a random e $(0 < e < phi(n))$ (used as the public exponent) such that e is relatively prime to $phi(n)$.
4. Bob computes d (private exponent) as the inverse of $e/mod/phi(n)$.
5. Bob publishes (n, e) as his public key and keeps (n, d) as his private key. The primes p and q must be kept secret or destroyed.

e should be relatively prime to $phi(n)$, otherwise, there will be no modular inverse and you cannot find any private exponent d.

At this point, assuming that Alice knows Bob's public key (n, e), but doesn't know Bob's private key (n, d), you can apply RSA algorithm to encrypt and decrypt data:

1. Alice encrypts message m by computing $c = m^e/mod/n$
2. Alice sends c to Bob.
3. Upon receipt, Bob decrypts c by computing $c^d/mod/n$ and gets back m

You can note that the encryption and decryption are inverse operations and the security of this scheme is based on the hope that the encryption function $c = m^e$ is one-way, since it'd be necessary for an opponent Eve to know the value of d in order to compute $m = c^d/mod/n$ (equivalent to factoring n to get p and q) [38].

3.4 Block Cipher Modes of Operation

In the case the plaintext you have to encrypt is longer than 128 bits, you ca adopt different strategies in order to manage it using more times the default size.

3.5 Electronic Codebook (ECB)

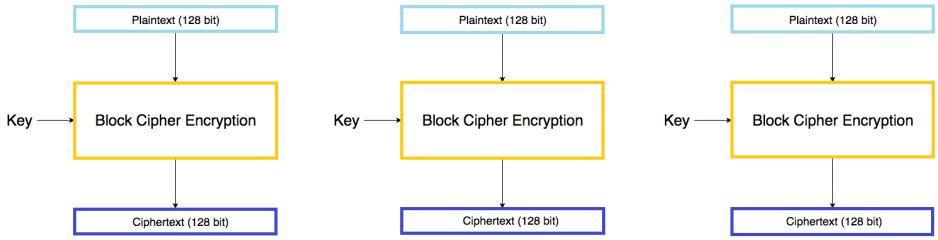

Figure 3.7: Electronic Codebook (ECB) Encryption Scheme

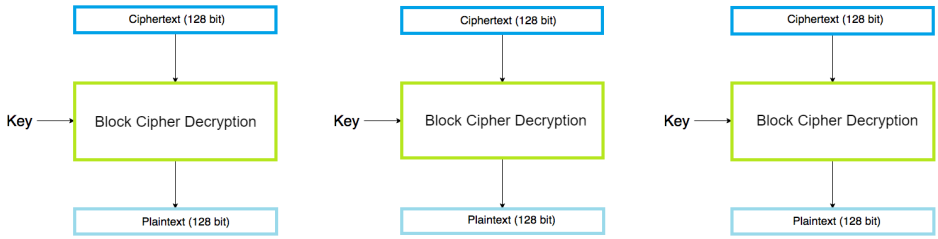

Figure 3.8: Electronic Codebook (ECB) Decryption Scheme

This is the simplest method because you have only to divide the plaintext and encrypt independently each part, but it is not so efficient (no high confidentiality).

3.6 Cipher Block Chaining (CBC)

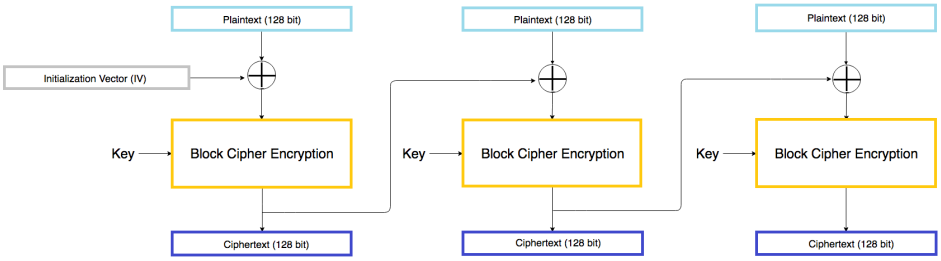

Figure 3.9: Cipher Block Chaining (CBC) Encryption Scheme

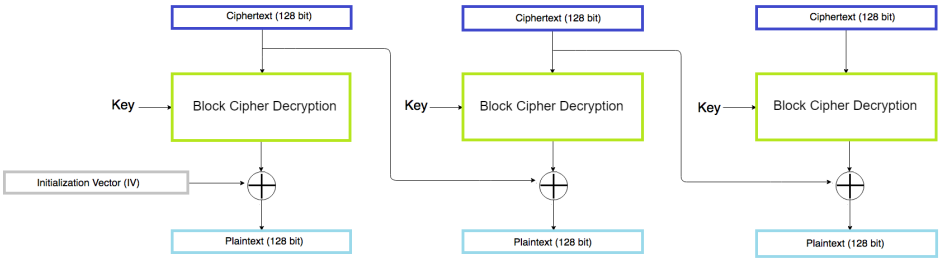

Figure 3.10: Cipher Block Chaining (CBC) Decryption Scheme

This is the most used modality and you have to do a XOR operation between the plaintext and the previous ciphertext (or an Initilization Vector (IV) in the case of the first block) before the encryption block.
The corresponding mathematical implementation is:

$$C_i = E_K(P_i \oplus C_{i-1})$$

$$C_0 = IV$$

As consequence, for the decryption:

$$P_i = D_K(C_i \oplus C_{i-1})$$

$$C_0 = IV$$

The main disadvantages of this mode are that it cannot be parallelized since is sequential and you need a length of the message corresponding to a multiple of the cipher block size.
An interesting difference between encryption and decryption is that in the first case an error in the plaintext or in IV has consequences on all the following blocks, while in the second case the error won't be propagated.

3.7 Propagating Cipher Block Chaining (PCBC)

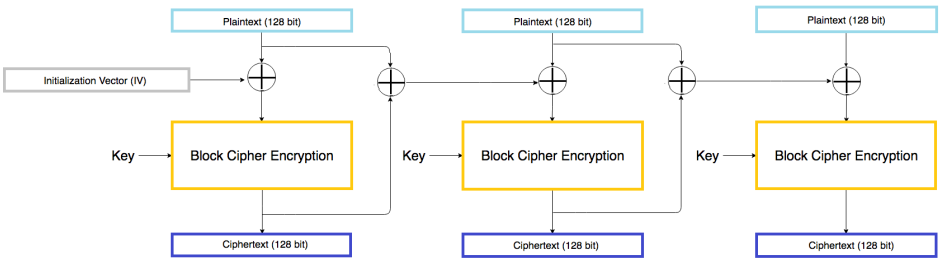

Figure 3.11: Propagating Cipher Block Chaining (PCBC) Encryption Scheme

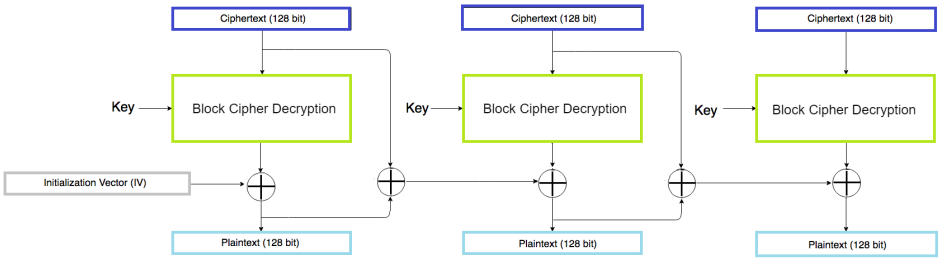

Figure 3.12: Propagating Cipher Block Chaining (PCBC) Decryption Scheme

Each subdivision of the plaintext has to be XORed with the previous plaintext and ciphertext (or for the first block only with the IV), that corresponds (for encryption) to:

$$C_i = E_K(P_i \oplus P_{i-1} \oplus C_{i-1})$$

$$P_0 \oplus C_0 = IV$$

While for decryption:

$$P_i = D_K(C_i) \oplus P_{i-1} \oplus C_{i-1}$$

$$P_0 \oplus C_0 = IV$$

3.8 Cipher Feedback (CFB)

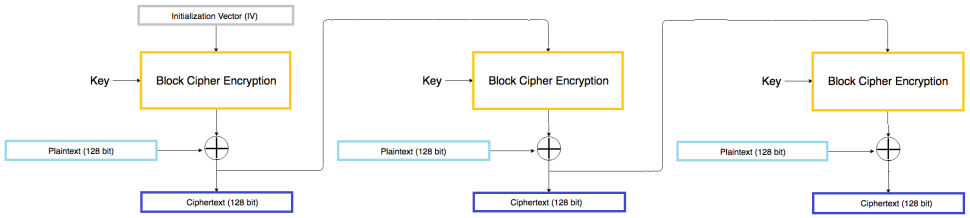

Figure 3.13: Cipher Feedback (CFB) Encryption Scheme

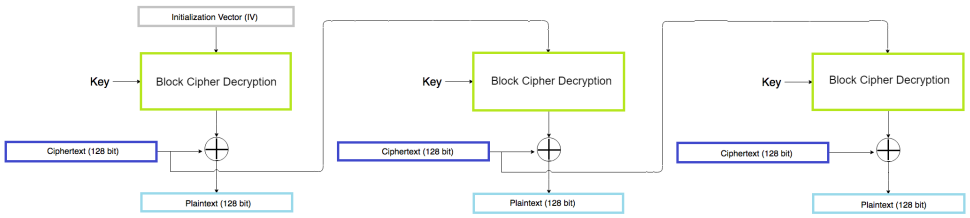

Figure 3.14: Cipher Feedback (CFB) Decryption Scheme

In this case you create a self-synchronizing stream cipher (if you loose part of the ciphertext due to a transmission error, you'll miss only part of the original message and you'll be able to continue correctly the decryption), using the three formulas below:

$$C_i = E_K(C_{i-1}) \oplus P_i$$

$$P_i = E_K(C_{i-1}) \oplus C_i$$

$$C_0 = IV$$

This method has some points in common with CBC, like that the propagation of the errors in the plaintext, you can have parallelization in decryption, but not in encryption and finally the propagation of the errors during decryption is stopped.

3.9 Output Feedback (OFB)

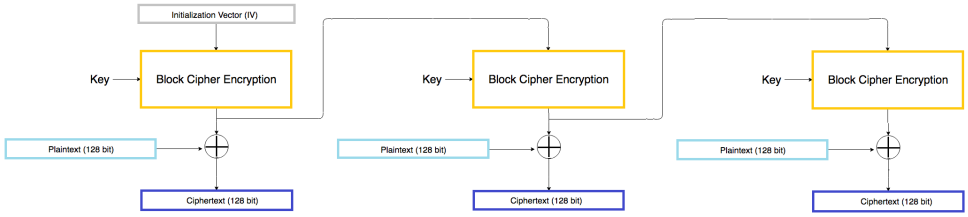

Figure 3.15: Output Feedback (OFB) EncryptionScheme

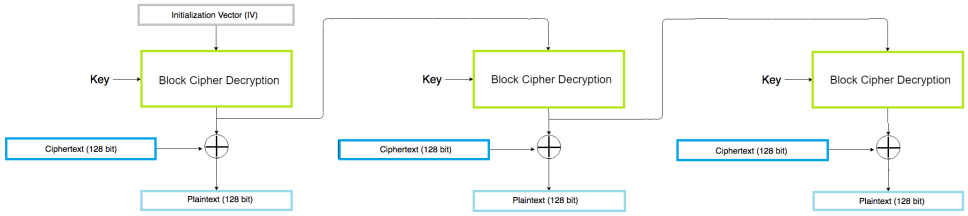

Figure 3.16: Output Feedback (OFB) Decryption Scheme

Also in this case you get a synchronous stream cipher, but now implementing:

$$C_j = P_j \oplus O_j$$

$$P_j = C_j \oplus O_j$$

$$O_j = E_K(I_j)$$

$$I_j = O_{j-1}$$

3.10 Counter (CTR)

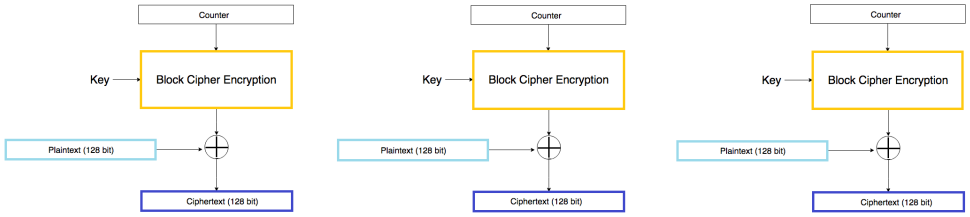

Figure 3.17: Counter (CTR) Encryption Scheme

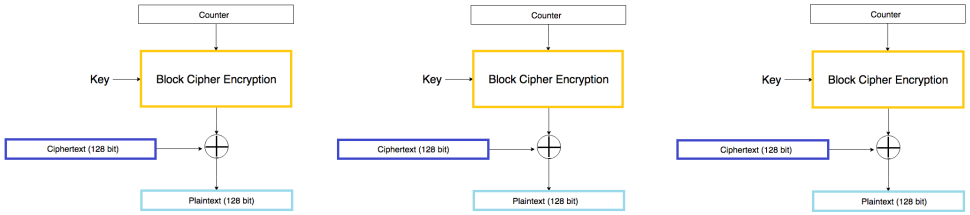

Figure 3.18: Counter (CTR) Decryption Scheme

Finally, you could also exploit a counter (with module high enough), encrypting its successive values getting a stream cipher.

3.11 State of the Art

In "Arithmetic Cryptography" [39] the authors studied the possibility of solving cryptographic problems in a way that is independent from the underlying algebraic domain (fully black-box arithmetic model over a finite field F). They proved several positive and negative results in this model for various cryptographic tasks: on the positive side, under coding-related intractability assumptions, computational primitives like commitment schemes, public-key encryption, oblivious transfer, and general secure two-party computation can be implemented in this model; on the negative side, garbled circuits, additively homomorphic encryption, and secure computation with low online complexity cannot be achieved in this model. The results reveal a qualitative difference between the standard Boolean model and the arithmetic model, and explain, in retrospect, some of the limitations of previous constructions.

The authors of "Lattice-based Cryptography" [40], showed that Lattice Based PKCS (Public Key Cryptosystems) are a promising field to introduce an immune cipher system against quantum cryptanalysis and they focused in particular on recent advancement in lattice-based public key cryptostystems and key exchange mechanisms based on the learning with errors (LWE) problem and its ring variant Ring-LWE. They obtained the following results: a) Lattice-based cryptography remained very impractical until the introduction of Regev's LWE cryptosystem, that significantly reduced the key sizes, from matrices to vectors; b) the passive security is guaranteed from the LWE assumption: the public key and ciphertext are indistinguishable from uniformly random distributions; c) the active

security can be achieved either using the Fujisaki-Okamoto transformation as in Peikert's KEM, or using trapdoor functions as in the GGH cryptosystem; d) the generation encryption and decryption times in modern lattice-based cryptosystems are almost faster than in classical PKCs; e) large ciphertext expansion factor of around 30, compared to classical PKCs where the ciphertext has the exact same size of plaintext.

In "Chaos-Based Cryptography: A Brief Overview" [41], they discussed the integration of the chaos theory in the cryptography.

In "Cryptanalysis Techniques: An Example Using Kerberos" [9], Jennifer Kay provided a guide to techniques for analysing a cryptosystem, using Kerberos authentication system, that uses a trusted key server to keep track of the private keys of clients and servers, as well as to generate session keys for client-server interaction.

Chris Peikert in his paper [42] presented the construction of public-key cryptosystems that are secure assuming the worst-case hardness of approximating the minimum distance on n-dimensional lattices to within small poly(n) factors. The main technical innovation is a reduction from variants of the shortest vector problem to corresponding versions of the "learning with errors" (LWE) problem. Finally, he presented also the construction of a natural chosen ciphertext-secure cryptosystem having a much simpler description and tighter underlying worst-case approximation factor than prior schemes.

In the paper "An FPGA Design of AES Encryption Circuit with 128-bit Keys" [43], they use a Pipelined Partial Rolling (PPR) architecture for the AES algorithm on a Altera Stratix EP1S20F780C5 FPGA, reducing the amount of memory up to 75% while increasing memory efficiency to 9.6%.

In "Accelerating Private-Key Cryptography via Multithreading on Symmetric Multiprocessors"[44] is demonstrated, using Wisconsin Wind Tunnel II, that exploiting Interleaving Chiper Block Chaining (ICBC), proposed by the cryptography community to deliver high performance while maintaining high level of security, you can exploit the full processing power, achieving an encryption rate of 92 Mbytes/s on a 16-processor SMP at 1 GHz, reaching a factor of improvement of almost 10.

In "ECvisual: A Visualization Tool for Elliptic Curve Based Ciphers" [45], they presented a tool for the visualization of elliptic curves over the real field and over a finite field of prime order, computation of arithmetic operations, encryption and decryption, and conversion of plaintext to a point on an elliptic curve.

In "RSAvisual: A Visualization Tool for the RSA Cipher" [46], they presented as well a tool that permits the user to visualize the steps of the RSA cipher, do encryption and decryption, learn simple factorization algorithms, and perform some elementary attacks.

In "AESvisual: A Visualization Tool for the AES Cipher" [47], is presented a software allowing the user to visualize all the major steps of AES encryption and decryption.

The authors of "CoARX: A Coprocessor for ARX-based Cryptographic Algorithms" [48] described a novel crypto-coprocessor, named CoARX, supporting multiple cryptographic algorithms based on Addition, Rotation and exclusive-or operations is proposed. CoARX supports diverse ARX-based cryptographic primitives and compared to dedicated hardware implementations and general-purpose microprocessors, it offers excellent performance-flexibility trade-off including adaptability to resist generic cryptanalysis.

In "A High-Performance and Scalable Hardware Architecture for Isogeny-Based Cryptography" [49] is presented a high-performance and scalable architecture for isogeny-based cryptosystems, using the architecture in a fast, constant-time FPGA implementation of the quantum-resistant supersingular isogeny Diffie-Hellman (SIDH) key exchange protocol. They used a Virtex-7 FPGA, showing that their architecture is scalable by implementing at 83, 124, 168, and 252-bit quantum security levels. Moreover it is the first SIDH implementation at close to the 256-bit quantum security level to appear in literature and their implementation completes the SIDH protocol 2 times faster than performance-optimized software implementations and 1.34 times faster than the previous best FPGA implementation, both running a similar set of formulas. For a constant-time implementation of 124-bit quantum security SIDH on a Virtex-7 FPGA, they generated ephemeral public keys in 8.0 and 8.6 ms and generated the shared secret key in 7.1 and 7.9 ms for Alice and Bob, respectively. Finally, they showed that this architecture could also be used to efficiently generate undeniable and digital signatures based on supersingular isogenies.

CHAPTER 4

AES VHDL Code and Board Programming

4.1 SEcube™ Overview

The SEcube™ (Secure Environment cube) is a powerful chip which integrates three key security elements in a single package:

1. Low-power floating-point ARM Cortex-M4 processor (STM32F4 CPU)
2. Flexible and fast Field-Programmable-Gate-Array (MachXO2-7000 FPGA)
3. EAL5+ certified Security Controller (SLJ52G SECURITY CONTROLLER-SMART CARD) [11]

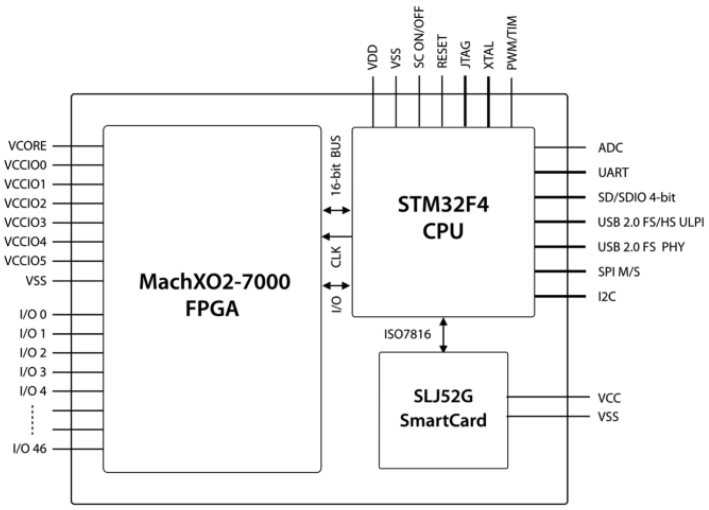

Figure 4.1: SEcube™ architectural overview [11]

Here you can observe the general scheme of the board:

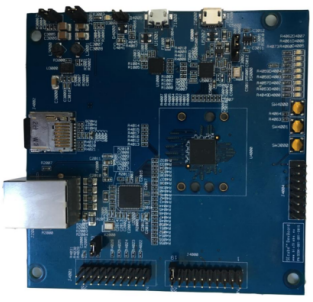

Figure 4.2: Photography of the SEcube™ board [11]

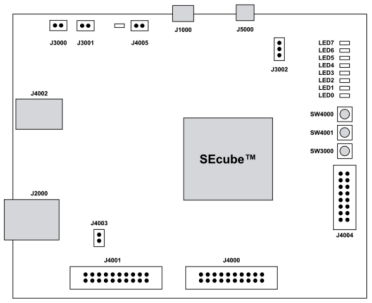

Figure 4.3: General scheme of the SEcube™ board [11]

In case of necessity, all the details about the board can be found in "SEcube^{TM} Development Kit - Getting Started" guide [11], that includes also the datasheet of the board.
In particular, the configuration of the pins was checked (starting from the one available on the SEcube^{TM} site `https://www.secube.eu`, in the folder `SEcube_SDK` → `Development` → `SEcubeDevBoard`) exploiting the tool STM32CubeMX, that is refereed to the microcontroller STM32F429NIHx, and with the help of the datasheet reported in [11].
For writing the `.lpf` file (requested by Lattice Diamond for the pin assignment), the reference was to the SEcube^{TM} internal connections and to the schematic reported below (fig.4.4-4.5):

Figure 4.4: SEcube^{TM} Internal Connections [11]

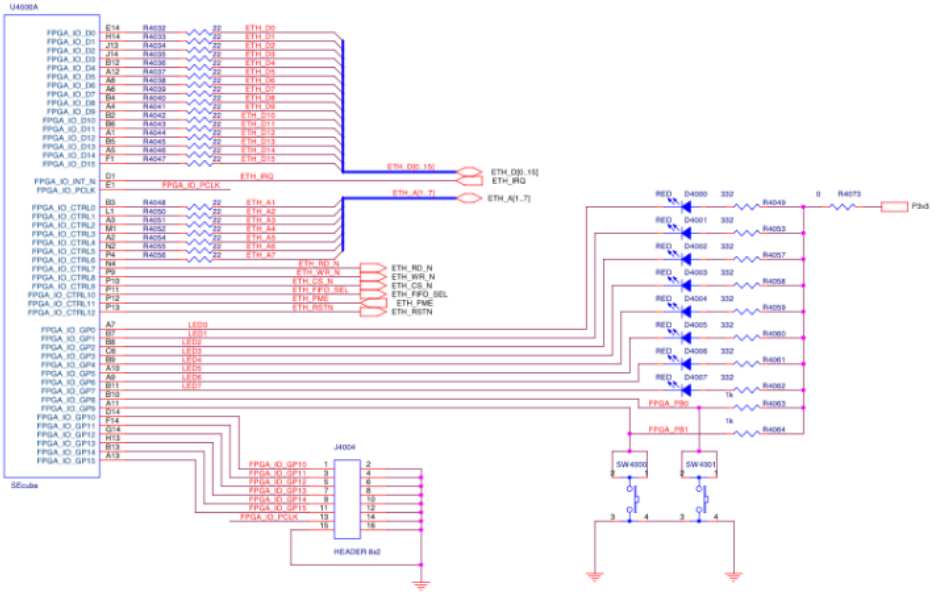

Figure 4.5: SEcube^{TM} Detailed Schematic [11]

4.2 Preparation of the AES-256 VHDL Code

4.2.1 ECB Mode

After some researches on the net and some synthesis attempts of the VHDL/Verilog codes available for AES-256 encryption, trying also to exploit the EBR RAM modules through IPexpress Tool provided by Lattice Diamond, in order to avoid area overhead, a VHDL structure organized in order to minimize the area paying some latency was chosen (available on the site `https://opencores.org` by Jerzy Gbur), in particular:

- Time of building key expansion : 157 clock cycles (to consider only after reset or if you want to change the key)
- Time of producing encoded data : 139 clock cycles (to consider every time you have a new plaintext and it is the range of time in which you have to analyze the power for the DPA attack)

The port map of the main entity is the following:

Port	Width	Direction
DATA_I	8	IN
VALID_DATA_I	1	IN
KEY_I	8	IN
VALID_KEY_I	1	IN
CLOCK_I	1	IN
RESET_I	1	IN
CE_I	1	IN
KEY_READY_O	1	OUT
VALID_0	1	OUT
DATA_O	8	OUT

Table 4.1: AES Core Port Map

Here is reported the RTL view given by Lattice Diamond (fig.4.6):

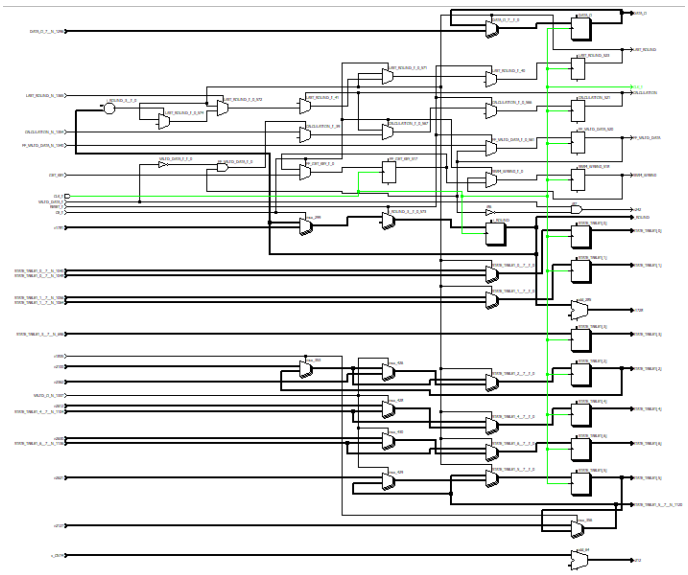

Figure 4.6: `aes_enc` RTL by Lattice Diamond

Starting from it, a VHDL code was written in order to define the environment in which it was desired to make work the AES-256, so a new top entity `aes_environment` was created, including `aes_enc` (the just described AES-256 code), `lfsr` and `ENCRYPTION_MANAGER` (fig. 4.7).

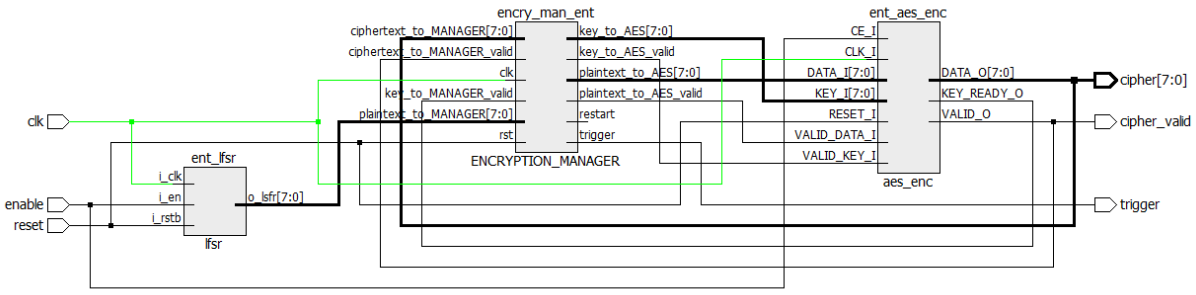

Figure 4.7: `aes_environment` RTL by Lattice Diamond

It was added a simple linear-feedback shift register (LFSR), that is a shift register whose input bit is a linear function of its previous state, in order to generate pseudo-random inputs for `aes_enc` and getting a large variety of inputs for maximizing the efficiency of my testbench (fig. 4.10).
It was decided to avoid using the classic LFSR configuration (also called many-to-one, fig.4.8), since the Galois LFSR implementation (also called modular, internal XORs, or one-to-many LFSR, fig.4.9) can generate the same output stream without having more than one level of combinational logic in the feedback path, so the structure is smaller and faster.

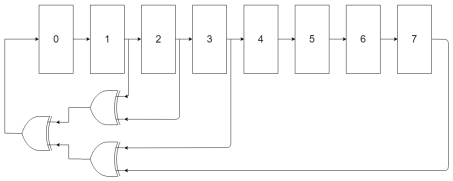

Figure 4.8: Many-to-one 8-bit LFSR

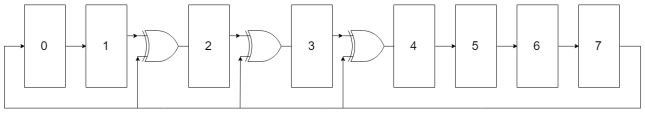

Figure 4.9: One-to-many 8-bit LFSR

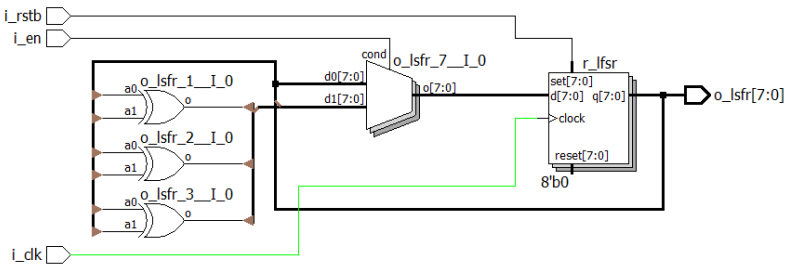

Figure 4.10: `lfsr` RTL by Lattice Diamond

The structure of `ENCRYPTION_MANAGER` instead is a bit more complicated and its goal is the correct management of the communication of the defined key to `aes_enc` and of sending (ciphertext) and receiving (plaintext) data.
Essentially it is organized like an FSM, whose block scheme is reported below (fig. 4.11):

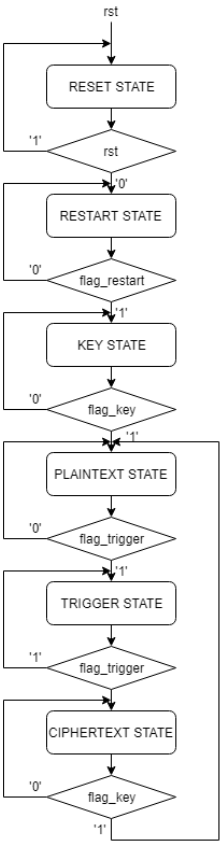

Figure 4.11: `ENCRYPTION_MANAGER` high level ASM

Here is reported a summary description of each state:

- RESET STATE: reset of the machine, setting all the signals like non-valid, putting zeros in all the registers and initializing all the counters;
- RESTART STATE: communication to `aes_enc` that the machine is exited from reset state and it has to prepare itself for the encryption;
- KEY STATE: the 256-bit-length key defined inside the block `ENCRYPTION_MANAGER` is communicated to `aes_enc`, that will be busy for 157 clock cycles in order to build the key expansion;
- PLAINTEXT STATE: communication of the entire plaintext in 16 clock cycles;
- TRIGGER STATE: it is the state where is the machine during the encryption time; this state is extremely important for the thesis goal since during this we have to consider the power consumption of the device that will be used for the application of the DPA attack;
- CIPHERTEXT STATE: communication of the entire ciphertext in 16 clock cycles.

The key is defined inside the `ENCRYPTION_MANAGER` since the target is the application of a DPA attack so obviously we want to maintain the key fixed (in this case, it is X"000102030405060708090a0b0c0d0e0f101112131415161718191a1b1c1d1e1f").

The RTL view reported by Lattice Diamond is the following (fig.4.12):

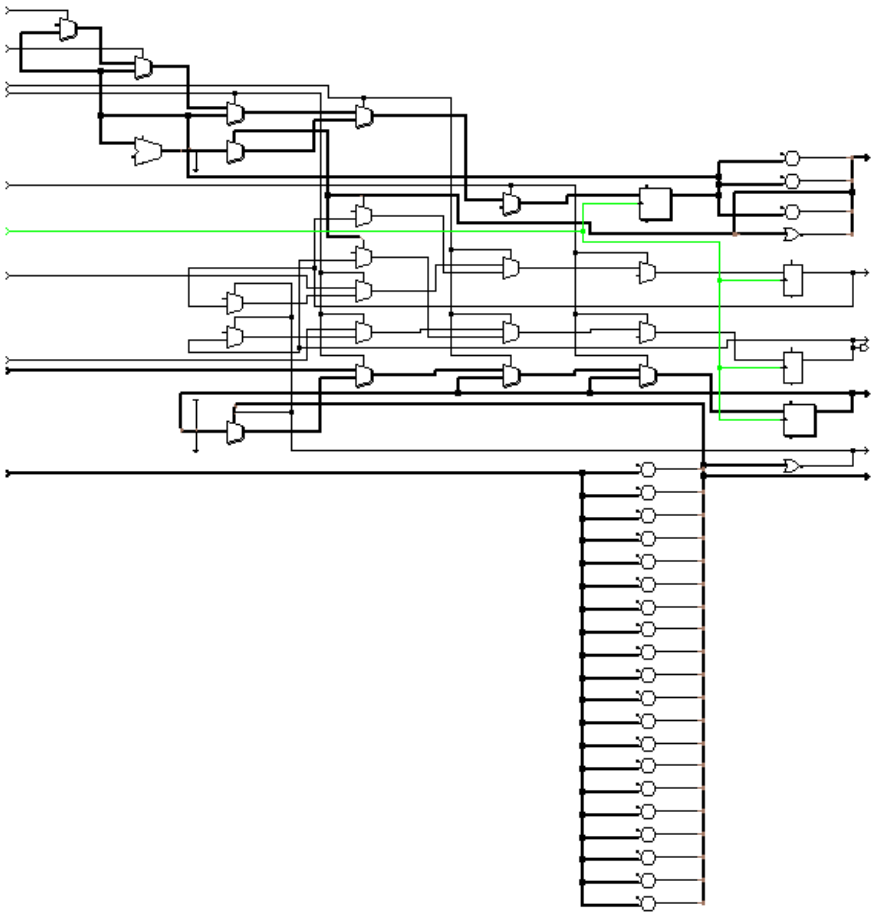

Figure 4.12: `ENCRYPTION_MANAGER` RTL by Lattice Diamond, ECB mode

The correct behaviour of the system was tested with a series of significant (since the data are provided by the LFSR so they are pseudo-random) testbenches:

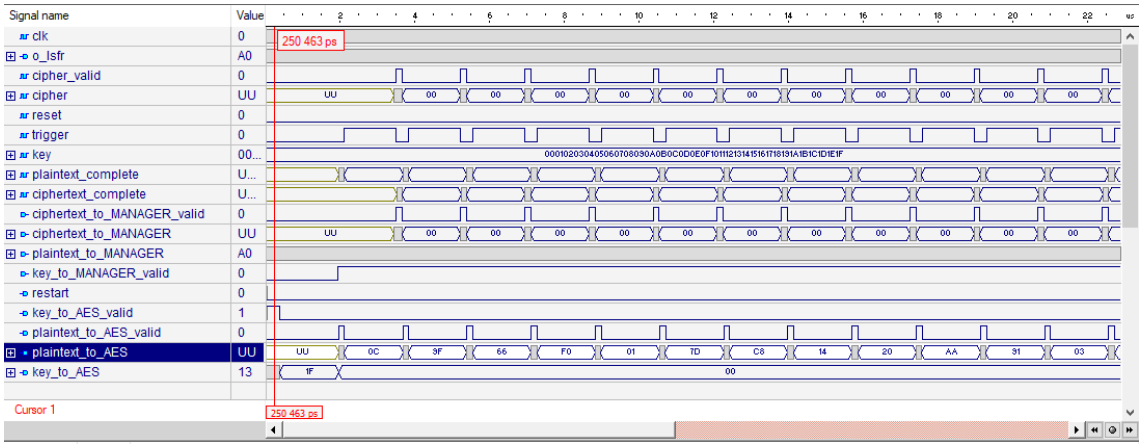

Figure 4.13: Testbench Overview, ECB mode

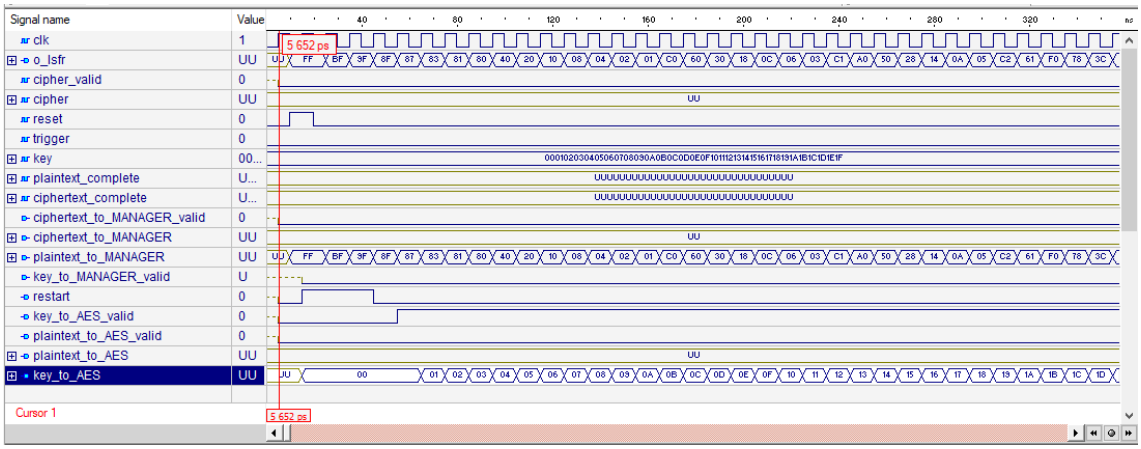

Figure 4.14: Testbench Focus on Reset and Communication of the Key, ECB mode

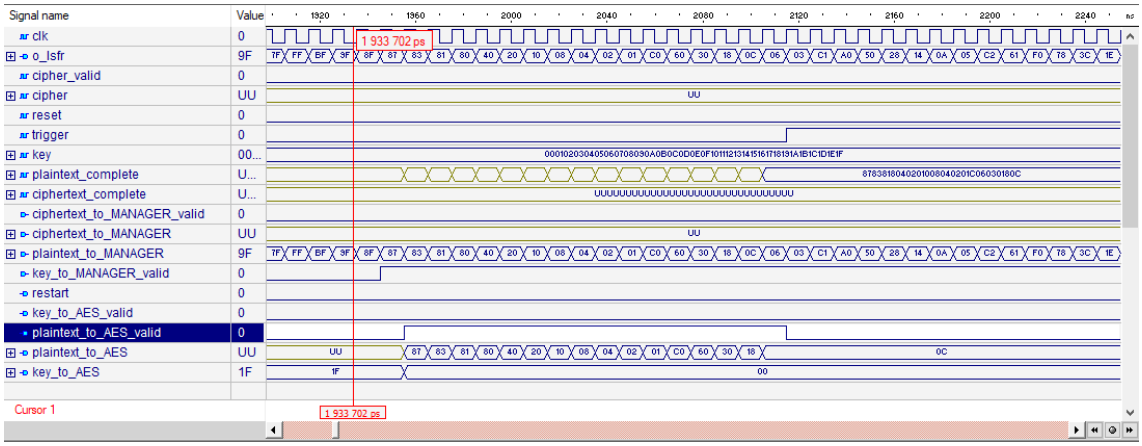

Figure 4.15: Testbench Focus on Communication of the Plaintext from LFSR, ECB mode

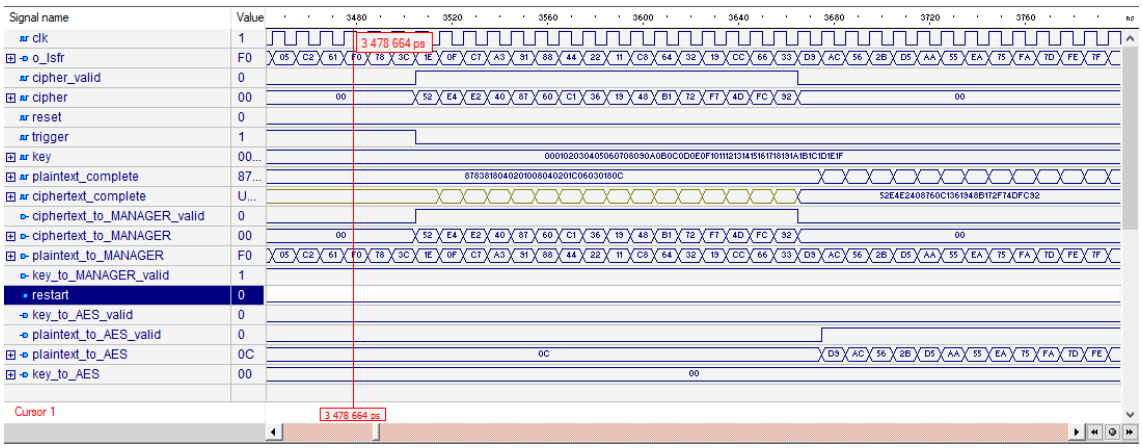

Figure 4.16: Testbench Focus on Communication of the first Ciphertext, ECB mode

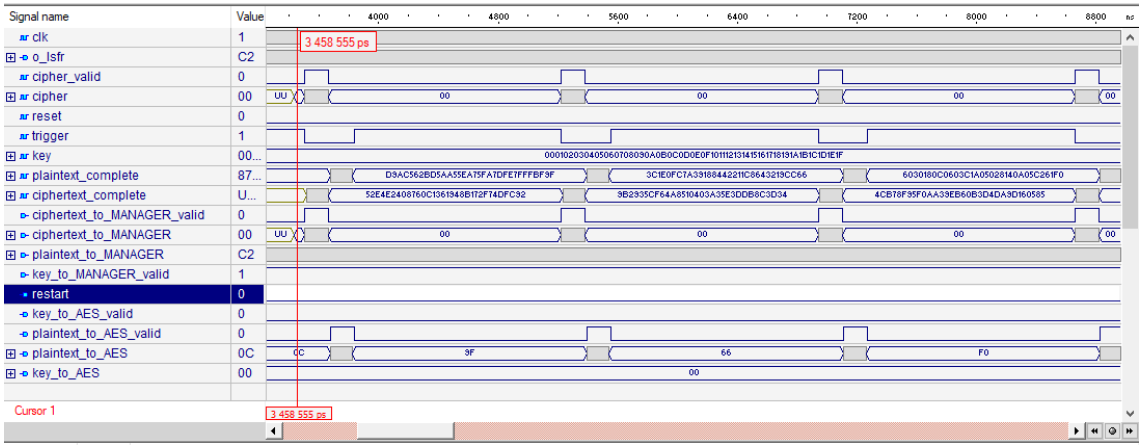

Figure 4.17: Testbench Focus on the second and the third Encryptions, ECB mode

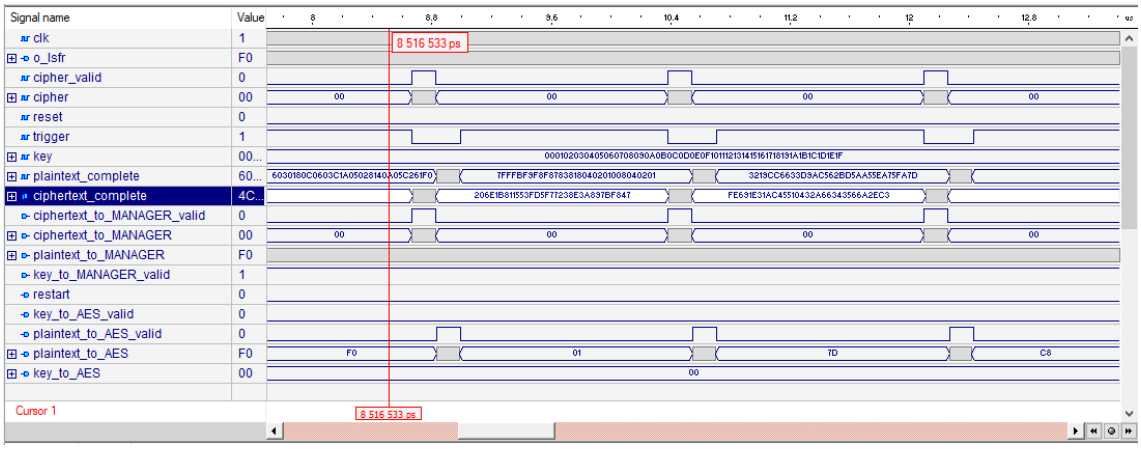

Figure 4.18: Testbench Focus on the fourth and the fifth Encryptions, ECB mode

The verification of the correctness of the encryption was done exploiting the online tool CRIPTOMAThIC (fig.4.19).

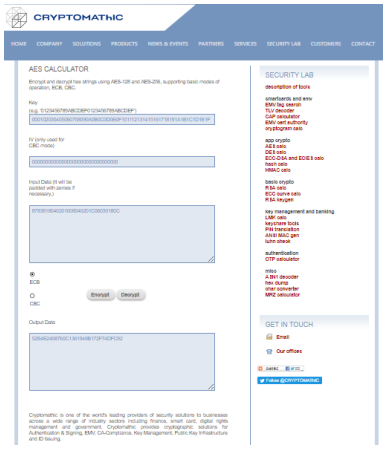

Figure 4.19: CRYPTOMAThIC Online Tool Main Page

Here are reported the values given by CRYPTOMAThIC for the cases considered in the images reported before:

- First Encryption:
  - Plaintext: X"87838180402010080402010C06030180C"
  - Ciphertext: X"52E4E2408760C1361948B172F74DFC92"
- Second Encryption:
  - Plaintext: X"D9AC562BD5AA55EA75FA7DFE7FFFBF9F"
  - Ciphertext: X"9B2935CF64A8510403A35E3DDB8C3D34"
- Third Encryption:
  - Plaintext: X"3C1E0FC7A39188442211C8643219CC66"
  - Ciphertext: X"4CB78F95F0AA39EB60B3D4DA9D160585"
- Fourth Encryption:
  - Plaintext: X"6030180C603C1A05028140A05C261F0"
  - Ciphertext: X"4CB78F95F0AA39EB60B3D4DA9D160585"
- Fifth Encryption:
  - Plaintext: X"7FFFBF9F8F8783818040201008040201"
  - Ciphertext: X"FE691E31AC45510432A66343566A2EC3"

The results given by Lattice Diamond reports are reported below (fig.4.20-4.21):

Design Summary

```
   Number of registers:    740 out of  7209 (10%)
      PFU registers:          740 out of  6864 (11%)
      PIO registers:            0 out of   345 (0%)
   Number of SLICEs:      1121 out of  3432 (33%)
      SLICEs as Logic/ROM:   1097 out of  3432 (32%)
      SLICEs as RAM:           24 out of  2574 (1%)
      SLICEs as Carry:         81 out of  3432 (2%)
   Number of LUT4s:        2236 out of  6864 (33%)
      Number used as logic LUTs:        2026
      Number used as distributed RAM:     48
      Number used as ripple logic:       162
      Number used as shift registers:      0
   Number of PIO sites used: 13 + 4(JTAG) out of 115 (15%)
   Number of block RAMs:  3 out of 26 (12%)
   Number of GSRs:  1 out of 1 (100%)
```

Figure 4.20: Design Summary, ECB mode

Timing Report Summary

```
--------------
------------------------------------------------------------------------------
Constraint                          |   Constraint|       Actual|Levels
------------------------------------------------------------------------------
                                    |             |             |
create_clock -period 5.000000 -name |             |             |
clk0 [get_nets clk_c]               |  200.000 MHz|   68.629 MHz|     9 *
                                    |             |             |
------------------------------------------------------------------------------
```

Figure 4.21: Timing Report Summary, ECB mode

Finally here is reported the technology view elaborated by Lattice Diamond (fig.4.22):

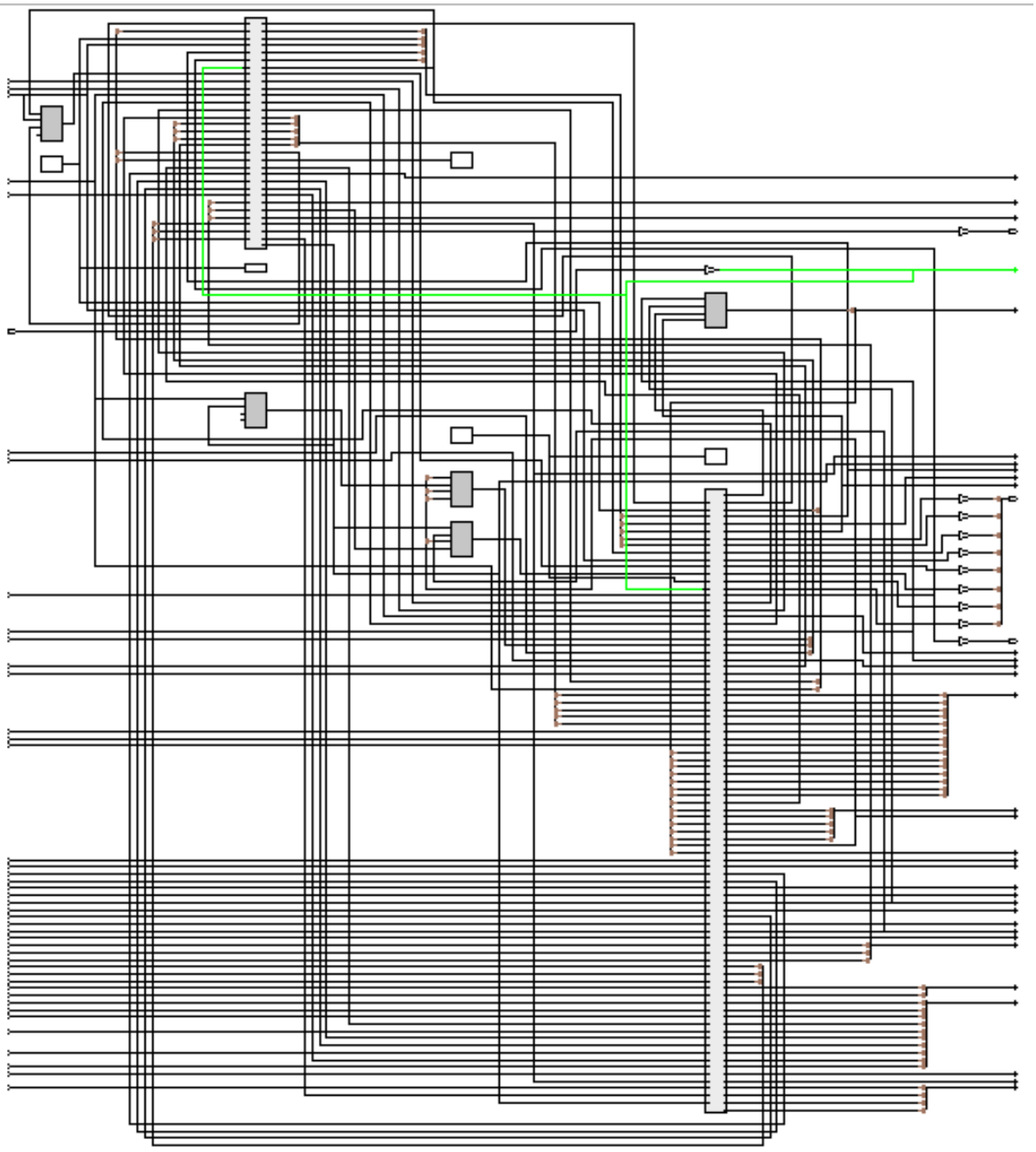

Figure 4.22: Technology View, ECB mode

4.2.2 CBC Mode

Since in "Chapter Three - Cryptography Overview" it is reported that CBC mode is extremely more efficient than ECB mode, the just described code was modified in order to get this configuration.
The same external appearance of the code was maintained, modifying only some details inside the `ENCRYPTION_MANAGER`, that are:

- In RESET STATE, it is present, in `ciphertext_complete`, the chosen Initialization Vector (IV);
- During PLAINTEXT STATE, instead of passing to `aes_enc` simply the plaintext from `lfsr`, it was passed xored with the corresponding part of `ciphertext_complete`.

Below are reported the main changes obtained (fig.4.23-4.26-4.27) and the verification of the correctness of the encryption (fig.4.24-4.25).

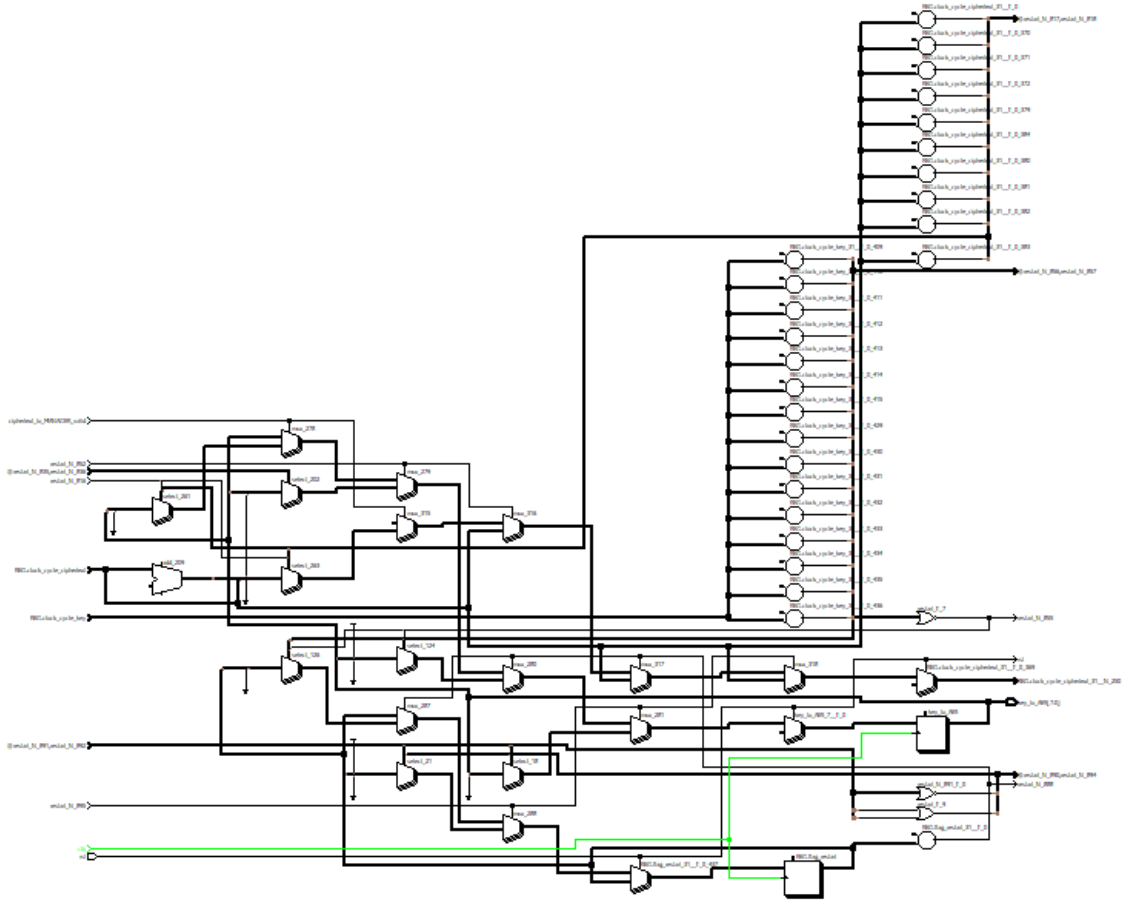

Figure 4.23: `ENCRYPTION_MANAGER` RTL by Lattice Diamond (CBC mode)

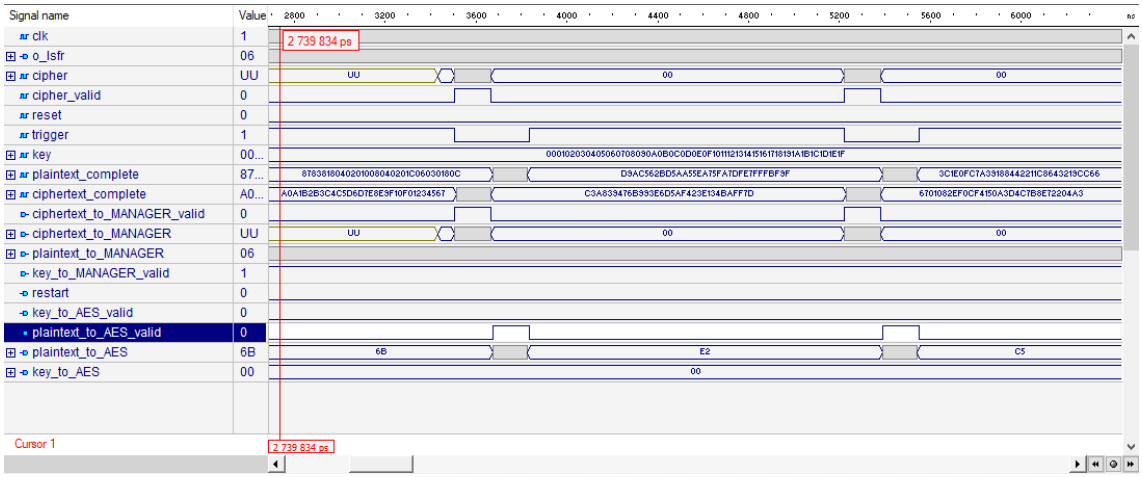

Figure 4.24: Testbench Focus on the first and the second Encryptions, CBC mode

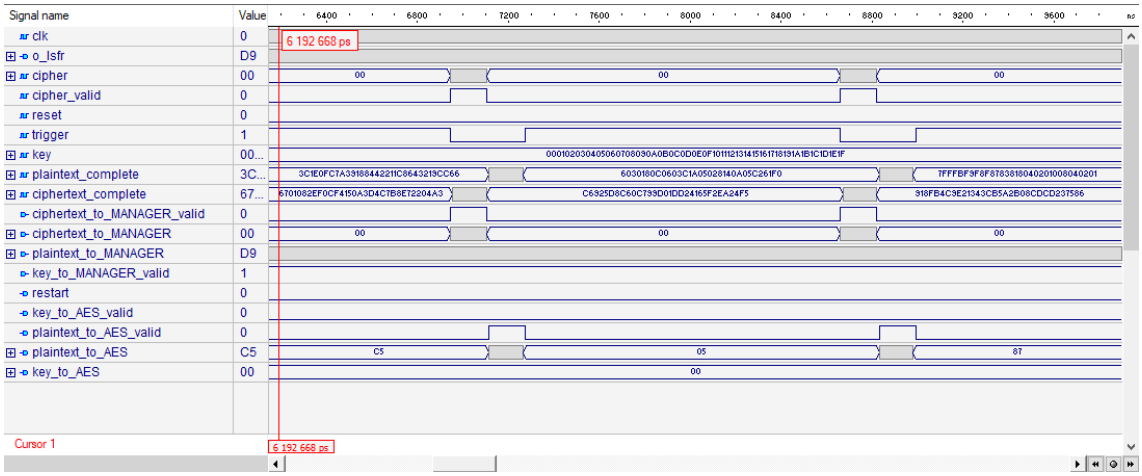

Figure 4.25: Testbench Focus on the third and the fourth Encryptions, CBC mode

- Initialization Vector (IV): X"A0A1B2B3C4C5D6D7E8E9F10F01234567"
- First Encryption:
  - Plaintext: X"87838180402010080402 01C06030180C"
  - Ciphertext: X"C3A839476B993E6D5AF423E134BAFF7D"
- Second Encryption:
  - Plaintext: X"D9AC562BD5AA55EA75FA7DFE7FFFBF9F"
  - Ciphertext: X"6701082EF0CF4150A3D4C7B8E72204A3"
- Third Encryption:
  - Plaintext: X"3C1E0FC7A39188442211C8643219CC66"
  - Ciphertext: X"C6925D8C60C799D01DD24165F2EA24F5"

- Fourth Encryption:
  - Plaintext: X"6030180C603C1A05028140A05C261F0"
  - Ciphertext: X"918FB4C9E21343CB5A2B08CDCD237586"

```
Design Summary
   Number of registers:    869 out of  7869 (11%)
      PFU registers:          869 out of  6864 (13%)
      PIO registers:            0 out of  1005 (0%)
   Number of SLICEs:      1193 out of  3432 (35%)
      SLICEs as Logic/ROM:   1169 out of  3432 (34%)
      SLICEs as RAM:           24 out of  2574 (1%)
      SLICEs as Carry:         81 out of  3432 (2%)
   Number of LUT4s:        2382 out of  6864 (35%)
      Number used as logic LUTs:        2172
      Number used as distributed RAM:     48
      Number used as ripple logic:       162
      Number used as shift registers:      0
   Number of PIO sites used: 13 + 4(JTAG) out of 335 (5%)
   Number of block RAMs:  3 out of 26 (12%)
   Number of GSRs:  1 out of 1 (100%)
```

Figure 4.26: Design Summary, CBC mode

```
Timing Report Summary
--------------
------------------------------------------------------------------------------
Constraint                              |   Constraint|       Actual|Levels
------------------------------------------------------------------------------
                                        |             |             |
create_clock -period 5.000000 -name     |             |             |
clk0 [get_nets clk_c]                   |  200.000 MHz|   61.166 MHz|    12 *
                                        |             |             |
------------------------------------------------------------------------------
```

Figure 4.27: Timing Report Summary, CBC mode

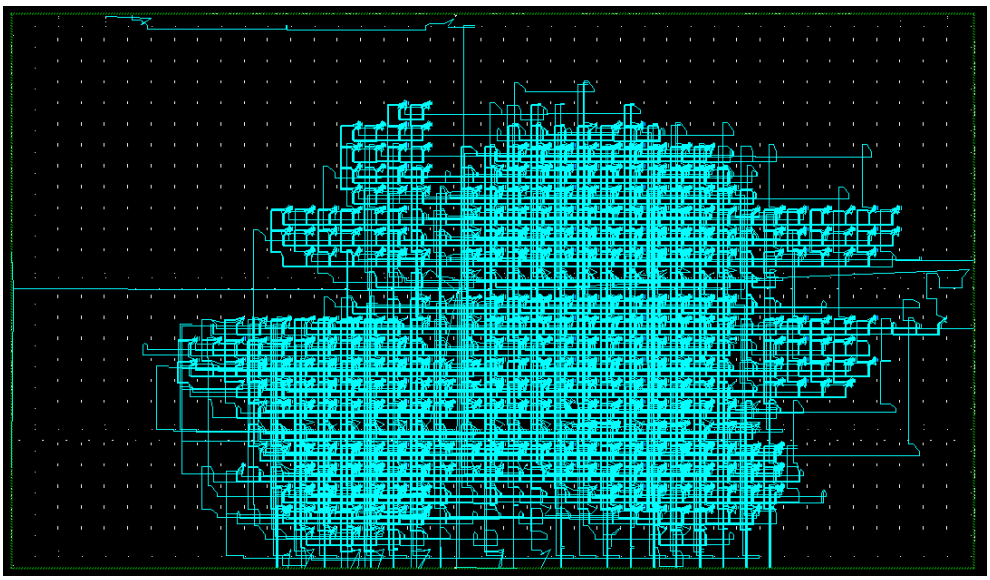

Figure 4.28: Physical View, CBC mode

4.2.3 Comparison between ECB and CBC Architectures

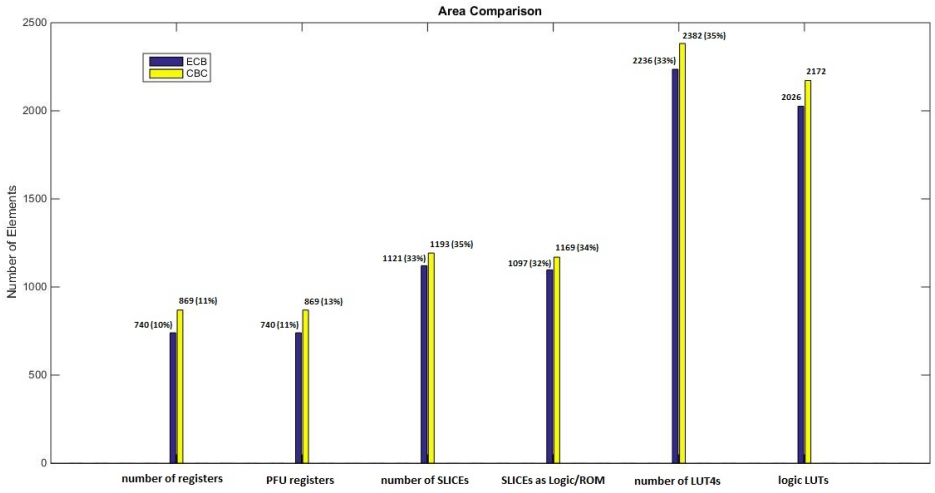

Figure 4.29: Area Comparison between ECB and CBC architectures

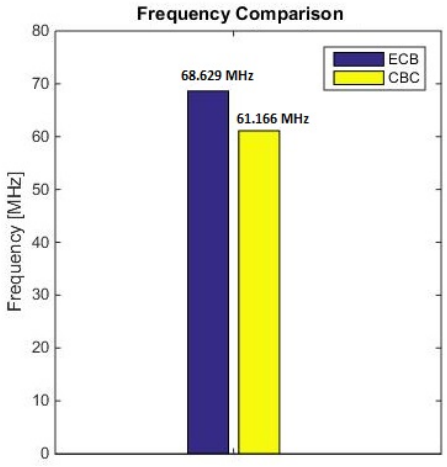

Figure 4.30: Frequency Comparison between ECB and CBC architectures

4.2.4 Modified AES-256 VHDL Code (CBC Mode) for the Laboratory Setting

After having obtained a valid code and having analysed his performances, it was modified in order to be able to take the necessary measurements in laboratory for the future application of the DPA attack. In particular, the `lfsr` module was substituted with another structure, that was called `PLAINTEXT_GENERATOR`, in order to generate the adapt inputs for the future application of the DPA attack (fig.4.31).

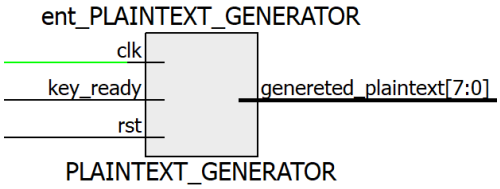

Figure 4.31: `PLAINTEXT_GENERATOR` RTL by Lattice Diamond

This entity is a FSM with 17 states (one IDLE state plus one state for each communication of a piece of plaintext), mainly driven by the signals `key_ready` (for indicating the end of the management of the key and so that you can start with the communication of the code) and `wait_finished` (associated to a counter of module 155, since you need 139 clock cycles for the encryption plus 16 clock cycles for the communication of the ciphertext). The transmitted plaintext is always the same, except for the last byte (0xBD0F0FFF3F2AF451B4C70000FF8732xx); indeed, in the corresponding state, it was used a module-256 counter instead of a fixed piece of plaintext, to obtain all the values from 00 to FF cyclically.

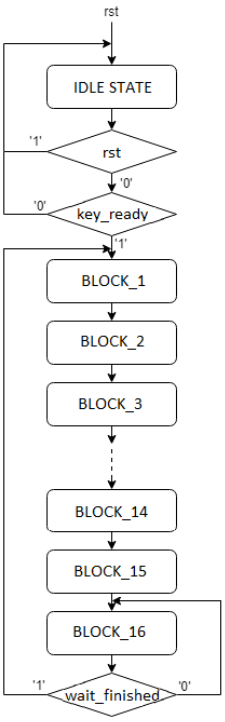

Figure 4.32: `PLAINTEXT_GENERATOR` high level ASM

In the top entity `aes_environment`, it was just substituted `lfsr` entity with `PLAINTEXT_GENERATOR`, connecting the opportune signals, then it was changed also the port map in order to be able to debug the board from the external environment through an optical inspection of the LEDs directly connected to the FPGA and through the measurement of the J4004 outputs with the digital oscilloscope (fig. 4.4-4.5). To sum up, the port map is:

Port	**Width**	**Direction**
clk	1	IN
rst	1	IN
led_out	8	OUT
testing_FPGA_IO	6	OUT

Table 4.2: `aes_environment` Port Map

For debugging, the following signals were used and connected to the outputs :

```
led_out(7) <= trigger_test;
led_out(6) <= cipher_from_AES_to_ENCRY_MAN(7);
led_out(5) <= cipher_from_AES_to_ENCRY_MAN(6);
led_out(4) <= clk;
led_out(3) <= restart_from_ENCRY_MAN_to_AES;
led_out(2) <= reset;
led_out(1) <= valid_output_from_AES_to_ENCRY_MAN; -- cipher_valid
led_out(0) <= valid_data_from_ENCRY_MAN_to_AES; -- plain_valid
testing_FPGA_IO(5) <= trigger_test;
testing_FPGA_IO(4) <= valid_output_from_AES_to_ENCRY_MAN; -- cipher_valid
testing_FPGA_IO(3) <= valid_data_from_ENCRY_MAN_to_AES; -- plain_valid
testing_FPGA_IO(2) <= clk;
testing_FPGA_IO(1) <= cipher_from_AES_to_ENCRY_MAN(7);
testing_FPGA_IO(0) <= reset;
```

Here some of the testbenches used for verifying the new version of the code are reported (fig.4.33-4.34-4.35-4.36):

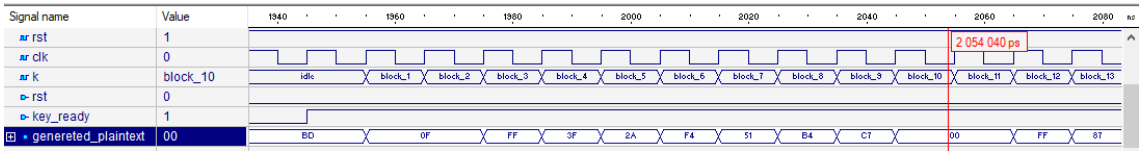

Figure 4.33: Testbench Focus on the First Transmission by `PLAINTEXT_GENERATOR`

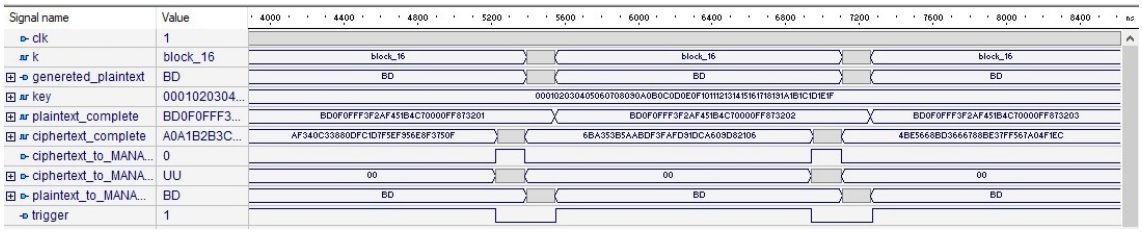

Figure 4.34: Testbench Focus on the Second, Third and Fourth Sequential Transmissions by `PLAINTEXT_GENERATOR`

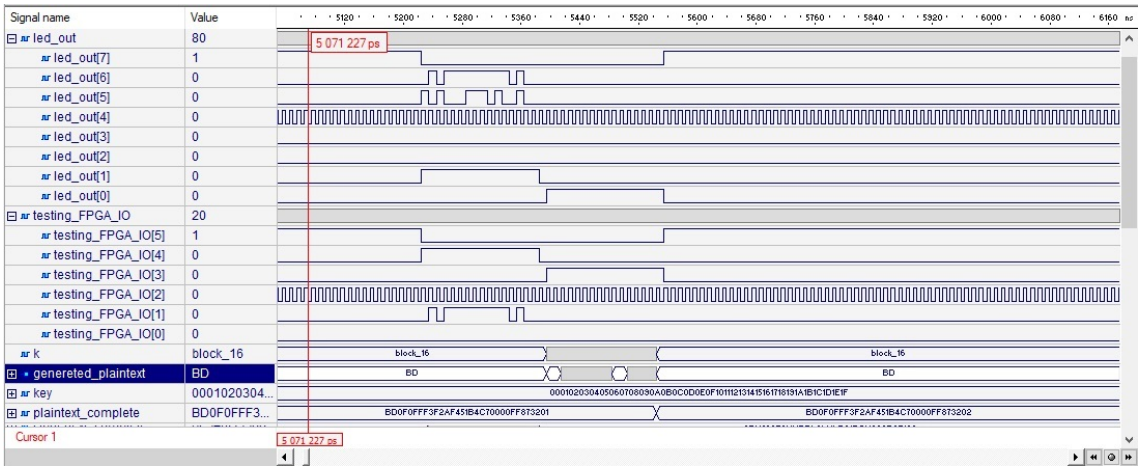

Figure 4.35: Testbench Focus on the Debug Signals referred to the Second Transmission by `PLAINTEXT_GENERATOR`

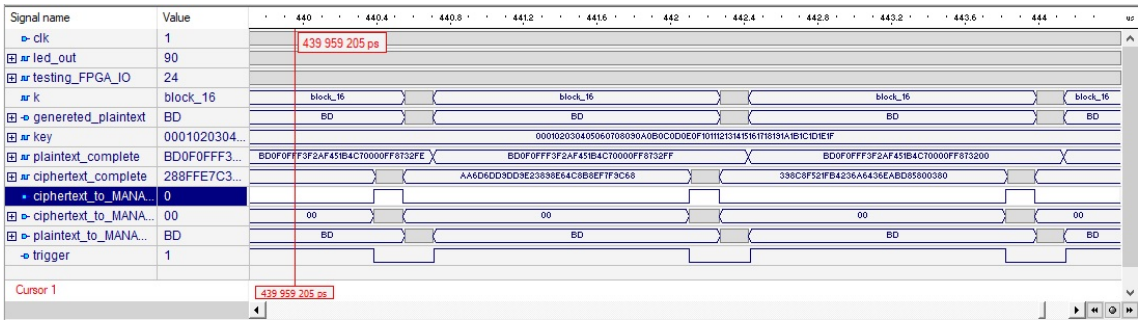

Figure 4.36: Testbench Focus on the Last Transmission and on the Cycle Restart by `PLAINTEXT_GENERATOR`

Some details of the reports are reported below:

```
Design Summary
   Number of registers:    919 out of  7869 (12%)
      PFU registers:          919 out of  6864 (13%)
      PIO registers:            0 out of  1005 (0%)
   Number of SLICEs:      1228 out of  3432 (36%)
      SLICEs as Logic/ROM:   1204 out of  3432 (35%)
      SLICEs as RAM:           24 out of  2574 (1%)
      SLICEs as Carry:        115 out of  3432 (3%)
   Number of LUT4s:        2442 out of  6864 (36%)
      Number used as logic LUTs:        2164
      Number used as distributed RAM:     48
      Number used as ripple logic:       230
      Number used as shift registers:      0
   Number of PIO sites used: 16 + 4(JTAG) out of 335 (6%)
   Number of block RAMs:  3 out of 26 (12%)
   Number of GSRs:  1 out of 1 (100%)
```

Figure 4.37: Design Summary, CBC Mode with `PLAINTEXT_GENERATOR`

Timing Report Summary

Constraint	Constraint	Actual	Levels
create_clock -period 5.000000 -name clk0 [get_nets n7800_c]	200.000 MHz	71.679 MHz	10 *

Figure 4.38: Timing Report Summary, CBC Mode with PLAINTEXT_GENERATOR

4.2.5 Comparison between CBC with LFSR and CBC with `PLAINTEXT_GENERATOR`

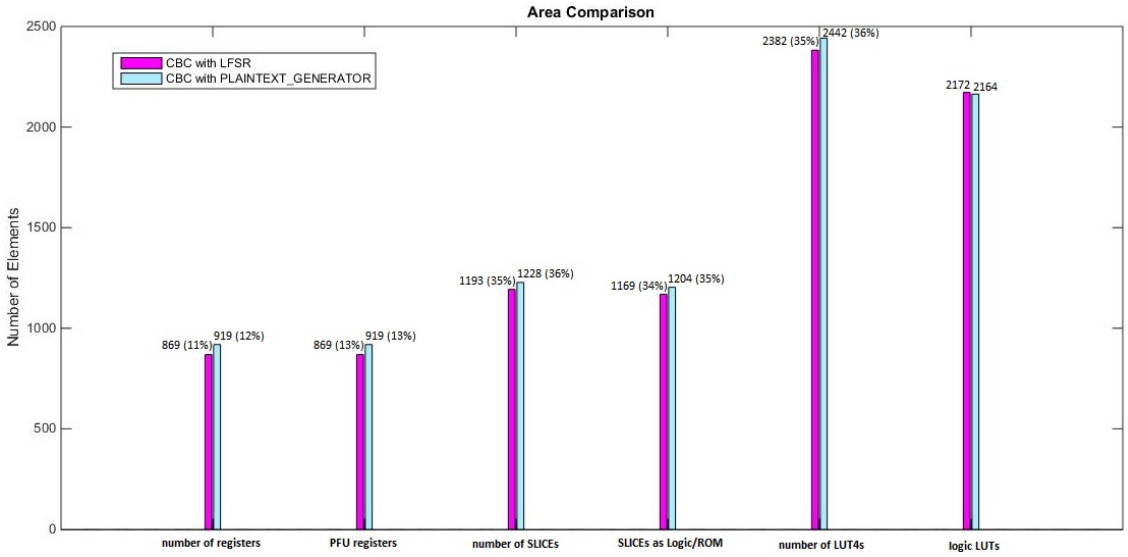

Figure 4.39: Area Comparison between CBC Architectures with LFSR and with `PLAINTEXT_GENERATOR`

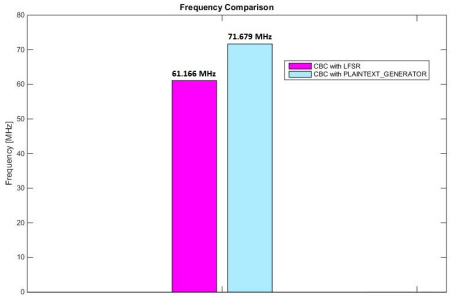

Figure 4.40: Frequency Comparison between CBC Architectures with LFSR and with `PLAINTEXT_GENERATOR`

4.3 Exportation of the VHDL Code

At this point the VHDL code was ready for the exportation in the tool System Workbench for STM32, through which you can program the FPGA module inside SEcube^{TM}.
First of all, the right configurations were checked and set . In particular, from `Project` → `Device...`, it was selected the family MachXO2, the device LCMXO2-7000HE and the package type FPBGA484; then, from `Project` → `Property Pages`, it was clicked on our implementation and it had been checked that the Synthesis Tool was *Lattice LSE*.

As you can read from "MachXO2 sysCLOCK PLL - Design and Usage Guide", the FPGAs of the family MachXO2 have an internal oscillator with an accuracy of ±5% and the available output frequencies are shown in the following table (fig.4.41):

2.08	4.16	8.31	15.65
2.15	4.29	8.58	16.63
2.22	4.43	8.87	17.73
2.29	4.59	9.17	19.00
2.38	4.75	9.50	20.46
2.46	4.93	9.85	22.17
2.56	5.12	10.23	24.18
2.66	5.32	10.64	26.60
2.77	5.54	11.08	29.56
2.89	5.78	11.57	33.25
3.02	6.05	12.09	38.00
3.17	6.33	12.67	44.33
3.33	6.65	13.30	53.20
3.50	7.00	14.00	66.50
3.69	7.39	14.78	88.67
3.91	7.82	15.65	133.00

Figure 4.41: Available Frequencies for the Internal Oscillator, from "MachXO2 sysCLOCK PLL - Design and Usage Guide"

The fragment of VHDL code is reported:

```
-- INTERNAL OSCILLATOR (SIGNALS AND COMPONENT)
    -- internal signal declarations
    signal iClk: std_logic;
    signal stdby_sed: std_logic;
    signal rTP      : std_logic_vector(13 downto 0);
    signal rFreqDown        : std logic vector(27 downto 0);
    -- local component declarations
    COMPONENT OSCH
    -- synthesis translate_off
    GENERIC (NOM_FREQ: string := "66.50");
    -- synthesis translate_on
    PORT (STDBY : IN std_logic;
        OSC : OUT std_logic;
        SEDSTDBY: OUT std_logic);
    END COMPONENT;
    attribute NOM_FREQ : string;
    attribute NOM_FREQ of OSCinst0 : label is "66.50";
```

```
-- INTERNAL OSCILLATOR (ISTANCE)
    -- component instantiation statements
    OSCInst0: OSCH
    -- synthesis translate_off
    GENERIC MAP
        (
        NOM_FREQ => "66.50"
        )
    -- synthesis translate_on
    PORT MAP
        (
        STDBY => '0',
        OSC => iClk,
        SEDSTDBY => stdby_sed
        );
```

Figure 4.42: Internal Oscillator VHDL Code

Starting from the schematics reported before (fig.4.4-4.5), in order to assign the desired pins to the input/outputs of our architecture, it was created a `.lpf` file (Diamond Project Logical Preference File, used for storing logical constraints for developing and implementing the design):

```
BLOCK RESETPATHS ;
BLOCK ASYNCPATHS ;
LOCATE COMP "led_out[0]" SITE "PR16C" ;
LOCATE COMP "led_out[1]" SITE "PR16D" ;
LOCATE COMP "led_out[2]" SITE "PR17C" ;
LOCATE COMP "led_out[3]" SITE "PR17D" ;
LOCATE COMP "led_out[4]" SITE "PR18C" ;
LOCATE COMP "led_out[5]" SITE "PR18D" ;
LOCATE COMP "led_out[6]" SITE "PR19A" ;
LOCATE COMP "led_out[7]" SITE "PR19B" ;
LOCATE COMP "rst" SITE "PR19C" ;
LOCATE COMP "testing_FPGA_IO[0]" SITE "PR21C" ;
LOCATE COMP "testing_FPGA_IO[1]" SITE "PR21D" ;
LOCATE COMP "testing_FPGA_IO[2]" SITE "PR22A" ;
LOCATE COMP "testing_FPGA_IO[3]" SITE "PR22B" ;
LOCATE COMP "testing_FPGA_IO[4]" SITE "PR22C" ;
LOCATE COMP "testing_FPGA_IO[5]" SITE "PR22D" ;
```

Figure 4.43: `.lpf` File

At this point, our implementation was re-run from the Run Manager of Lattice Diamond, then from the panel on the left, it was clicked on the Process Tab and, from there, first the JEDEC File (used for Flash FPGA programming) and then the Bitstream File (for the generation of a configuration bitstream (bit images) file were run. They contained all of the design's configuration information that defines the internal logic and interconnections of the FPGA, as well as device-specific information from other files).
After getting these fundamental files for the exportation, the Lattice Diamond Programmer tool was opened: it was checked that all the information was consistent with the ones previously setted and it was added the `.jed` file in the corresponding gap.
Subsequently, the "Cable Settings" was modified, selecting the cable "HW-USBN-2B (FTDI)" and the port "FTUSB-0". In the section "Programming Speed Settings", it was chosen "Use default Clock Divider", while in the section "I/O Settings", the option "Use default I/O settings".
Finally, the `.xcf` file (Configuration Chain File, used for programming devices in a JTAG daisy chain) was saved.

Afterwards, the `.xcf` file was converted to a vector of bitstreams. For doing this, the Diamond Deployment Tool application was opened and a new deployment was created, selecting "Embedded System" as Function Type and "JTAG Slim VME Embedded" as Output File Type. In the next passage I added the just saved `.xcf` file and then I selected both the options "Compress VME Data File" and "Convert VME file to HEX(.c) for Prom-Based Embedded VME".
After using the command "Generate", the two files *impl_name*`_algo.c` and *impl_name*`_data.c`, needed for the preparation of the C code in the System Workbench for STM32 environment, were obtained.

4.4 Preparation of the C Code

Referring to the starting guide for the SEcube^{TM} board ([11]), after downloading `Environment.zip` (inside the SDK package that you can find on the download page of the site `https://www.secube.eu`) and extracting its content, Eclipse through System Workbench for STM32 environment was launched and it was checked that all the settings were complaining with the ones specified in [11]. The Eclipse workspace was switched to the folder `ws` contained in the just extracted folder, then the project was run firstly in Debug Mode and then in Release Mode.
At this point, the DevKit (SEcube^{TM} board and ST-Link/v2 programmer) was carefully connected, as described in the guide, then right-clicking on the main folder of the project, it was chosen `Target` → `Program Chip` and it was selected "Reset after programming. When the programming process finished, getting the SEcube^{TM} board fully configured and the STM32 microprocessor ready to be used to develop applications.

After testing the FPGA module inside the SEcube^{TM} board with the example available on the site for blinking the LEDs directly connected, from `File` → `Import...` it was selected "Filesystem", then it was clicked on "Next" and it was browsed the directory "SEcube_SDK\Libraries\Examples\TestFPGA\" and all the contained files were selected. In the following passage it was set "SEcubeDevBoard\Application\src" as "Destination Folder", then it was clicked on finished and the configuration of the builder was checked from the project properties.

Finally, the file `main.c` was opened: here there were included the header file `FPGA.h` and there was added a call to the function `B5_FPGA_Programming()`. After it was opened the file `gpio.c` and, inside the function `MX_GPIO_Init()`, the lines of code reported in the guide for the configuration of the GPIO pins (dedicated to the FPGA programming) were added. Then it was built the just modified project and it was re-programmed the SEcube^{TM} board. After this, it was observed that the LEDs were blinking and so the correct behaviour of the board was verified.

Later, the `TEST_FPGA.h` file was modified, substituting the content of `const uint8_t __fpga_alg[]` and `const uint8_t __fpga_data[]` vector with the values that compose the vectors `xdata const unsigned char g_pucAlgoArray[129857]` and `xdata const unsigned char g_pucDataArray[202653]` (paying attention to maintain exactly the same format, so in particular you have to add a comma after the last value before closing the brace) contained respectively in the files *impl_name*`_algo.c` and *impl_name*`_data.c`, generated by Diamond Deployment Tool. It was saved the file then it was modified also the file `FPGA.c`, changing the sizes of the vectors that become: `uint32_t g_iAlgoSize = 129857;` and `uint32_t g_iDataSize = 202653;`.
At last, the project was re-built and the SEcube^{TM} board was programmed as described before.

During this phase, for some additional checks of the correct programming of the board, it was exploited also the program "STM32 ST-LINK Utility" (fig.4.44). Indeed, through this tool, you can

get useful information about the used microcontroller and in particular the data present inside the flash memory in that moment; moreover you can reset it in the case you'd have some issues using "System Workbench for STM32" and verify if everything it is like expected.

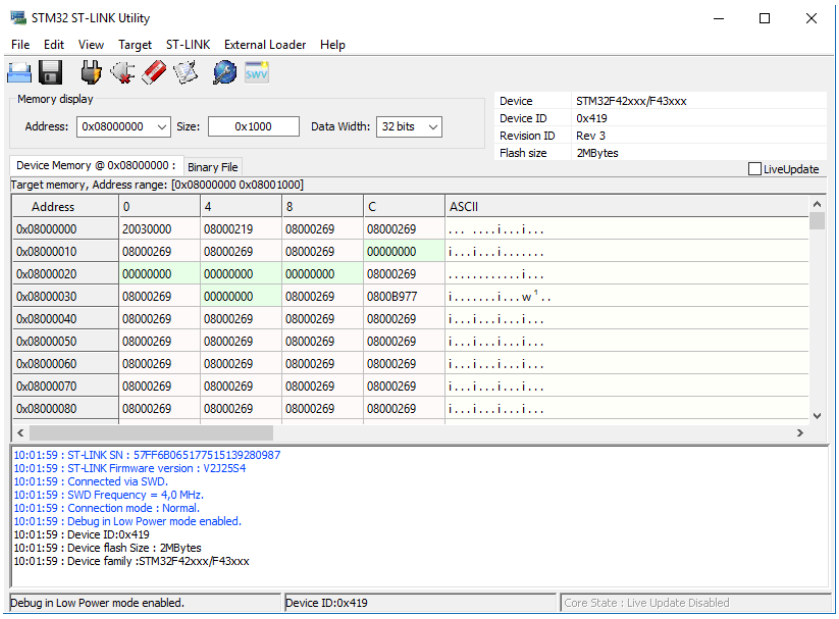

Figure 4.44: STM32 ST-LINK Utility

CHAPTER 5

Measurements in Laboratory

For taking the measurements it was used a digital oscilloscope LeCroy waveRunner 6030, with a 350 MHz bandwidth, that can reach a sample rate of 2.50 GS/s.

First of all, the correct behaviour of the board was verified exploiting all the testing signals that were defined as described in the previous sections. Just for example, a photo where the channel 1 (yellow trace) is connected to the trigger signal, the channel 2 (pink trace) to the signal that indicates the transmission of the plaintext and the channel 3 (blue trace) to the one related to the transmission of the ciphertext, is reported below (fig.5.1).

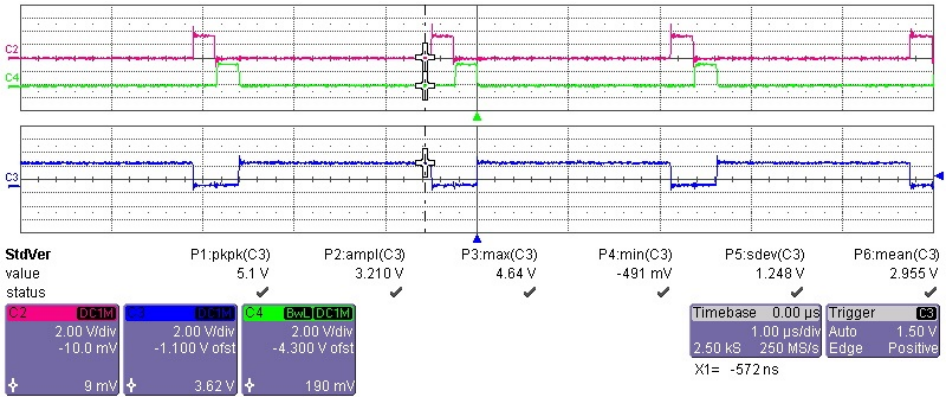

Figure 5.1: Representation of Some Testing Signals on the Digital Oscilloscope

Different attempts were made in order to get relevant traces and in the following subsections it is reported a summary of them.

For each of them it was used the trigger signal from the pin 11 of J4004 on a channel that it was set as trigger source from the "Trigger Setup" option of the digital oscilloscope, with positive edge and level 1.5 V.
For the "Horizontal Setup", it was changed the sample rate, depending on the chosen frequency (500MS/s for 66.50MHz, 250MS/s for 38.00MHz and 25MS/s for 2.08MHz) for the internal oscillator and on the setting of the divisions in order to get an adequate number of samples.
Since the used digital oscilloscope provides two Auto Save options ("Wrap", that overwrites the old files and "Fill", that creates a new file for each trace), it was used the second one and it was selected the format `.dat`.

For the first traces the reset bottom was pressed in order to have a reference and to be sure of the identity of the first trace, corresponding to the plaintext vector "0xBD0F0FFF3F2AF451B4C70000F F873200", then the bottom was released for the following acquisitions.

5.1 Running Frequency 66.50 MHz and Custom Handmade USB-USB/BNC Connector

First of all, the SEcube^{TM} board was programmed, uploading inside the FPGA module our AES-256 core with the internal oscillator set at the maximum frequency allowed (66.50 MHz).
Then, taking as reference the article [50] together with the custom handmade USB-USB/BNC connector, kindly provided by the Laboratory Manager Gianfranco Albis, the instruments were connected as shown in picture 5.2.

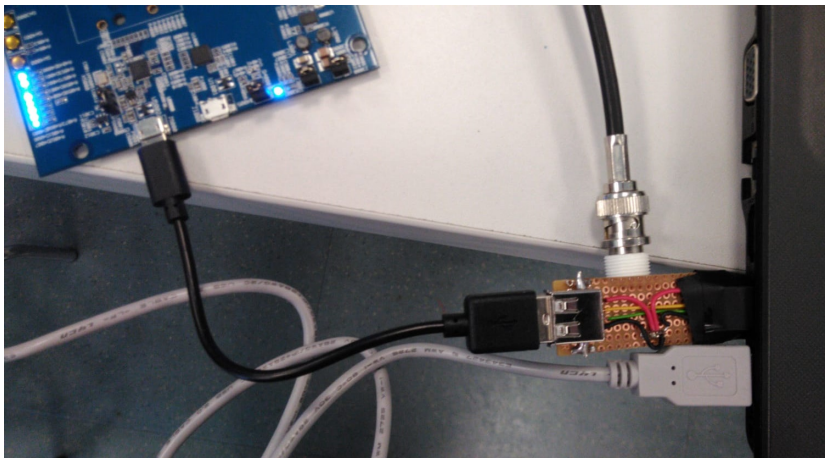

Figure 5.2: Connections using the Custom Handmade USB-USB/BNC Connector

For reference, also a photo (fig.5.3), with some power traces and the trigger signal, is reported.

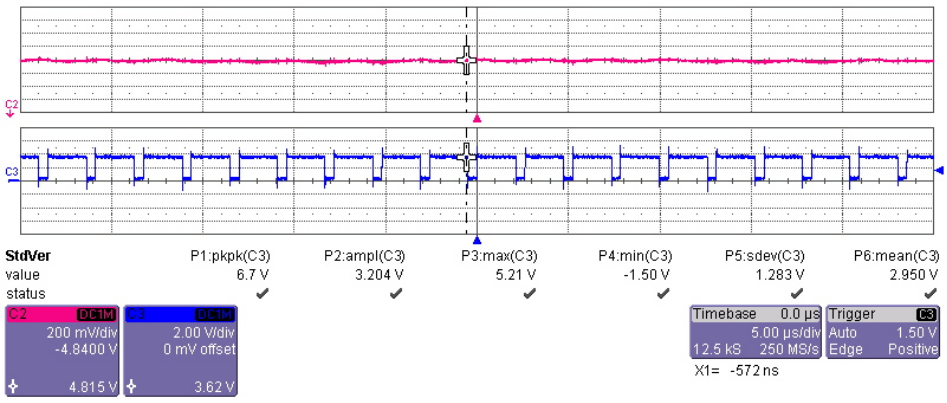

Figure 5.3: Representation of Some Traces and of the Trigger Signal on the Digital Oscilloscope

Finally, some of the traces acquired are presented here, plotted with MATLAB (fig.5.4).

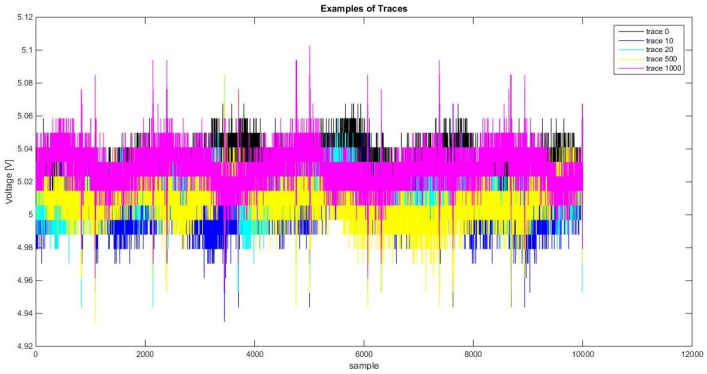

Figure 5.4: Plots of Some Traces on MATLAB (Running Frequency 66.50 MHz and Custom Handmade USB-USB/BNC Connector)

As shown in the last image (fig.5.4) the sets of measurements acquired with this configuration didn't result relevant since there is not a sensible variation with respect the execution of the code (neither with/without the pressure of the reset bottom).

5.2 Running Frequency 38.00 MHz and Current Probe

Since it was provided a Current Probe Tektronix TM502A (50 MHz bandwidth), it was decided to try to get another set of measurements decreasing the working frequency of the internal oscillator in order to respect the bandwidth limit of the current probe. So the VHDL code related to the internal oscillator was changed and it was chosen a lower frequency from the table (fig.4.41), then all the passages for the exportation of the VHDL code and the preparation of the C code described in the previous sections were followed again and the SEcube^{TM} board was re-programmed.
Then it was carefully prepared everything for using the current probe. The digital oscilloscope was connected and set coherently and the probe was with the same division settings (taking as reference the fig.5.5, the output was connected to one channel of the digital oscilloscope, while the input was connected to the probe). It was constantly checked during the measurements if there was the necessity of a degauss re-calibration (that could easily happens due to the electromagnetic interferences and has to be done with the probe removed from the circuit).

Figure 5.5: Current Probe

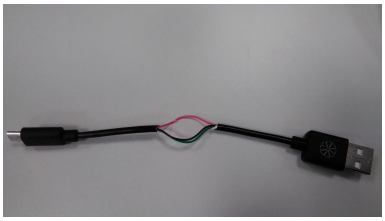

Figure 5.6: USB Cable without part of Covering

Moreover, it was used also a USB cable without a part of its covering (fig.5.7) in order to be able to measure with the probe only the supply cable (indeed, an USB cable is generally consisting of four cables: a black one for the ground, a green one for the negative data, a white one for the positive data

and finally the red one for the +5V power supply).

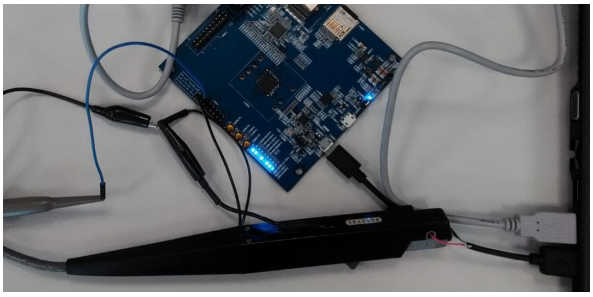

Figure 5.7: Connections using the Current Probe

Below it is reported the plot of some of the acquired traces, but also in this case there were no relevant variations neither with respect the traces taken with the reset active (fig.5.8).

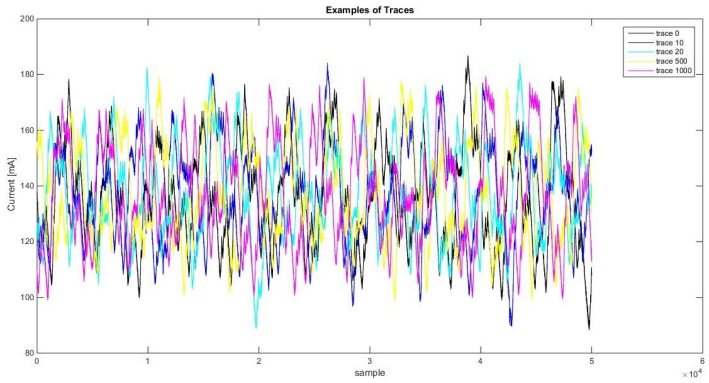

Figure 5.8: Plots of Some Traces on MATLAB (Running Frequency 38.00 MHz and Current Probe)

5.3 Running Frequency 2.08 MHz and Current Probe

Subsequently, the VHDL code was modified again and some traces were obtained using the lower frequency available (fig.4.41).

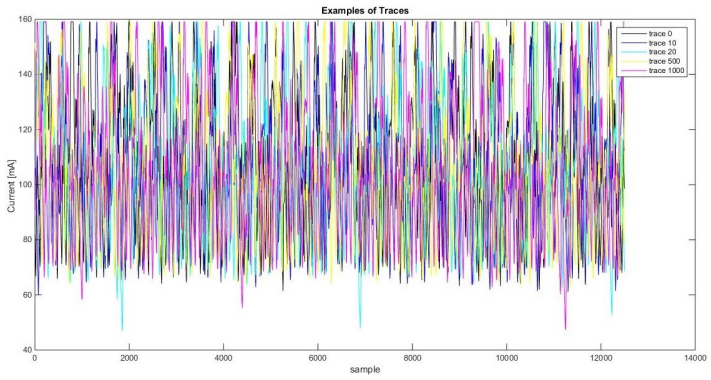

Figure 5.9: Plots of Some Traces on MATLAB (Running Frequency 2.08 MHz and Current Probe)

5.4 Running Frequency 2.08 MHz and Custom Handmade USB-USB/BNC Connector

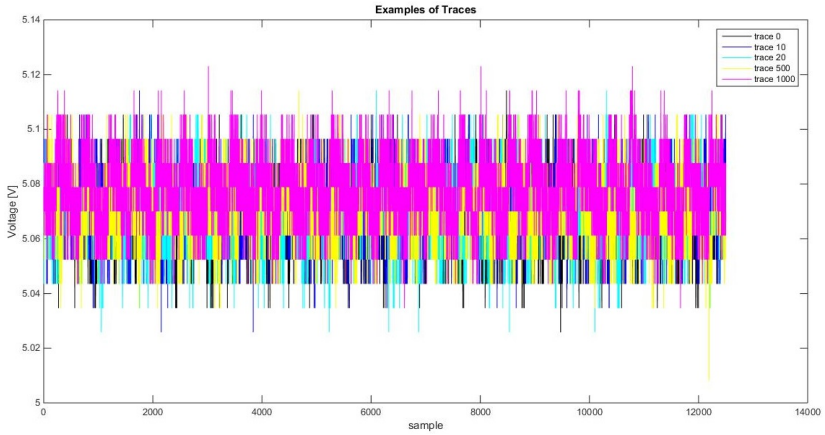

Figure 5.10: Plots of Some Traces on MATLAB (Running Frequency 2.08 MHz and Custom Handmade USB-USB/BNC Connector)

At this point, it began to make measurements from all the available sources of power supply in order to verify if from one of them it would be possible getting some useful traces.
The reference schematics used for finding the possible accesses to the power supply are reported in detail in "SEcube^{TM} Development Kit - Getting Started" guide [11].

5.5 Running Frequency 2.08 MHz and J3000 Power Supply (1.2V)

Here it was used the J3000 access that allows to measure through a standard voltage probe a power supply reference of 1.2 V, that is connected to the FPGA (fig.5.11-5.12).

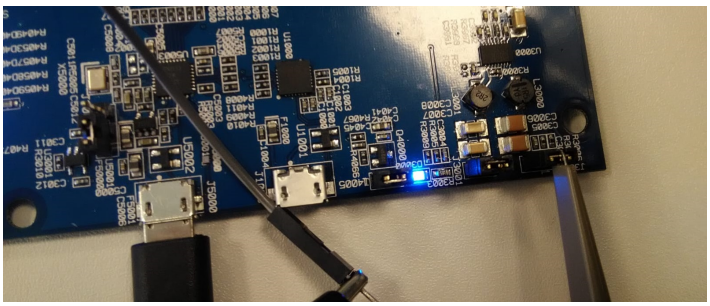

Figure 5.11: Connection to J3000 Power Supply (1.2V)

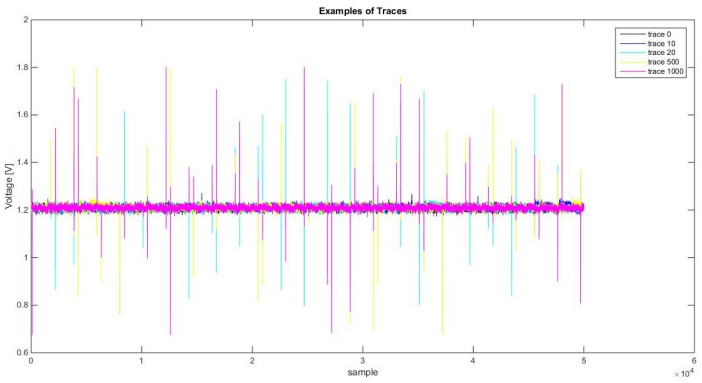

Figure 5.12: Plots of Some Traces on MATLAB (Running Frequency 2.08 MHz and Connection to J3000 Power Supply (1.2V))

5.6 Running Frequency 2.08 MHz and J3001 Power Supply (3.3V)

The figures 5.13 and 5.14 are related to the measurements related to J3001 access to the 3.3 V power supply, the twin of the one just considered.

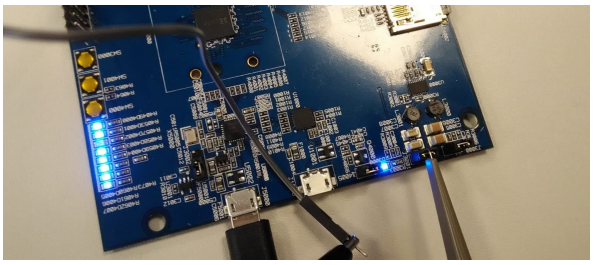

Figure 5.13: Connection to J3001 Power Supply (3.3V)

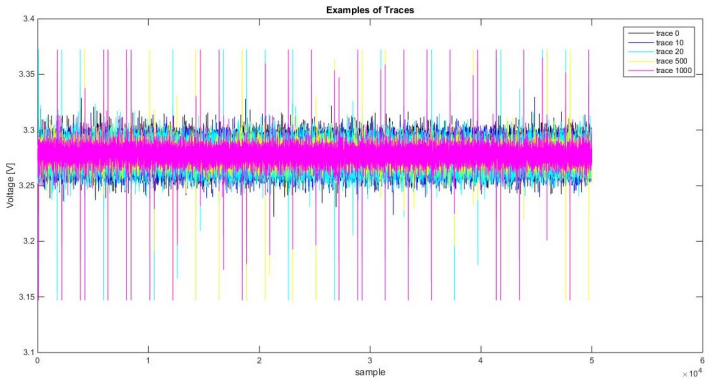

Figure 5.14: Plots of Some Traces on MATLAB (Running Frequency 2.08 MHz and Connection to J3001 Power Supply (3.3V))

5.7 Running Frequency 2.08 MHz, J3001 Power Supply (3.3V) and Current Probe

In this case, it was considered the same power supply source, but acquiring the measurements with two jumpers and the current probe, as shown in figure 5.15. Results shown in the MATLAB plot are shown (fig. 5.16).

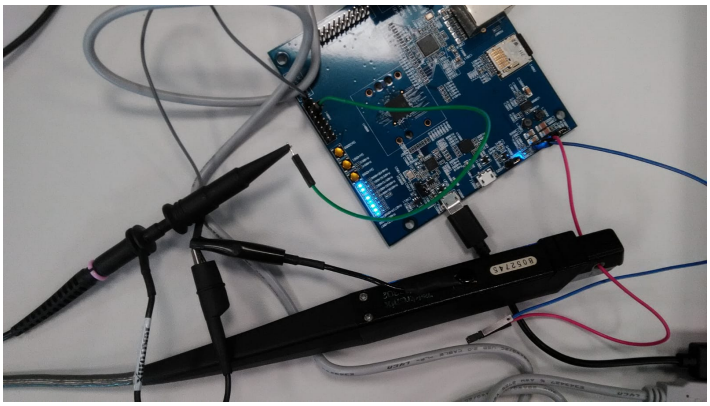

Figure 5.15: Connection to J3001 Power Supply (3.3V) using Current Probe

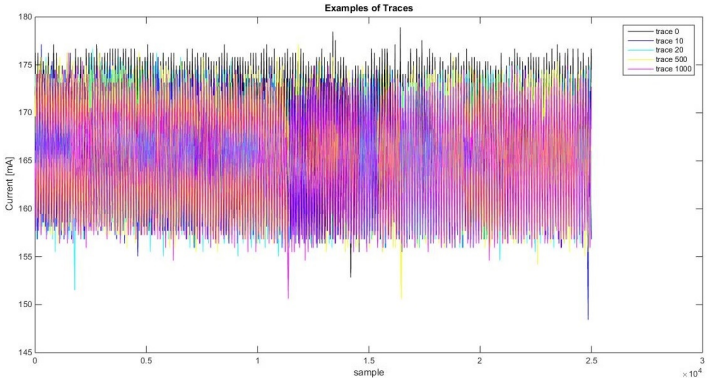

Figure 5.16: Plots of Some Traces on MATLAB (Running Frequency 2.08 MHz and Connection to J3001 Power Supply (3.3V) using Current Probe)

5.8 Running Frequency 2.08 MHz and J4001 Power Supply (3.3V)

Here the pin 1-2 of J4001 were exploited. They are generally used for programming the board through the ST-Link/v2 programmer.

The reference figures (fig.5.17 and fig.5.18) are reported in the following page.

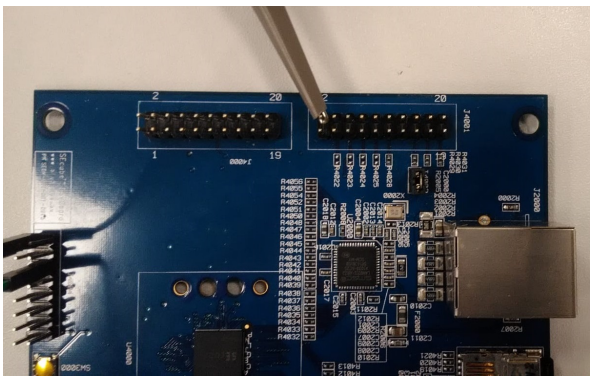

Figure 5.17: Connection to J4001 Power Supply (3.3V)

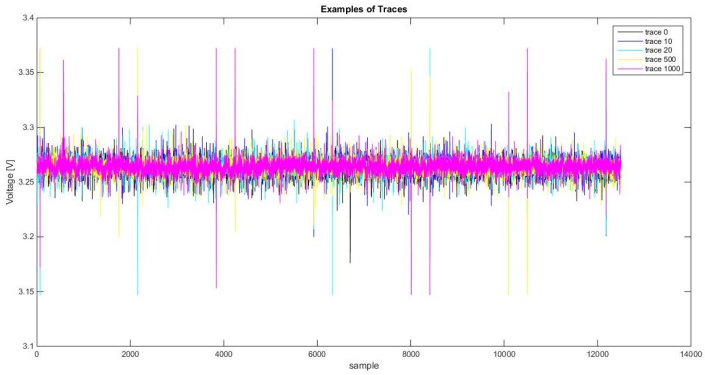

Figure 5.18: Plots of Some Traces on MATLAB (Running Frequency 2.08 MHz and Connection to J4001 Power Supply (3.3V))

5.9 Running Frequency 2.08 MHz and J4002 Power Supply (3.3V)

In this case the pin 4 from J4002 was employed. Due to the difficult in getting a stable signal and the limited accessibility of the pin, the traces are not reported, but also in this case there weren't relevant variations with and without the activation of the reset.

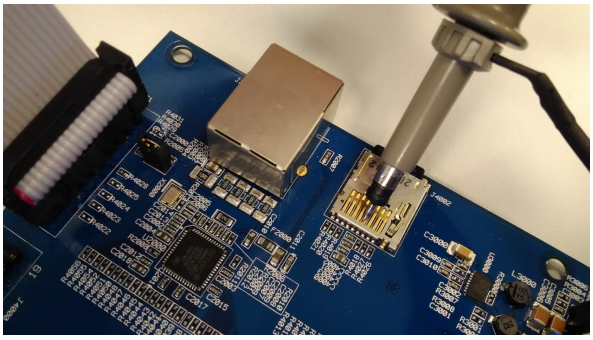

Figure 5.19: Connection to J4002 Power Supply (3.3V)

Another possible tentative would be using an external supply for the board, but, since the SEcube^{TM} board is designed with the focus of maximizing the security, it was impossible doing also this attempt.

5.10 Considerations about the Results

For reference, these data (at 2.08 MHz with the current probe) were compared with the ones of the colleague Giulia Giordano, that worked on a microcontroller STM32F401, so in very different conditions with respect the FPGA inside SEcube^{TM} board: indeed, she charged a C code exploiting a library for the AES cryptography in modality CBC, with a 256-bit-length key, using an external power supply and the same current probe, at the frequency of 8 MHz.
In figure 5.20 are reported a trace acquired in reset condition and other successive two, both for the STM32F401 and for SEcube^{TM}. It can be noticed that the current consumption of the SEcube^{TM} is almost three times the STM32F401 one. Moreover, while the consumption of the STM32F401 is incremented approximately from 5 mA to 30 mA when the encryption is active, the SEcube^{TM} current consumption always remain around 110 mA and the contribute of noise is heavy (roughly a 100 mA interval, so in relative terms around 91%).

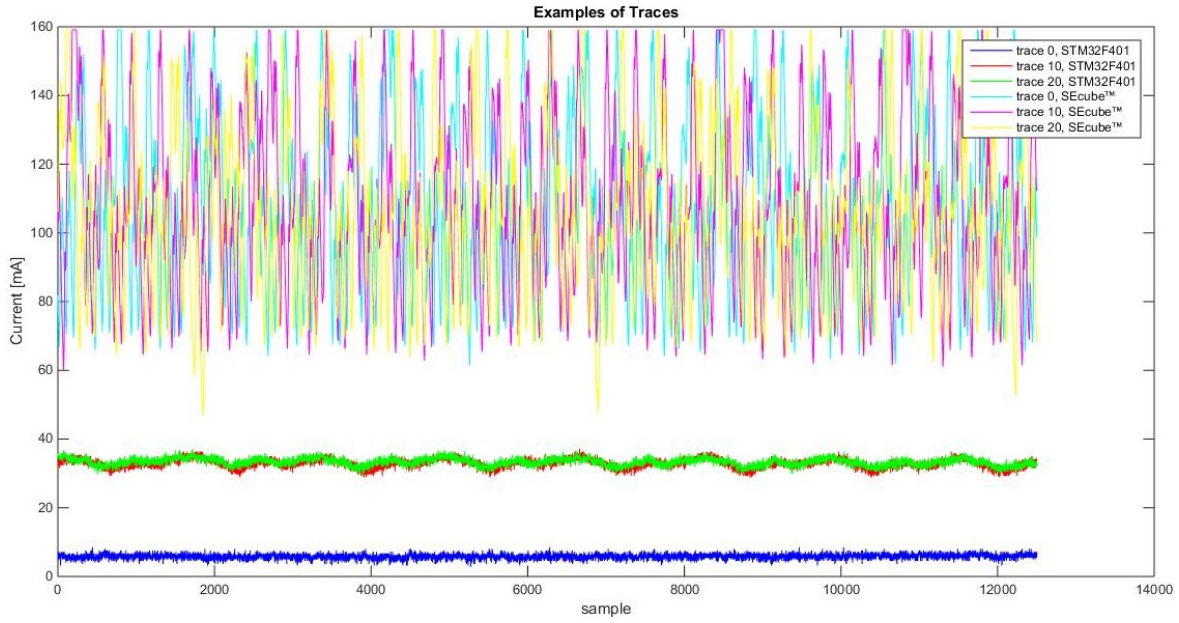

Figure 5.20: Comparison of Traces from STM32F401 and from SEcube^{TM}

To sum up, considering this disastrous contribute of the noise, that neither allow to distinguish when the code is active or not, without using averaging techniques, it will be almost impossible getting some satisfactory results from the DPA tool.

In order to compare the obtained results with the simulated ones, an analysis of the expected power consumption was executed, exploiting Lattice Diamond Power Calculator Tool for each of the different working frequencies considered and the results are reported in the following table:

		66.50 MHz	**38.00 MHz**	**2.08 MHz**
Current by Power Supply	*Static (A)*	0.004655	0.004642	0.004631
	Dynamic (A)	0.005752	0.003689	0.001089
	Total (A)	0.010408	0.008332	0.005720
Power by Power Supply	*Static (W)*	0.005586	0.005571	0.005557
	Dynamic (W)	0.006903	0.004427	0.001307
	Total (W)	0.012489	0.009998	0.006864
Power by Block	*Logic Block (W)*	0.004881	0.004866	0.004853
	Clocks (W)	0.000165	0.000165	0.000164
	I/O (W)	0.000131	0.000130	0.000130
	EFB (W)	0.000004	0.000004	0.000004
	Block RAM (W)	0.000158	0.000158	0.000158
	PLL (W)	0.000001	0.000001	0.000001
Peak Startup (A)		0.0680	0.0680	0.0680
Thermal Profile with Ambient Temperature $25°C$	*Effective* θ_{JA}	11.7°C/W	11.7°C/W	11.7°C/W
	Junction Temperature	25.15°C	25.12°C	25.08°C
	Maximum Safe Ambient	84.6°C	84.63°C	84.67°C

Table 5.1: Summary Table of the Results from "Lattice Diamond Power Calculator Tool" for AES-256 at 66.50 MHz, 38.00 MHz and 2.08 MHz

Even if these results (table 5.1) are just the product of a simulation and they could be a very different from the reality, with some important considerations, they can be extremely interesting and clarifying.
It can be observed that surely the noise is an important component, but it is insufficient for completely justify the obtained results. Indeed, just for reporting an example, the total amount of current absorbed by the power supply in the case of a 2.08 MHz working frequency is assumed 4.631 mA from the Lattice Diamond Power Calculator Tool: even if this corresponds to around the 4.6% of the noise spread and about the 4.2% of the total current absorption (coherent since the FPGA inside SEcube^{TM} chip has really a marginal role in the global power consumption of the board, that incorporates lot of other components and peripherals), it should be possible to observe from the reported plot (fig.5.20) a small shift towards the bottom passing from the not-reset to the reset state, but it didn't appear. Moreover, you have to consider that the design specifications of the USB cables enclose a constant voltage and variable current.
To sum up, this phenomena has a deeper reason that can be found in the presence of a series of decoupling capacitors, that can be considered like a natural defence from DPA attack.
As a consequence, even with some more sophisticated instruments and advanced post-filtering and averaging techniques, it would be extremely hard getting traces sufficiently correlated to the activity of the FPGA.

Another consideration is that there is a decrement of the power consumption reducing the frequency (table 5.1) and this could be consider a disadvantage. However, this decrement isn't too drastic and it is compensated by the theoretic improvement of the quality of the samples (since it is strongly correlated to the clock rate and waveform) generally reported in literature and by the possibility of getting a larger number of samples, given the available maximum sampling rate.
To have also a practical reference, in figure 5.21 is reported a comparison between traces at different frequencies.

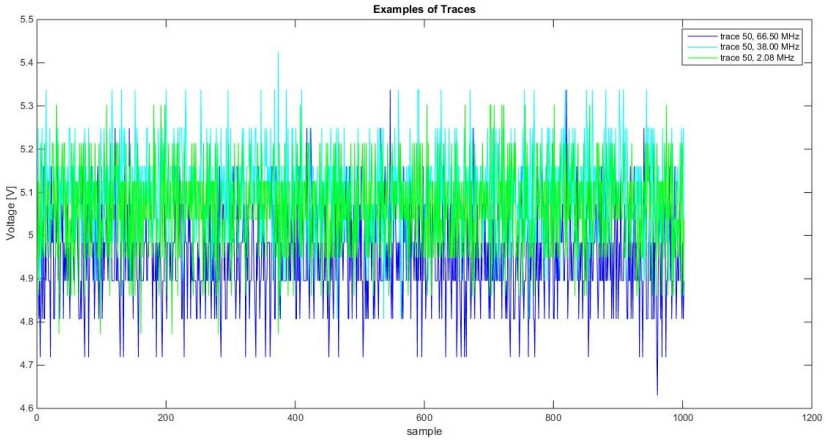

Figure 5.21: Comparison of Traces from SEcube^{TM} with Custom Handmade USB-USB/BNC Connector at Different Frequencies

5.11 General Considerations about the SEcube^{TM} Package and Power Supplies

Another option, not feasible without damaging the board, would be desoldering the SEcube^{TM} BGA package, give the power supply from an external source, the most stable and noiseless as possible, and try to get some useful traces from the available power supply sources, shown in the figure 5.23 below from the "SEcube^{TM} Development Kit - Getting Started" guide [11].
The relevant practical problems in doing an operation like this are another advantage from the security point of view of the SEcube^{TM} board, in addition to all the others extremely important precautions followed during its design, explained more in detail in [11].

Figure 5.22: SEcube^{TM} Package [11]

A1	FPGA_IO_D12	B12	FPGA_IO_D4	E3	CPU_JTAG_TMS	H2	CPU_SDIO_D2	L1	FPGA_IO_CTRL1	N6	FPGA_VCORE
A2	FPGA_IO_CTRL4	B13	FPGA_IO_GP14	E12	CPU_USB_ULPI_D6	H3	CPU_SDIO_CLK	L2	CPU_SC_PWR	N7	CPU_SPI_SSN
A3	FPGA_IO_CTRL2	B14	FPGA_VCORE	E13	CPU_USB_ULPI_D5	H6	VSS	L3	CPU_UART_TX	N8	CPU_USB_ULPI_NXT
A4	FPGA_IO_D9	C1	FPGA_VCCIO0	E14	FPGA_IO_D0	H7	VSS	L12	CPU_USB_ULPI_CLK	N9	CPU_GP1
A5	FPGA_IO_D14	C2	CPU_VCAP2	F1	FPGA_IO_D15	H8	VSS	L13	FPGA_VCORE	N10	CPU_USB_ULPI_STP
A6	FPGA_IO_D7	C3	CPU_VDD	F2	CPU_JTAG_TDI	H9	VSS	L14	FPGA_VCORE	N11	CPU_I2C_SCL
A7	FPGA_IO_GP0	C4	CPU_VDD	F3	CPU_UART_CTS	H12	CPU_VDD	M1	FPGA_IO_CTRL3	N12	CPU_I2C_SDA
A8	FPGA_IO_D6	C5	CPU_UART_RX	F6	VSS	H13	FPGA_IO_GP13	M2	CPU_JTAG_TRST	N13	CPU_USB_ULPI_D0
A9	FPGA_IO_GP6	C6	FPGA_IO_GP3	F7	VSS	H14	FPGA_IO_D1	M3	VSS	N14	VSS
A10	FPGA_IO_GP5	C7	CPU_SDIO_D1	F8	VSS	J1	CPU_USB_ULPI_D7	M4	CPU_VDD	P1	FPGA_VCCIO0
A11	FPGA_IO_GP9	C8	CPU_SDIO_D0	F9	VSS	J2	CPU_VDD	M5	CPU_SPI_CLK	P2	VSS
A12	FPGA_IO_D5	C9	CPU_GP0	F12	CPU_VCAP1	J3	CPU_VDD	M6	CPU_XTAL_IN	P3	FPGA_VCCIO5
A13	FPGA_IO_GP15	C10	FPGA_VCCIO1	F13	CPU_USB_ULPI_D4	J6	VSS	M7	CPU_SPI_MOSI	P4	FPGA_IO_CTRL6
A14	FPGA_VCCIO1	C11	CPU_VDD	F14	FPGA_IO_GP11	J7	VSS	M8	CPU_RSTN	P5	CPU_USB_ULPI_DIR
B1	FPGA_VCCIO1	C12	CPU_VDD	G1	FPGA_VCCIO0	J8	VSS	M9	CPU_ADC	P6	FPGA_VCORE
B2	FPGA_IO_D10	C13	FPGA_VCORE	G2	CPU_JTAG_TCK	J9	VSS	M10	CPU_WKUP	P7	FPGA_VCCIO4
B3	FPGA_IO_CTRL0	C14	FPGA_VCCIO2	G3	CPU_SDIO_D3	J12	CPU_VDD	M11	CPU_VDD	P8	CPU_SPI_MISO
B4	FPGA_IO_D8	D1	FPGA_IO_CTRL13	G6	VSS	J13	FPGA_IO_D2	M12	VSS	P9	FPGA_IO_CTRL8
B5	FPGA_IO_D13	D2	FPGA_VCORE	G7	VSS	J14	FPGA_IO_D3	M13	VSS	P10	FPGA_IO_CTRL9
B6	FPGA_IO_D11	D3	FPGA_VCORE	G8	VSS	K1	FPGA_VCORE	M14	FPGA_VCCIO2	P11	FPGA_IO_CTRL10
B7	FPGA_IO_GP1	D12	CPU_USB_DM	G9	VSS	K2	FPGA_VCORE	N1	SC_VCC	P12	FPGA_IO_CTRL11
B8	FPGA_IO_GP2	D13	CPU_TIMER_PWM	G12	CPU_USB_DP	K3	CPU_JTAG_TDO	N2	FPGA_IO_CTRL5	P13	FPGA_IO_CTRL12
B9	FPGA_IO_GP4	D14	FPGA_IO_GP10	G13	CPU_USB_ULPI_D3	K12	CPU_USB_ULPI_D2	N3	VSS	P14	FPGA_VCCIO3
B10	FPGA_IO_GP8	E1	FPGA_IO_CTRL14	G14	FPGA_IO_GP12	K13	CPU_USB_ULPI_D1	N4	FPGA_IO_CTRL7		
B11	FPGA_IO_GP7	E2	CPU_UART_RTS	H1	CPU_SDIO_CMD	K14	FPGA_VCCIO2	N5	CPU_XTAL_OUT		

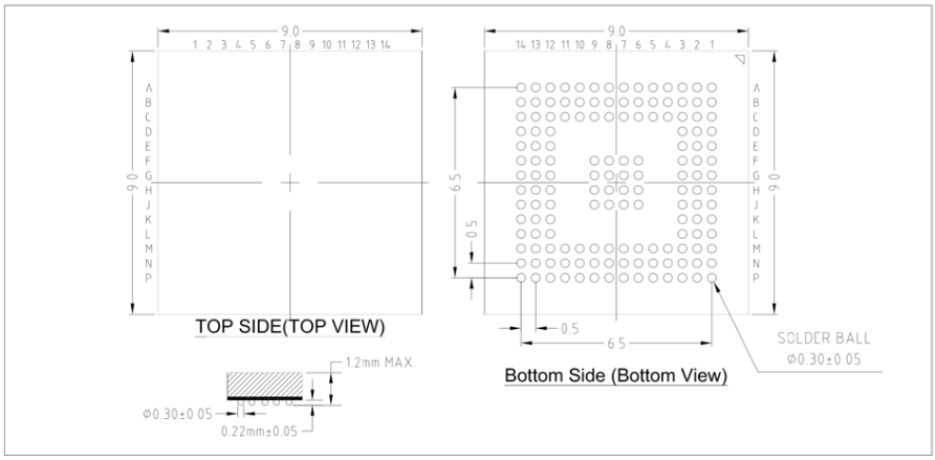

Figure 5.23: SEcube^{TM} Package Scheme [11]

Anyway, also in the case a direct access to the BGA package would be possible in some way, another potential problem in getting power traces strongly sensible to the running core could be the presence of capacitors in front of the accesses to the power supply (fig.5.24) and the same problem is also described in the article "Experiments in Attacking FPGA-Based Embedded Systems using Differential Power Analysis" [20].

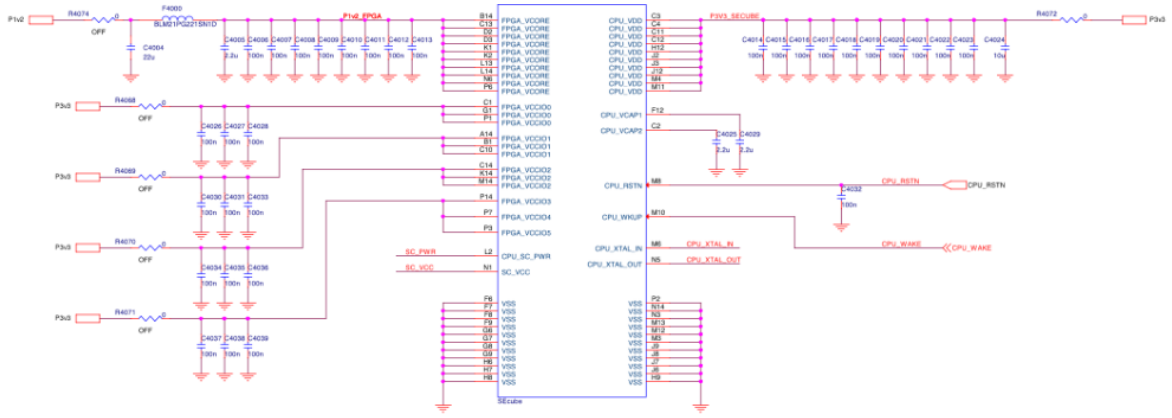

Figure 5.24: SEcube^{TM} Power Supply and Filtering [11]

5.12 Comparison with AES-128

5.12.1 Modifies related to VHDL code

The tools available online for the application of DPA attacks are in general focused only on AES-128 or, in the uncommon case they consider also the AES-256, anyway they easier get better results with AES-128.
For this reason and since the traces are not so satisfactory, it was decided to try also with an an AES-128 implementation.
In order to get it, starting from the previous one (AES-256, CBC mode with `PLAINTEXT_GENERATOR` and internal oscillator at 2.08 MHz), the following main modifies were done:

- Adaptation of the parameters related to the key and to the number of rounds for the AES-128 case in the entities: `aes_enc`, `key_expansion` and `aes_environment`
- In the FSM written inside the entity `PLAINTEXT_GENERATOR`, the previous counter of module 155 was substituted with another one of module 115, since the time for producing an encoded data with the 128-key is 99 clock cycles and then you need 16 clock cycles for the transmission of the ciphertext
- Concerning the entity `ENCRYPTION_MANAGER`, the length of the signal `key` were modified. Now it will be obviously 128 and it was assigned to this the value X"000102030405060708090a0b0c0d0e0f"; moreover it was modified also the KEY STATE, since now it is necessary just having 16 clock cycles for the communication of the key, instead of 32 and then for the management of the key expansion you need only 133 clock cycles instead of 157

After changing the VHDL code, it was verified through a simulation using "Lattice Diamond - Active HDL" that the encryption would be correct, exactly like it was done also for the previous versions of the core (fig.5.25)

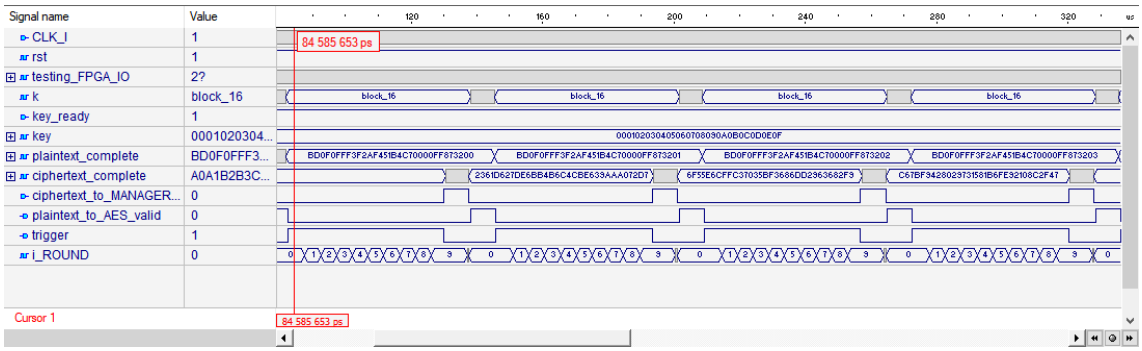

Figure 5.25: Testbench Focus of the first three Encryptions

For reference it was used again the CRYPTOMAThIC Online Tool (fig.4.19), getting the following results:

- Initialization Vector (IV): X"A0A1B2B3C4C5D6D7E8E9F10F01234567"
- First Encryption:
  - Plaintext: X"BD0F0FFF3F2AF451B4C70000FF873200"
  - Ciphertext: X"2361D627DE6BB4B6C4CBE639AAA072D7"

- Second Encryption:
  - Plaintext: X"BD0F0FFF3F2AF451B4C70000FF873201"
  - Ciphertext: X"6F55E6CFFC37035BF3686DD2963682F9"
- Third Encryption:
  - Plaintext: X"BD0F0FFF3F2AF451B4C70000FF873202"
  - Ciphertext: X"C67BF9428029731581B6FE92108C2F47"

The Design Summary from Lattice Diamond is exposed here (fig.5.26):

```
Design Summary
   Number of registers:    914 out of  7869 (12%)
      PFU registers:          914 out of  6864 (13%)
      PIO registers:            0 out of  1005 (0%)
   Number of SLICEs:      1210 out of  3432 (35%)
      SLICEs as Logic/ROM:   1186 out of  3432 (35%)
      SLICEs as RAM:           24 out of  2574 (1%)
      SLICEs as Carry:        115 out of  3432 (3%)
   Number of LUT4s:        2402 out of  6864 (35%)
      Number used as logic LUTs:        2124
      Number used as distributed RAM:     48
      Number used as ripple logic:       230
      Number used as shift registers:      0
   Number of PIO sites used: 15 + 4(JTAG) out of 335 (6%)
   Number of block RAMs:  3 out of 26 (12%)
   Number of GSRs:  1 out of 1 (100%)
```

Figure 5.26: Design Summary, AES-128

Comparing the results above with the ones obtained with AES-256 core in the same conditions, there aren't relevant differences in terms of area and neither in terms of current and power consumption (fig.5.27). Below I reported also the complete table with the values from Lattice Diamond Power Calculator Tool:

		AES-256(2.08MHz)	**AES-128(2.08MHz)**
Current by Power Supply	*Static (A)*	0.004631	0.004625
	Dynamic (A)	0.001089	0.001089
	Total (A)	0.005720	0.005714
Power by Power Supply	*Static (W)*	0.005557	0.005550
	Dynamic (W)	0.001307	0.001307
	Total (W)	0.006864	0.006857
Power by Block	*Logic Block (W)*	0.004853	0.004846
	Clocks (W)	0.000164	0.000164
	I/O (W)	0.000130	0.000130
	EFB (W)	0.000004	0.000004
	Block RAM (W)	0.000158	0.000158
	PLL (W)	0.000001	0.000001
Peak Startup (A)		0.0680	0.0680
Thermal Profile with Ambient Temperature 25°C	*Effective* θ_{JA}	11.7°C/W	11.7°C/W
	Junction Temperature	25.08°C	25.08°C
	Maximum Safe Ambient	84.67°C	84.67°C

Table 5.2: Summary Table of the Results from "Lattice Diamond Power Calculator Tool" for AES-256 (2.08 MHz) and AES-128 (2.08 MHz)

Finally, in order to summarize the situation, it was plotted with MATLAB a graph concerning the current absorbed by the power supply in all the analysed cases (fig.5.27):

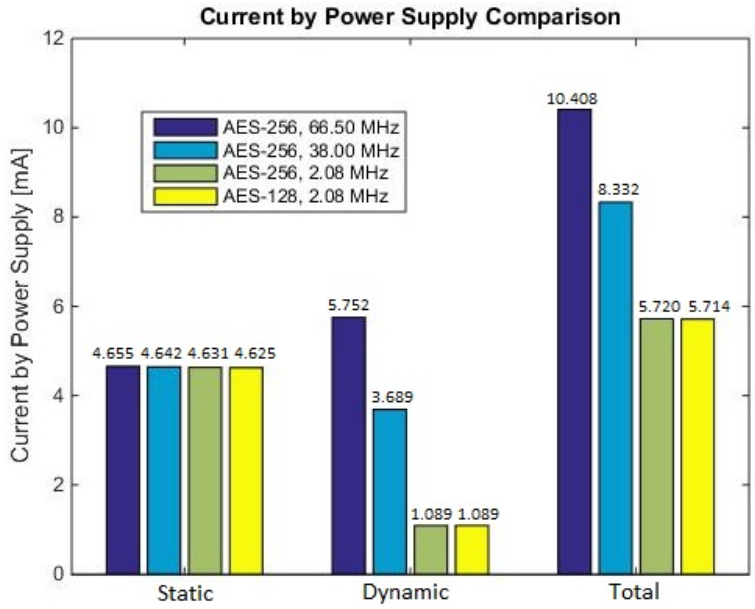

Figure 5.27: Current Consumption Comparison

5.12.2 Measurements in Laboratory

It was decided to try to get a conspicuous set of traces with the higher sampling possible (2.50 GS/s) in order to maximize the possibilities of getting good results from the DPA attack, both with the custom handmade USB-USB/BNC connector and the current probe (fig.5.28-5.29).

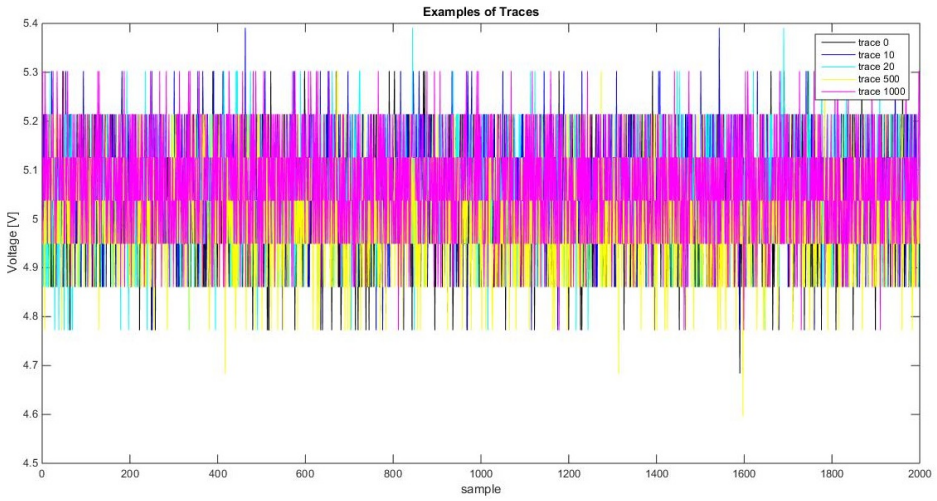

Figure 5.28: Plots of Some Traces on MATLAB (Running Frequency 2.08 MHz and Custom Handmade USB-USB/BNC Connector)

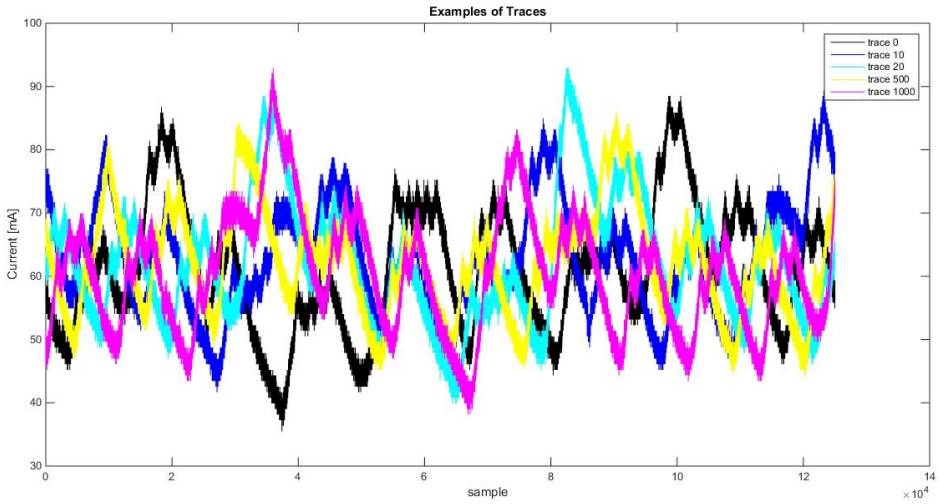

Figure 5.29: Plots of Some Traces on MATLAB (Running Frequency 2.08 MHz and Current Probe)

Apparently in both the cases there aren't relevant improvements since there are still no variations between reset and not-reset state.

A comparison with the AES-256 traces acquired in the same conditions is reported (fig.5.30), but also from this figure you can notice that there is not a substantial improvement.

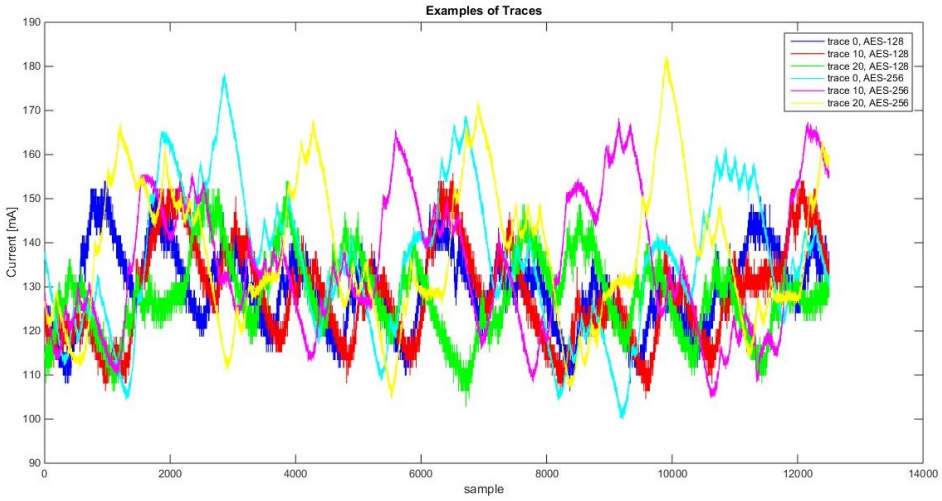

Figure 5.30: Comparison of Traces from AES-256 and from AES-128

CHAPTER 6

DPA Attack and Results

Even if apparently the conditions are not optimal, there is still the possibility that there could be some positive results from the differential power analysis exploiting a large number of samples and with a careful application of some tips.
It was decided to start taking 5120 traces, each of them composed by 1250022 samples in the conditions of AES-128 core used, working frequency of 2.08 MHz, the maximum sample rate available (2.50 GS/s) and employment of the current probe and of the handmade USB-USB/BNC connector: in this way, it was tried to maximize as much as possible the efficacy of the traces.
As a consequence 20 cycles of encryption had to be managed, resetting the system every 256 traces (in order to do, in the next step, a correct average with equal plaintexts, since using CBC mode there is a constant transformation of the effettive plaintext that will be analysed by the tool) and a total of $5120 \cdot 125002 = 640010240$ samples, that is around four times the number of samples they acquired in the article "Side-channel analysis of SEcube^{TM} platform" ($15872 \cdot 10000 = 158720000$) [50].
First of all, the traces were reorganized: they were thrown away, for each of the 20 set, the first traces taken in reset mode (even if it cannot be noticed from the plots) and also the last ones that overcame the 256-th. At this point, since the number of samples for each trace was extremely elevate and it was problematic to manage files too heavy, all the remaining 5120 traces weren't concatenated together, but they were organized in 20 different folders, each one containing a complete cycle of encryption (from the encryption of the plaintext 0xBD0F0FFF3F2AF451B4C70000FF8732*00* to the one of 0xBD0F0FFF3F2AF451B4C70000FF8732*FF*) and then the content of each folder was concatenated, exploiting the following command by Windows Command Prompt:

```
> copy *.dat merged_file.dat
```

Note that using this procedure, you'll have to cancel at the end of `merged_file.dat`, the ending string SUB in order to import it correctly.
Consequently, having obtained 20 sets of traces, composed by 32000512 samples and it was decided to average them as described in "Chapter 2 - Differential Power Analysis (DPA) Key Concepts", in order to improve the results of the analysis. For doing this, you can use a simple code in MATLAB or in Python that simply adds all the sets in a vector an then divides the values accumulated in the vector by the number of the sets and saves this in a new file.
In order to facilitate the averaging operations (due to the large amount of data), the 20 sets were grouped in 4 parts, each composed by 5 sets, then the total average was done.
In the following image, the first 5 sets of traces and their average are plotted with MATLAB (fig.6.1-6.2).

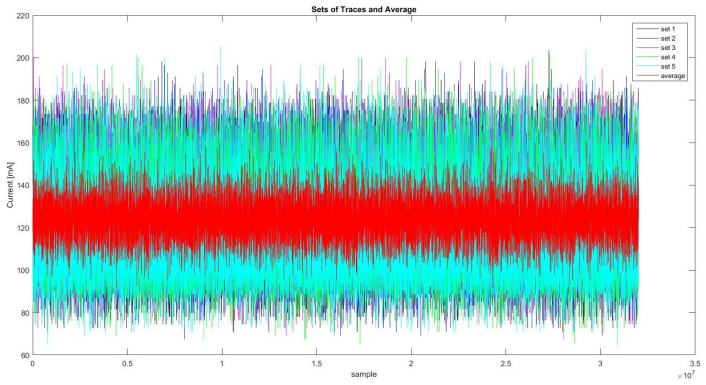

Figure 6.1: First Five Set of Traces and Average (Current Probe)

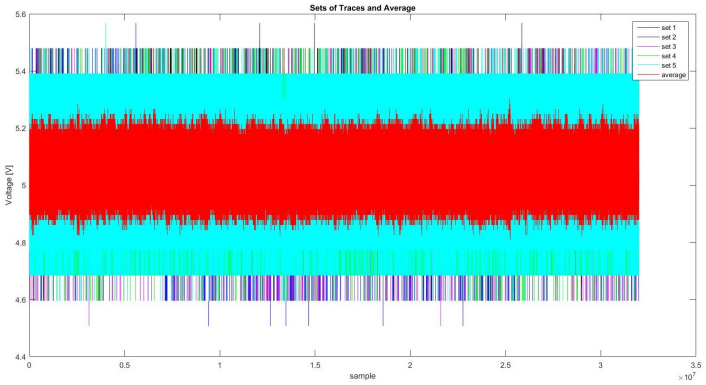

Figure 6.2: First Five Set of Traces and Average (Custom Handmade USB-USB/BNC Connector)

At this point, in order to be able to use the DPA tool, two files containing all the 256 plaintexts and the corresponding ciphertexts had to be generated. For doing this, a simple C program was used, including the AES library with a for cycle managing all the plaintexts and the two files were saved in `.dat` format.
Note that, working in the CBC mode (scheme in figure 3.9), the DPA tool doesn't consider this modality, but it operates just in ECB mode. To use it anyway, you can't save directly the original plaintext, but a dummy version of the plaintext xored with the ciphertext of the previous encryption (or with the IV in the the case of the first plaintext).
Later it was prepared the setup for the chosen DPA tool, called `Jlsca`, from `https://github.com/Riscure/Jlsca`.
`Jlsca` is a toolbox written in Julia, that is a programming language similar to Python, specifically designed for high-performance numerical analysis and, in order to use it, it is necessary to do some preliminary operations.
First of all, you have to download a stable release of Julia (I used Julia 1.0) from `https://julialang.org/downloads/`. Then you can open the Julia Command-Line REPL (Read-Eval-Print-Loop) and here you have to add `IJulia` (`https://github.com/JuliaLang/IJulia.jl`), that is a Julia-language backend combined with the Jupyter Notebook interactive environment, together with others necessary packages. To sum up, I reported here all the commands required:

```
> using Pkg
> Pkg.add("IJulia")
> Pkg.add("Conda")
```

```
> using Conda
> rm(Conda.ROOTENV, recursive=true)
> ENV["CONDA_JL_VERSION"]="2"
> using Pkg
> Pkg.build("IJulia")
> Pkg.clone("https://github.com/Riscure/Jlsca")
> Pkg.add("Plots")
> ]add PyCall
> add PyPlot
```

After this, every time you want to run the tool, you just have to open Jupyter Notebook using the commands:

```
> using IJulia
> notebook()
```

For the execution of the DPA, you can start from the available examples on `https://github.com/ikizhvatov/jlsca-tutorials`, in particular the ones referred to similar conditions, converting the files from the format `.dat` to `.npy` just modifying the converter provided together with the examples of the tutorial.
Before launching the tool, it was calculated on which samples to focus for analysing only the first round of encryption. As you can see from fig.6.3, it takes 11 clock cycles, so the correlated samples will be:

$$\left\lceil \frac{\text{sample rate}}{\text{working frequency}} \cdot \text{first round clk cycles} \right\rceil = \left\lceil \frac{2.50\ \text{GS/s}}{2.08\ \text{MHz}} \cdot 11 \right\rceil = 13222$$

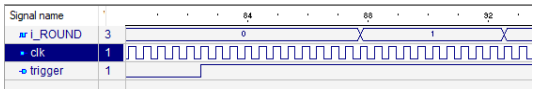

Figure 6.3: Focus on First Round of AES-128

For the last round, a similar approach can be adopted, but using the figure 6.4:

$$\left\lceil \frac{\text{sample rate}}{\text{working frequency}} \cdot \text{last round clk cycles} \right\rceil = \left\lceil \frac{2.50\ \text{GS/s}}{2.08\ \text{MHz}} \cdot 8 \right\rceil = 9616$$

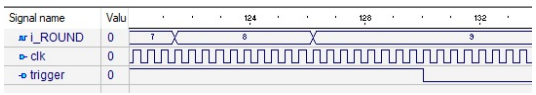

Figure 6.4: Focus on Last Round of AES-128

In the following subsections some guidelines and results for the specific cases are exposed.

6.0.1 Piece of SCAke

This example is based on a correlation power analysis attack (CPA) on an unprotected AES-128 that they charged on a 8-bit AVR microcontroller.
The first step in the code is importing the libraries and converting the traces from the format `.npy` to `.trs`. Then there is an exploration of traces phase, in which you open the converted file and plot

some of the traces with the desired zoom. Here they are reported as reference the results related to the case of AES-128 in ECB mode, both for the measurements acquired with the current probe and the custom handmade USB-USB/BNC connector (fig.6.5-6.6).

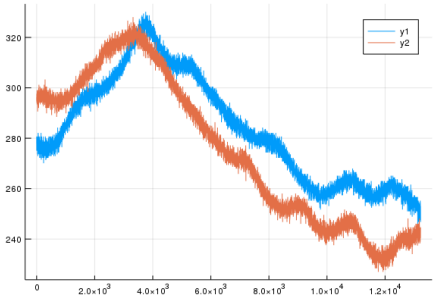

Figure 6.5: First Round Focus (Current Probe)

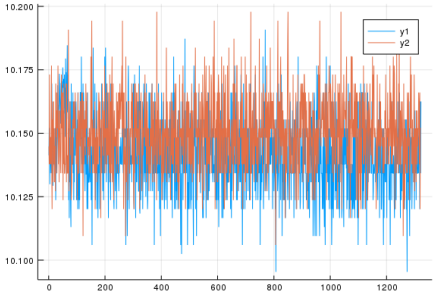

Figure 6.6: First Round Focus (Custom Handmade USB-USB/BNC Connector)

In this case, they assumed there is no misalignment of the traces, so the alignment operations are skipped and the CPA on first round with the Hamming weight leakage model is directly applied ad with the acquired traces you get as proposed key the values:

- Current Probe: "0x000000000000000000000000000000b5"
- USB-USB/BNC Connector: "0x000000000000000000000000000007c"

You can notice that since in this case the plaintext is directly given to the algorithm and there is only a variation on the last byte, applying the algorithm on the first round, only the last byte of the key is computed.
Finally, there is a part of code for the verification of the proposed key using a plaintext/ciphertext pair from the trace set and in our case a negative result was obtained.

6.0.2 Still Not SCAry

The code is more or less the same of "Piece of SCAke", but there is the additional passage of the alignment.

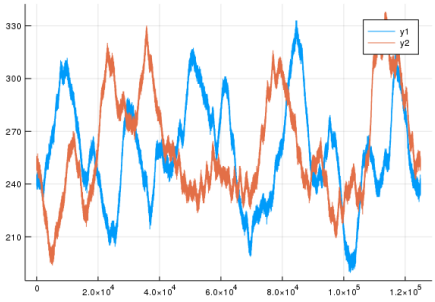

Figure 6.7: Before Alignment (Current Probe)

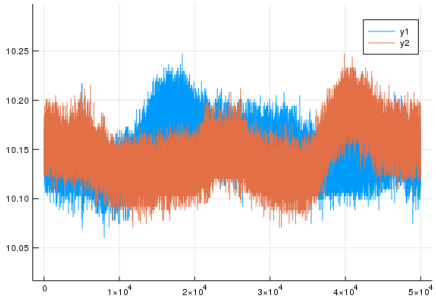

Figure 6.8: Before Alignment (Custom Handmade USB-USB/BNC Connector)

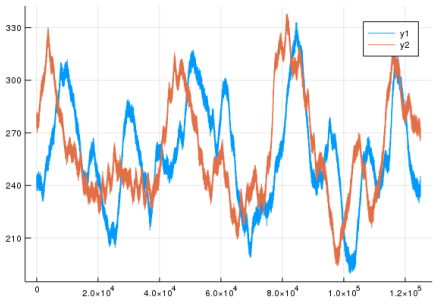

Figure 6.9: After Alignment (Current Probe)

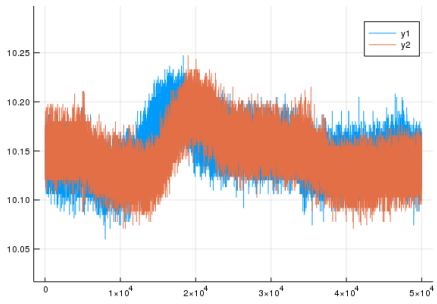

Figure 6.10: After Alignment (Custom Handmade USB-USB/BNC Connector)

Furthermore, despite of considering the first round, in this case they consider the last round of the AES-128 and instead of the CPA (Correlated Power Analysis), they use a LRA (Linear Regression Analysis).
The obtained keys are:

- Current Probe: "0x1311d24b88ed47ade3248faeaf591fba"
- USB-USB/BNC Connector: "0xb8e9af66c73280566e998f54ca0441b8"

6.0.3 SCAlate

Here they use again CPA, like in "Piece of SCAke", but adding the alignment passage. The code is divided in two parts: in the first part, the conditional averaging (like the standard averaging with the difference that not all trials from the trial list will contribute to the average) is applied, while, in the second part, they exploit the incremental correlation technique (slower, but more effective).

Conditional Averaging:

- Current Probe: "0xb3f290978c5c8418034f8979bf7e2202"
- USB-USB/BNC Connector: "0x9f12f62e5f830bc91e14a35d40004a89"

Incremental Correlation:

- Current Probe: "0x75fd10a19b67342f328f76648b85cb82"
- USB-USB/BNC Connector: "0xaf3c9b25e37a136d0e729f9b228b6b90"

6.1 Summary of DPA Results

The four analysis were repeated for all the following cases: AES-256 at 2.08 MHz, both with the USB-USB/BNC connector and the current probe, in CBC and ECB block cipher modes, AES-128 in CBC mode at 2.08 MHz with both the type of measurement (using 5 sets and 20 sets of measurements), AES-128 in ECB mode at 66.50 MHz only with USB-USB/BNC connector, since this frequency is out of the bandwidth of the current probe and finally AES-128 using both USB-USB/BNC connector and current probe at 38.00 MHz and 2.08 MHz (using 5 sets and 20 sets of measurements).
In the following tables the obtained results for AES-128 are illustrated (tables6.1-6.2-6.3).

		Key	**Number of Hit Bits**	**Percentage of Hit Bits**
AES-128	*Piece of SCAke*	0xc69f1e06887875db7ccc9cb8e131cfd6	60	46.87%
CBC	*Still Not SCAry*	0x4d9215dcb9c0d3eeeec27fb8a187bf88	53	41.41%
2.08 MHz	*SCAlate - CA*	0xc4603d51bf52bcc82c363c3a6f65b760	58	45.31%
BNC	*SCAlate - IC*	0x0ee76ecada4c631c632e6d549fb292b2	54	42.19%
AES-128	*Piece of SCAke*	0x24fa6771248b5e79fe21b1ce1ceaa1fd	61	47.66%
ECB	*Still Not SCAry*	0x485dfd024e6e157adf1219a29e2044b9	64	50.00%
66.50 MHz	*SCAlate - CA*	0x182fe5db9bae73bc6cfa2059621ed97c	56	43.75%
BNC	SCAlate - IC	0x4e5fe152a77a94dfec764c6b6b2978db	61	47.66%
AES-128	*Piece of SCAke*	0x0000000000000000000000000000001e	62	48.44%
ECB	*Still Not SCAry*	0x2109f525c13fddbe2956c5b25edb00d9	61	47.66%
38.00 MHz	*SCAlate - CA*	0x359c31d911f95aa14f21a26822bcce2b	68	53.12%
BNC	*SCAlate - IC*	0x92ffc91b7e71677e97f796cb57586950	55	42.97%
AES-128	*Piece of SCAke*	0x000000000000000000000000000000b9	59	46.09%
ECB	*Still Not SCAry*	0x5b3cc3006ed27075e416363d47356566	60	46.87%
2.08 MHz	*SCAlate - CA*	0x58af3c2a1a2204a552a058d565f6baba	63	49.22%
BNC	*SCAlate - IC*	0xfc14250075625f6d4fdb3a64a8647bf4	61	47.66%

Table 6.1: Summary Table of AES-128 Results (BNC, 5 sets)

		Key	**Number of Hit Bits**	**Percentage of Hit Bits**
AES-128	*Piece of SCAke*	0x23a0cfb2c0a53a79b34bfdfceaa4501c	59	46.09%
CBC	*Still Not SCAry*	0x6047816bc90e5b59e50ff513b2de6d1f	65	50.78%
2.08 MHz	*SCAlate - CA*	0x6ecd59cd45d9c91a624a95ff67998bed	59	46.09%
Current Probe	*SCAlate - IC*	0x3123c0e225ee1ce91f2a6f71fa903912	63	49.22%
AES-128	*Piece of SCAke*	0x000000000000000000000000000000ec	59	46.09%
ECB	*Still Not SCAry*	0x400849a5969c7c0d1e8a5388fe1eee55	75	58.59%
38.00 MHz	*SCAlate - CA*	0xcc67252d0d16b2a5835d51a3378be462	67	52.34%
Current Probe	*SCAlate - IC*	0x13b81c1b6cfce85f93e658d87a296903	62	48.44%
AES-128	*Piece of SCAke*	0x000000000000000000000000000000b5	59	46.09%
ECB	*Still Not SCAry*	0x1311d24b88ed47ade3248faeaf591fba	75	58.59%
2.08 MHz	*SCAlate - CA*	0xb3f290978c5c8418034f8979bf7e2202	69	53.91%
Current Probe	*SCAlate - IC*	0x75fd10a19b67342f328f76648b85cb82	66	51.56%

Table 6.2: Summary Table of AES-128 Results (Current Probe, 5 sets)

		Key	Number of Hit Bits	Percentage of Hit Bits
AES-128	*Piece of SCAke*	0xc232517b4bc421247cef2dda9446e1a6	63	49.22%
CBC	*Still Not SCAry*	0xbe39613758a14f699a5ddc23c65240b8	64	50.00%
2.08 MHz	*SCAlate - CA*	0x38238a942aa587e4bdaf6effa425a8b0	70	54.69%
BNC	*SCAlate - IC*	0xceaabdeade185710aba04e62d52e1b6f	63	49.22%
AES-128	*Piece of SCAke*	0xe7f763c7b01144fd4da1f963e5544bda	64	50.00%
CBC	*Still Not SCAry*	0x0497a279c4b03308d9cd3107f8f61fe6	68	53.12%
2.08 MHz	*SCAlate - CA*	0xa4d69c8145c56402b42152ec78e8f846	69	53.91%
Current Probe	*SCAlate - IC*	0x64c3769aff5a31d8624c678dc278d2e8	53	41.41%
AES-128	*Piece of SCAke*	0x0000000000000000000000000000007c	59	46.09%
ECB	*Still Not SCAry*	0xb8e9af66c73280566e998f54ca0441b8	64	50.00%
2.08 MHz	*SCAlate - CA*	0x9f12f62e5f830bc91e14a35d40004a89	68	53.12%
BNC	*SCAlate - IC*	0xaf3c9b25e37a136d0e729f9b228b6b90	59	46.09%
AES-128	*Piece of SCAke*	0x000000000000000000000000000000f7	59	46.09%
ECB	*Still Not SCAry*	0x7f482f91849946da4fc5af075bc363b7	66	51.56%
2.08 MHz	*SCAlate - CA*	0xc081bac396ed75ef6162b64ff4ae4f5b	73	57.03%
Current Probe	*SCAlate - IC*	0x2ee501deb3a4145eda829d61c7a13dfe	62	48.44%

Table 6.3: Summary Table of AES-128 Results (Current Probe, 20 sets)

In the case of "Piece of SCAke" applied to AES in ECB mode, the tool calculates only the last byte of the key and leaves all the others byte at zero. Since the used keys are the standard ones ("0x000102030405060708090a0b0c0d0e0f" for AES-128 and "0x000102030405060708090a0b0c0d0e0f10 111213141516171819 1a1b1c1d1e1f" for AES-256) and the percentage of zeros in this particular case is extremely hight, in order to make this result comparable with the other ones it was assumed that the not computed bytes of the key had a percentage of success of the 50% and the number and the percentage of hit bits were normalized on this assumption. For the same reason, in the comparisons between AES-256 and AES-128, it wasn't taken in consideration the number of hit bits, but only the percentage with respect the total length of the key.
For better analysing the results, a MATLAB code was written, in which the keys were saved, converting them in binary vectors exploiting the function `hexToBinaryVector(hexNumber,numberOfBits,bitOrder)`. In figure 6.11, the plot shows the better hypothesis of key obtained (AES-128, ECB mode, current probe, 2.08 MHz using 5 sets of encryptions), compared with the true key.

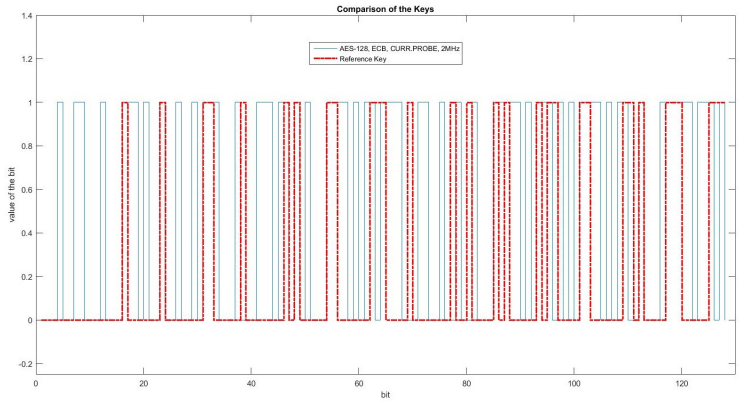

Figure 6.11: Comparison of the Hypothesis of Key (AES-128, ECB mode, current probe, 2.08 MHz, 5 sets of encryptions) with respect the Reference Key

Then, since the meaning of the results is not so easy to appreciate from this representation, it was created a new vector for each key in which it was put the difference between the key guess and the official key. An example (again using the key hypothesis that resulted nearer to the true one) is illustrated in figure 6.12.

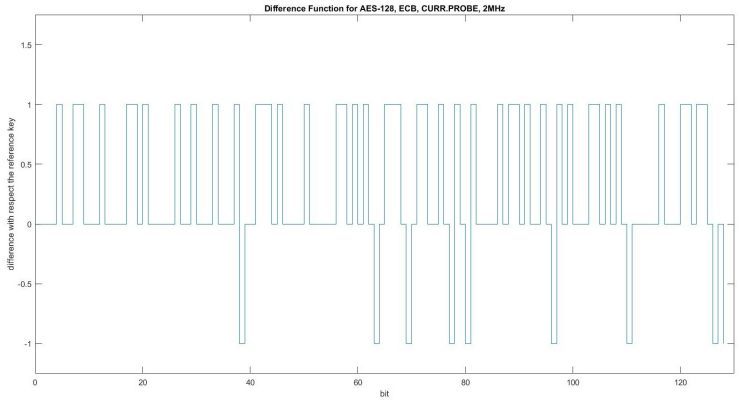

Figure 6.12: Difference Function for AES-128, ECB mode, current probe, 2.08 MHz, 5 sets of encryptions, with respect the Reference Key

Finally, the `sum` function was used in order to get, for each hypothesis, the number of bit corresponding to the real key (`number_of_zeros_in_difference_function = sum(difference_function(:)==0)`) and, after the computation of the percentage with respect the length of the key, the results were plotted using `barh` function (fig.6.13-6.14-6.15-6.18).

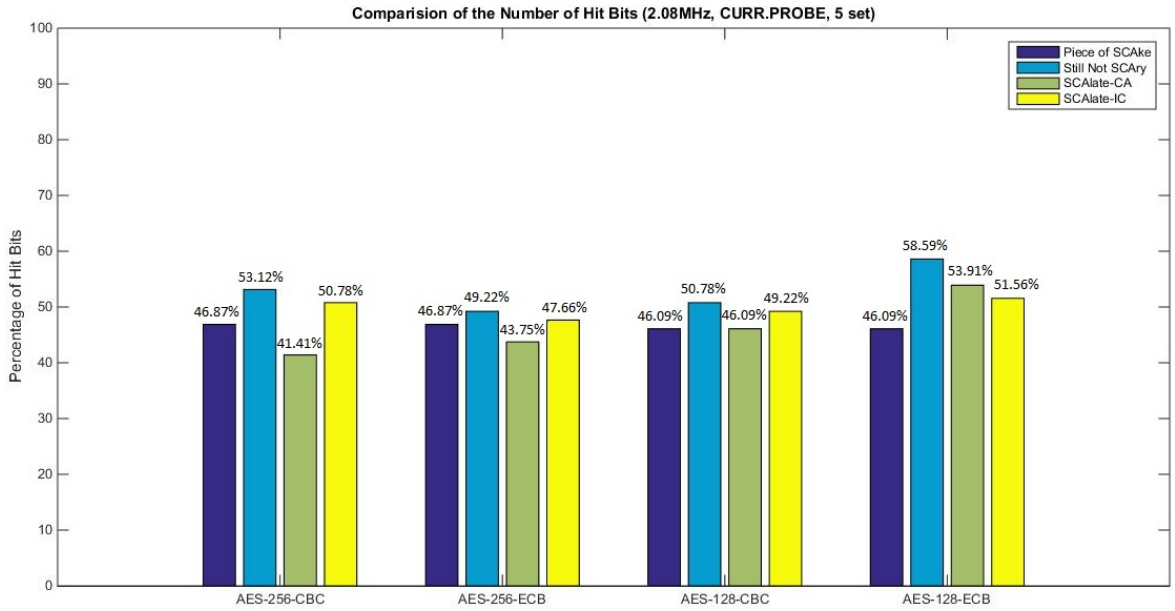

Figure 6.13: Comparisons between AES-256 and AES-128, both ECB and CBC Block Cipher Modes, using the Current Probe, at 2.08 MHz (5 Sets of Encryptions)

From the plot in figure 6.13, you can notice that, as expected, there is a good increment of the percentage of hit bits passing from AES-128 (CBC) to AES-128 (ECB); on the contrary, the data related to AES-256 are not so compliant to the theory since there is a light decrement passing from

AES-256 (CBC) to AES-256 (ECB), except for "SCAlate - CA". Finally, in general there is an higher number of hit bits for the AES-128 with respect the AES-256, coherently with the expectations.

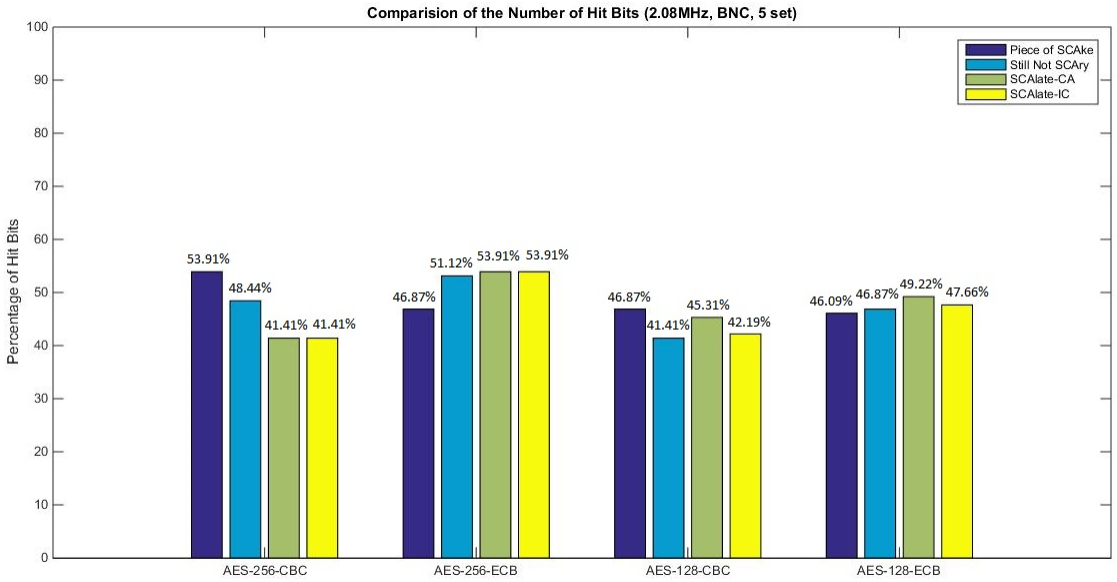

Figure 6.14: Comparisons between AES-256 and AES-128, both ECB and CBC Block Cipher Modes, using the Custom Handmade USB-USB/BNC Connector, at 2.08 MHz (5 Sets of Encryptions)

With respect the previous figure (fig.6.13), in figure 6.14 there is a deterioration of all the results and also an increment of the number of inconsistencies with respect the theory. This is justified by the exploitation of the custom handmade USB-USB/BNC connector instead of the current probe, that allows to get better acquisitions.

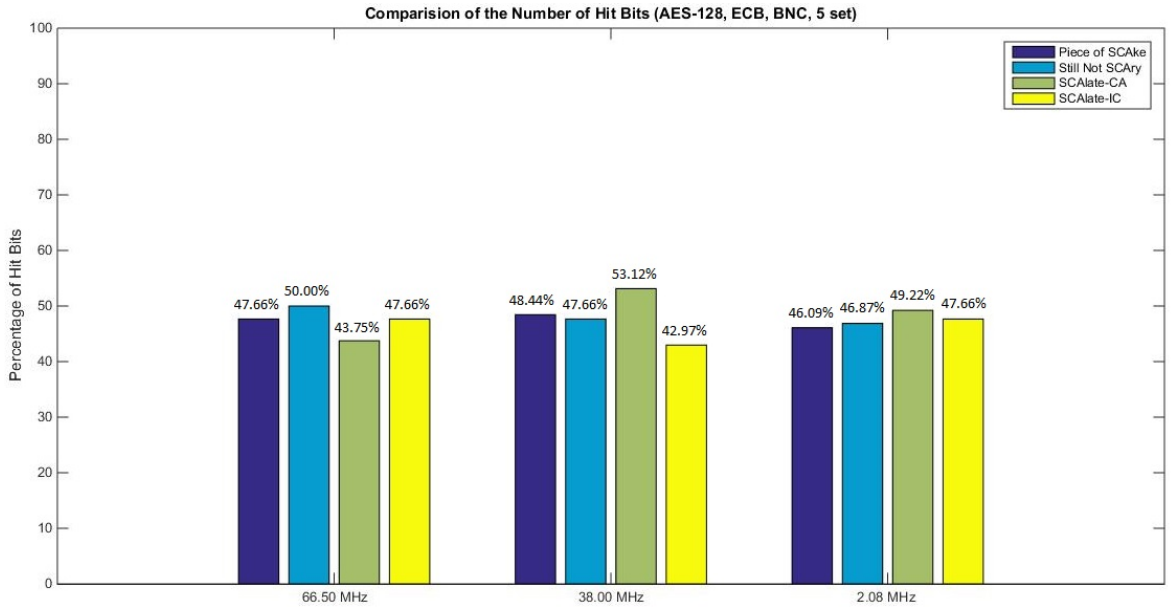

Figure 6.15: Comparisons of AES-128 in ECB Block Cipher Mode, using the Custom Handmade USB-USB/BNC Connector, at different frequencies: 66.50 MHz, 38.00 MHz and 2.08 MHz (5 Sets of Encryptions)

In the figure above (fig.6.15), the results obtained with the custom handmade USB-USB/BNC connector for AES-128 (ECB mode) are compared at three different working frequencies: 66.50 MHz, 38.00 MHz and 2.08 MHz. An improvement of the results is expected decrementing the working frequencies, since, fixed the maximum sampling rate of the digital oscilloscope, you get an higher number of samples and you have a clock without overshoot and with sharper edges, so obviously also a better defined trigger signal (fig.6.16-6.17).
The results are not compliant due to the exploitation of the custom handmade USB-USB/BNC connector, that generally provides lower quality results, and due to the proximity to the 50%, that makes them not comparable since too sensible to the random noise, despite the averaging technique.

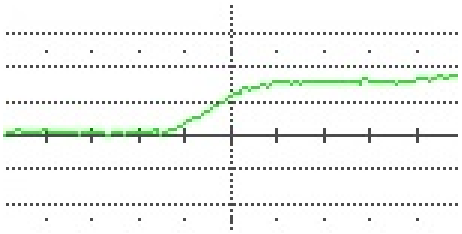

Figure 6.16: Focus on Trigger Signal at 66.50 MHz Working Frequency

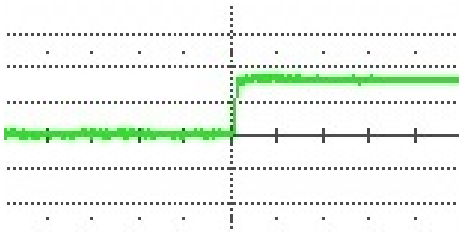

Figure 6.17: Focus on Trigger Signal at 2.08 MHz Working Frequency

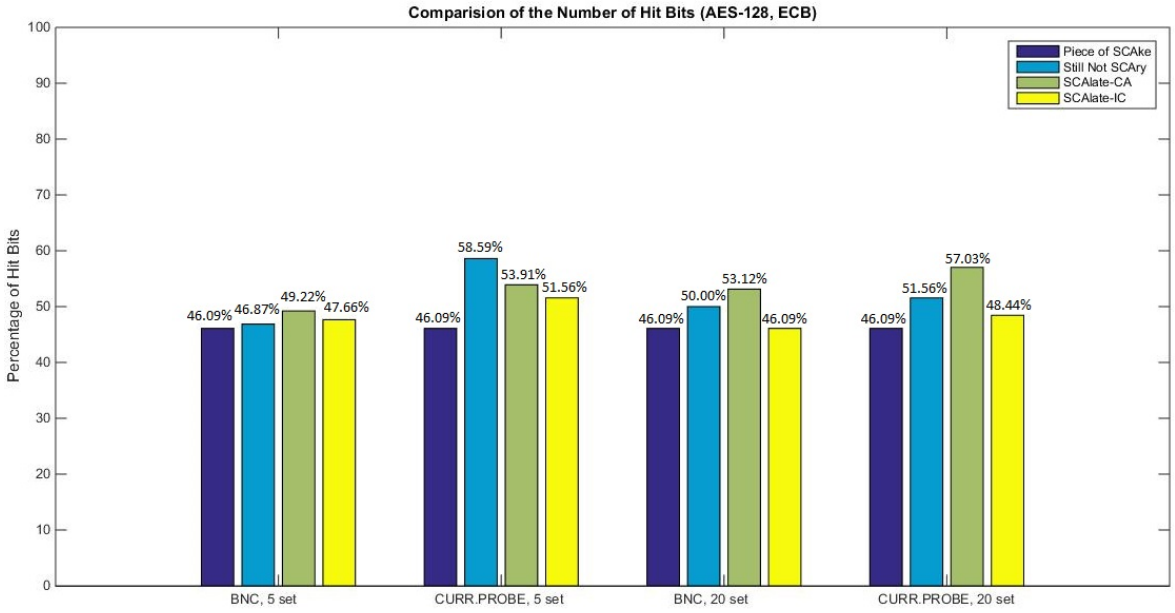

Figure 6.18: Comparisons of AES-128 in ECB Block Cipher Mode, using both the Custom Handmade USB-USB/BNC Connector and the Current Probe and acquiring 5 or 20 Sets of Encryptions)

In this last picture (fig.6.18) there is a comparison between the obtained results using only 5 sets of measurements and incrementing the number of sets up to 20. Some improvements are present, but they aren't so relevant and with some of the used DPA models the effect is the opposite and the higher number of hit bits is obtained using the current probe and taking only 5 sets of measurements.

Generally, the obtained results, in particular the ones related to AES-256 and the ones acquired with the custom handmade USB-USB/BNC connector, are not totally compliant with what you'd expected from the theory. This phenomena is caused by the limits of the DPA tool (in the case of AES-256) and of the instrumentation (in the case of the custom handmade USB-USB/BNC connector). Moreover you have to consider also all the other problems found during the measurements acquisition, detailed in "Chapter Five - Measurements in Laboratory", besides that all the obtained results are around 50%, that is the theoretical probability of guessing a bit of the key choosing it randomly.

CHAPTER 7

Conclusions

The low numbers of hit bits obtained from the `Jlsca` tool is compliant with the results reported in [13], where is reported the description of a DPA attack of AES-128 on SEcube^{TM} and on ST Microelectronics Nucleo board equipped with the same microprocessor and the authors found just half of last byte of the key attacking only the Sbox of the AES and they get even worse results considering the whole AES.
In the SEcube^{TM} FPGA case, described in this thesis, these results have several motivations.
First of all, you have to consider that the available instrumentation wasn't comparable with the one generally used in literature. Indeed, they always exploit commercial tools and instruments, like for example ChipWhisperer® (`https://newae.com/tools/chipwhisperer/`) or the large variety of solutions by Riscure(`https://www.riscure.com`) or SASEBO, the Side-channel Attack Standard Evaluation Board (`http://www.toptdc.com/en/product/sasebo/`), specifically designed for this kind of goals and with the maximum level of optimization possible.
Another important element is the impossibility of the use of an external power supply, that could be more sensible to the encryption (comparison in figure 5.20), in addition to the fact that also a filter designed *ad hoc* could be introduced as described in "Chapter Two - Differential Power Analysis (DPA) Key Concepts", but this problem is not resolvable due to the intrinsic nature of the SEcube^{TM} board, designed in order to minimize the accessibility and maximize the security.
A possible improvement, without spending resources for getting better tools and the instrumentation, could be trying to use a core that better fit the dimensions of the FPGA, because there could be the possibility that there would be a sufficient increment of the power consumption with respect the noise, to get, in combination with all the other mathematical tips and precautions, some improvements in the results. A focus on this aspect is not present because it wasn't an expected target and there are some difficulties in finding a way to get an AES-256/AES-128 core without area overhead due to the small dimensions of the FPGA. It couldn't be so easy applying this type of shrewdness and the improvements would be probably not so relevant, without considering that it wouldn't be neither so rational from a practical point of view, since generally an FPGA should be able to do also others operations on the data in the meanwhile, but it'd be interesting to make a comparison and verify these considerations.
Moreover, also reducing the noise, de-soldering the SEcube^{TM} BGA package from the board and using an external power supply, there'll always be the problem of the decoupling capacitors (fig.5.24), that causes the insensibility of the power supply to the running code and frustrates all the others efforts and this is the main problem that would be really hard to compensate without using more sophisticated tools. This problem is also reported in the article [20], where the authors reported negative results in the application of the DPA and they write that the FPGA board should be physically broken before successfully applying DPA, making it no longer a passive attack and that the decoupling capacitor is

a natural countermeasure difficult to overcome. Finally, they also observed that another limit is the presence of more than one electrical device on a board and they may overwhelm the power consumed by the reconfigurable logic, without considering that there would be more than one module concurrently running on the FPGA.
Another reference to this problem is in [27], where they showed that removing decoupling capacitors plays the major role in success rate of a DPA attack. Furthermore, they reported also that replacing standard switched-mode power supply with accumulators and linear stabilizers simplifies the attack, but its effect is not that significant, since, even if the circuit is powered from a noisy power supply, it can be successfully attacked incrementing the number of traces, while the presence of the decoupling capacitors is extremely more problematic.

Anyway, a future proposal of work on the SEcube^{TM} board, besides the acquisition of the traces exploiting a differential probe (not available in our case), could be analyse the possibility of application of other types of side-channel attacks, but every option will have to be carefully evaluated in relation to the connected costs and necessary instrumentation.
For example, there are new researches related to temperature side-channel attacks, or, moving towards an higher level of abstraction, like the architectural one, there is the possibility of a Differential Fault Analysis (DFA) attack realization: it contemplates the possibility of introducing unexpected environment conditions (high temperature, unsupported voltage or current, overclocking, strong electric or magnetic fields, ionizing radiation and so on...) to produce faults and to deduce the internal state of the device. Anyway, this kind of procedure could be too invasive and create some damages for the device, without considering that, in lot of cases, countermeasures like fault tolerance and error detection schemes are present.
Another possibility could be a timing attack, but this could be applied only for the microprocessor inside SEcube^{TM} board and not for the FPGA, since for the last one the time necessary for the execution of the cryptography is independent from the given inputs.

Bibliography

[1] L. Bossuet, "Teaching fpga security," in *2013 International Conference on Field-Programmable Technology (FPT)*, Dec 2013, pp. 306–309. vi, 4

[2] K. Meritt, "Differential power analysis attacks on aes," *Cryptography II VCSG-706*, 2012. vi, 5, 6

[3] M. Varchola and M. Drutarovsky, "The differential power analysis laboratory setup," in *Proceedings of 22nd International Conference Radioelektronika 2012*, April 2012, pp. 1–4. vi, 4, 7, 8, 9, 10, 25

[4] Z. Martinasek, V. Clupek, and T. Krisztina, "General scheme of differential power analysis," in *2013 36th International Conference on Telecommunications and Signal Processing (TSP)*, July 2013, pp. 358–362. vi, 5, 11, 18, 19, 20, 21, 25

[5] G. D. Natale, M. L. Flottes, and B. Rouzeyre, "An integrated validation environment for differential power analysis," in *4th IEEE International Symposium on Electronic Design, Test and Applications (delta 2008)*, Jan 2008, pp. 527–532. vi, 11, 12, 23

[6] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, Apr 2011. [Online]. Available: https://doi.org/10.1007/s13389-011-0006-y vi, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 20, 21, 25

[7] M. Hellman, "An overview of public key cryptography," *IEEE Communications Society Magazine*, vol. 16, no. 6, pp. 24–32, November 1978. vi, 29

[8] W. A. Al-Hamdani, "Missing factors in teaching cryptography algorithms for information security tracks," in *2009 Information Security Curriculum Development Conference*, ser. InfoSecCD '09. New York, NY, USA: ACM, 2009, pp. 15–20. [Online]. Available: http://doi.acm.org/10.1145/1940976.1940982 vi, 31

[9] J. Kay, "Cryptanalysis techniques: An example using kerberos," *School of Computer Science Carnegie Mellon University Pittsburgh, Pennsylvania 15213-3890*, 1995. vi, 32, 33, 36, 43

[10] A. Kuldmaa, "Block ciphers: The example of aes," *Seminar Report for Research Seminar in Cryptography*, 2014. vi, 34, 35, 36

[11] G. Airò Farulla (CINI Cyber Security National Lab), A. Carelli (CINI Cyber Security National Lab), P. Prinetto (President, CINI), G. Somma (Business Development Manager, Blu5 Labs Ltd), and A. Varriale (Managing Director, Blu5 Labs Ltd), "Secube™ development kit - getting started," Tech. Rep., 15 January 2017. vi, viii, 45, 46, 65, 71, 77, 78

[12] J. J. Paul Kocher and B. Jun, "Introduction to differential power analysis and related attacks," *http://www.cryptography.com/*, 1998. 4, 6

[13] M. Bollo, A. Carelli, S. D. Carlo, and P. Prinetto, "Side-channel analysis of secube; platform," in *2017 IEEE East-West Design Test Symposium (EWDTS)*, Sept 2017, pp. 1–5. 5, 24, 94

[14] J. J. Paul Kocher and B. Jun, "Differential power analysis," *Michael Wiener (Ed.): CRYPTO'99, LNCS 1666, pp. 388-397, 1999. Springer-Verlag Berlin Heidelberg 1999*, 1999. 6, 22

[15] M. Aigner and E. Oswald, "Power analysis tutorial," *http://iaik.tugraz.at/content/research/implementation_attacks/introduction_to_impa/dpa_tutorial.pdf*, -. 6, 8, 23

[16] B. Gierlichs, L. Batina, B. Preneel, and I. Verbauwhede, "Revisiting higher-order dpa attacks:," in *Topics in Cryptology - CT-RSA 2010*, J. Pieprzyk, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 221–234. 6, 23

[17] C.L.Pitu and R.Câmpeanu, "A practical approach to power trace measurement for differential power analysis based attacks," *Bulletin of the Transilvania University of Braşov Series I: Engineering Sciences Vol. 6 (55) No. 2*, 2013. 8, 23

[18] M. Petrvalsky, M. Drutarovsky, and M. Varchola, "Differential power analysis attack on arm based aes implementation without explicit synchronization," in *2014 24th International Conference Radioelektronika*, April 2014, pp. 1–4. 9, 24

[19] A. Moradi, M. Kasper, and C. Paar, "On the portability of side-channel attacks - an analysis of the xilinx virtex 4 and virtex 5 bitstream encryption mechanism." -, vol. 2011, p. 391, 01 2011. 16, 17, 18, 22

[20] S. Sun, Z. Yan, and J. Zambreno, "Experiments in attacking fpga-based embedded systems using differential power analysis," in *2008 IEEE International Conference on Electro/Information Technology*, May 2008, pp. 7–12. 16, 78, 94

[21] S. M. Trimberger and J. J. Moore, "Fpga security: Motivations, features, and applications," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1248–1265, Aug 2014. 16, 17, 24

[22] S. Picek, A. Heuser, A. Jovic, S. A. Ludwig, S. Guilley, D. Jakobovic, and N. Mentens, "Side-channel analysis and machine learning: A practical perspective," in *2017 International Joint Conference on Neural Networks (IJCNN)*, May 2017, pp. 4095–4102. 21, 23

[23] S. Bhasin, N. Bruneau, J.-L. Danger, S. Guilley, and Z. Najm, "Analysis and improvements of the dpa contest v4 implementation," in *Security, Privacy, and Applied Cryptography Engineering*, R. S. Chakraborty, V. Matyas, and P. Schaumont, Eds. Cham: Springer International Publishing, 2014, pp. 201–218. 22

[24] M. Bollo and P. Maistri, "Composite fields against side channel analysis for the advanced encryption standard," in *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*, Dec 2014, pp. 542–545. 24

[25] S. Sun, Z. Yan, and J. Zambreno, "Experiments in attacking fpga-based embedded systems using differential power analysis," in *2008 IEEE International Conference on Electro/Information Technology*, May 2008, pp. 7–12. 24, 26

[26] M. Masoomi, M. Masoumi, and M. Ahmadian, "A practical differential power analysis attack against an fpga implementation of aes cryptosystem," in *2010 International Conference on Information Society*, June 2010, pp. 308–312. 25

[27] L. Mazur and M. Novotný, "Differential power analysis on fpga board: Boundaries of success," in *2017 6th Mediterranean Conference on Embedded Computing (MECO)*, June 2017, pp. 1–4. 25, 95

[28] W. Luis, G. R. Newell, and K. Alexander, "Differential power analysis countermeasures for the configuration of sram fpgas," in *MILCOM 2015 - 2015 IEEE Military Communications Conference*, Oct 2015, pp. 1276–1283. 25

[29] J. Wu, Y. Shi, and M. Choi, "Fpga-based measurement and evaluation of power analysis attack resistant asynchronous s-box," in *2011 IEEE International Instrumentation and Measurement Technology Conference*, May 2011, pp. 1–6. 26

[30] M. Masoumi, "A dpa resistant fpga implementation of aes cryptosystem with very low hardware overhead," in -, 2012. 26

[31] J. P. Kaps and R. Velegalati, "Dpa resistant aes on fpga using partial ddl," in *2010 18th IEEE Annual International Symposium on Field-Programmable Custom Computing Machines*, May 2010, pp. 273–280. 26

[32] N. N. Anandakumar and S. Dillibabu, "Correlation power analysis attack of aes on fpga using customized communication protocol," in *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology*, ser. CCSEIT '12. New York, NY, USA: ACM, 2012, pp. 683–688. [Online]. Available: http://doi.acm.org/10.1145/2393216.2393330 26

[33] M. I. Aziz and S. Akbar, "Introduction to cryptography," in *2005 International Conference on Microelectronics*, Dec 2005, pp. 144–147. 27, 28, 29, 30

[34] W. Diffie and M. E. Hellman, "Privacy and authentication: An introduction to cryptography," *Proceedings of the IEEE*, vol. 67, no. 3, pp. 397–427, March 1979. 27, 29, 36

[35] R. Verdult, "Introduction to cryptanalysis: Attacking stream ciphers," *Institute for Computing and Information Sciences Radboud University Nijmegen, The Netherlands.*, -. 28, 31, 32

[36] B. Young, "Foundations of computer security - lecture 45: Stream and block encryption," *Department of Computer Sciences University of Texas at Austin*, -. 28

[37] F. Kuusisto, "Ancient and modern cryptography," *XRDS*, vol. 21, no. 3, pp. 57–57, Mar. 2015. [Online]. Available: http://doi.acm.org/10.1145/2748053 29

[38] P. K. Mohapatra, "Public key cryptography," *Crossroads*, vol. 7, no. 1, pp. 14–22, Sep. 2000. [Online]. Available: http://doi.acm.org/10.1145/351092.351098 36, 37

[39] B. Applebaum, J. Avron, and C. Brzuska, "Arithmetic cryptography," *J. ACM*, vol. 64, no. 2, pp. 10:1–10:74, Apr. 2017. [Online]. Available: http://doi.acm.org/10.1145/3046675 42

[40] A. W. Mohsen, A. M. Bahaa-Eldin, and M. A. Sobh, "Lattice-based cryptography," in *2017 12th International Conference on Computer Engineering and Systems (ICCES)*, Dec 2017, pp. 462–467. 42

[41] L. Kocarev, "Chaos-based cryptography: a brief overview," *IEEE Circuits and Systems Magazine*, vol. 1, no. 3, pp. 6–21, Third 2001. 43

[42] C. Peikert, "Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract," in *Proceedings of the Forty-first Annual ACM Symposium on Theory of Computing*, ser. STOC '09. New York, NY, USA: ACM, 2009, pp. 333–342. [Online]. Available: http://doi.acm.org/10.1145/1536414.1536461 43

[43] H. Qin, T. Sasao, and Y. Iguchi, "An fpga design of aes encryption circuit with 128-bit keys," in *Proceedings of the 15th ACM Great Lakes Symposium on VLSI*, ser. GLSVLSI '05. New York, NY, USA: ACM, 2005, pp. 147–151. [Online]. Available: http://doi.acm.org/10.1145/1057661.1057697 43

[44] P. Dongara and T. N. Vijaykumar, "Accelerating private-key cryptography via multithreading on symmetric multiprocessors," in *2003 IEEE International Symposium on Performance Analysis of Systems and Software. ISPASS 2003.*, March 2003, pp. 58–69. 43

[45] J. Tao, J. Ma, M. Keranen, J. Mayo, and C.-K. Shene, "Ecvisual: A visualization tool for elliptic curve based ciphers," in *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '12. New York, NY, USA: ACM, 2012, pp. 571–576. [Online]. Available: http://doi.acm.org/10.1145/2157136.2157298 43

[46] J. Tao, J. Ma, M. Keranen, J. Mayo, C.-K. Shene, and C. Wang, "Rsavisual: A visualization tool for the rsa cipher," in *Proceedings of the 45th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '14. New York, NY, USA: ACM, 2014, pp. 635–640. [Online]. Available: http://doi.acm.org/10.1145/2538862.2538891 43

[47] J. Ma, J. Tao, J. Mayo, C.-K. Shene, M. Keranen, and C. Wang, "Aesvisual: A visualization tool for the aes cipher," in *Proceedings of the 2016 ACM Conference on Innovation and Technology in Computer Science Education*, ser. ITiCSE '16. New York, NY, USA: ACM, 2016, pp. 230–235. [Online]. Available: http://doi.acm.org/10.1145/2899415.2899425 43

[48] K. Shahzad, A. Khalid, Z. E. Rákossy, G. Paul, and A. Chattopadhyay, "Coarx: A coprocessor for arx-based cryptographic algorithms," in *Proceedings of the 50th Annual Design Automation Conference*, ser. DAC '13. New York, NY, USA: ACM, 2013, pp. 133:1–133:10. [Online]. Available: http://doi.acm.org/10.1145/2463209.2488898 43

[49] B. Koziel, R. Azarderakhsh, and M. Mozaffari-Kermani, "A high-performance and scalable hardware architecture for isogeny-based cryptography," *IEEE Transactions on Computers*, pp. 1–1, 2018. 44

[50] M. Bollo, A. Carelli, S. D. Carlo, and P. Prinetto, "Side-channel analysis of SEcube^{TM} platform," in *2017 IEEE East-West Design Test Symposium (EWDTS)*, Sept 2017, pp. 1–5. 68, 83