User Friendly SCA Tool

Rui Miguel Alexandrino Tomé

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Ricardo Jorge Fernandes Chaves

Examination Committee

Chairperson: Prof. Nuno João Neves Mamede
Supervisor: Prof. Ricardo Jorge Fernandes Chaves
Member of the Committee: Prof. Gonçalo Nuno Gomes Tavares

October 2019

Acknowledgements

I would like to thank my advisor, Professor Ricardo Chaves, for giving me the opportunity to work in this area, which was unknown to me when this chapter started. His knowledge was essential to the development of this work, being always supportive and providing me flexibility for my own ideas and guiding me in the right direction when needed.

I would like to thank Tiago Francisco, Telma Fernandes, Luís Miguel, Gil Dias, and João Ramos for the support and for the great times that we have been through, as teams and as friends, allowing me to have great stories to remember later.

Last but not least, a special thank to my family for all the encouragement and endless support, since the beginning of my first day in this institution until the end of this journey.

Lisboa, October 2019
Rui Miguel Alexandrino Tomé

Dedicated to my family,

Resumo

A análise de canais laterais tira partido da fuga de informação que os dispositivos têm, com o objetivo de extrair a chave secreta. Existem múltiplas fontes de fugas de informação, tais como o consumo de energia, radiação electromagnética, tempo decorrido, ou calor emitido. Esses sinais necessitam de ser recolhidos e manipulados por software adequado de forma a permitir a um investigador analisá-los e executar algoritmos de análise de canais laterais de forma a conseguir extrair a chave secreta. Este processo é composto por duas fases: captura e análise. A captura consiste na recolha de dados do dispositivo em teste enquanto que a componente de análise trata da seleção, optimização e aplicação de algoritmos que permitem a extração da chave.

Contudo, as ferramentas de código-aberto disponíveis atualmente não acompanham o crescimento nesta área, faltando funcionalidades que permitiriam aos investigadores testarem os sistemas em análise. Neste trabalho, foi desenvolvida uma ferramenta flexível e optimizada que permite efetuar análise de energia electromagnética em dispositivos criptográficos, que combina a recolha dados enquanto controla uma plotter XY que suporta este processo. Para optimizar a componente de análise, uma ferramenta de código aberto de análise de canais laterais (ChipWhisperer Analyzer) foi extendida com capacidade de conversão de dados em Matlab, introdução de uma versão optimizada de CPA como algoritmo de ataque e suporte ao uso de código em Matlab, permitindo ter maior flexibilidade e desempenho no processo de investigação.

Abstract

Side-channel analysis exploits information leakage related to a device, with the aim of extracting the secret key. There are multiple sources of information leakages, such as power consumption, electromagnetic radiation, timing or heat. These signals need to be collected and handled by proper software in order to allow a researcher to analyze them, perform side-channel analysis algorithms, and be able to extract the secret key. This process comprises two components: capture and analysis. The capture component consists in collecting data from the device under test, whilst the analysis component tries to extract the secret key by optimizing the data in analysis and applying algorithms on it.

However, the open-source tools available today do not follow the growth in this area, missing features that would allow researchers to test cryptographic devices. In this work, an optimized and flexible tool that allows to perform power and electromagnetic analysis was developed. It supports a XY Plotter to assist the capture process of electromagnetic traces over a given cryptographic chip area, allowing to run analysis scripts on the collected traces. To optimize the analysis component, an open-source tool in side-channel analysis (ChipWhisperer Analyzer) was extended, providing the ability to convert data from/to Matlab. An interface to Matlab code and an optimized CPA algorithm were also added to the platform, providing flexibility and performance in the research work.

Palavras Chave
Keywords

Palavras Chave

Análise de Canais Laterais

Análise Diferencial de Potência

Análise de Correlação de Potência

Análise Eletromagnética

Plataforma de avaliação

Plotter XY

Projeto ChipWhisperer

Keywords

Side-channel Analysis

Differential Power Analysis

Correlation Power Analysis

Electromagnetic Analysis

Evaluation Platform

XY Plotter

ChipWhisperer Project

Contents

List of Figures

List of Tables

Acronyms

AES Advanced Encryption Standard

CNC Computer numerical control

CPA Correlation Power Analysis

DEMA Differential Electromagnetic Analysis

DES Data Encryption Standard

DPA Differential Power Analysis

DUT Device Under Test

EM Electromagnetic

EMA Electromagnetic analysis

FPGA Field Programmable Gate Array

GUI Graphical User Interface

HD Hamming-distance

HODPA High-Order Differential Power Analysis

HW Hamming weight

INESC-ID Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento

LSB Least Significant Bit

MIT Massachusetts Institute of Technology

PAA Power Analysis Attacks

RFID Radio-Frequency Identification

RSA Rivest–Shamir–Adleman

SCA Side-channel Attack

SNR Signal-to-Noise Ratio

SPA Simple Power Analysis

SSD Solid State Drive

1 Introduction

Cryptographic algorithms are responsible to protect the communications of millions of electronic devices. They provide confidentiality, integrity and authenticity of data. A cryptographic algorithm is a mathematical function that usually takes as input two parameters, a key and a message (plaintext), and maps these parameters to an output: the ciphertext. This process is called encryption, and follows the Kerckhoffs' Principle [4] that states that all the details about this process are publicly known and only the secret key is kept secret.

Two classes of cryptographic algorithms can be distinguished: symmetric and asymmetric. Symmetric algorithms are cryptographic algorithms that use the same secret key to encrypt and decrypt a message. Thus, the original message can be obtained using the same key that was used to encrypt it. Currently, the most widely used symmetric cryptographic algorithm is the Advanced Encryption Standard (AES) [5].

On the other side, asymmetric cryptography relies on a key pair to ensure the security of the messages. This means that every user has a key pair. There is a public key that is publicly known and a private key that is only known by his owner. In practice, the public key is used to encrypt plaintext or to verify a digital signature [6], and the private key is used to decrypt ciphertext or to create a digital signature. For example, if Alice wants to send an encrypted message to Bob using asymmetric cryptography, Alice uses Bob's public key to encrypt the message, and Bob uses his private key to decrypt it. In this class, the Rivest–Shamir–Adleman (RSA) algorithm [6] is the most widely used algorithm. However, these algorithms are computationally expensive and slower than the symmetric ones.

Cryptographic keys and cryptographic algorithms are typically implemented and stored in electronic devices. These devices are called cryptographic devices. Some examples of cryptographic devices are smartcards, tokens or Radio-Frequency Identification (RFID) tags. They are more suitable to store cryptographic keys due to their security and ability to provide the output of cryptographic operations to the outside. Algorithms that are popular today are considered to be secure. A cryptographic algorithm is considered to be computationally secure if breaking it, i.e. finding the secret key, requires a period long enough to be practicable. Part of the security of these algorithms relies in the number of bits of the key. As the number of bits increase, more effort will take to break it.

However, the implementation of a secure algorithm in cryptographic devices does not ensure the device's security. When algorithms are implemented in cryptographic devices, not only the security of the cryptographic algorithm is of interest, but the security of the whole system needs to be considered. The fact that these algorithms are implemented in software, hardware, or a combination of both gives

rise to new security risks.

Indeed, there are physical properties of the implementation that may provide additional, unintended sources of information. Power consumption, electromagnetic radiation, timing or heat dissipation are some properties that can provide some information. These sources are called side-channels and their exploitation is called Side-channel Attack (SCA). Both asymmetric and symmetric algorithms are vulnerable to these attacks [7, 8, 9]. Since Kocher et al. presented the first attack [10] in 1998, the aim of the research community has been to improve the efficiency of these attacks and the implementation of adequate countermeasures.

Researchers also aimed to integrate the most powerful attacks into a single evaluation platform that could assess the security of cryptographic devices. However, the platforms available today are either closed and expensive tools or platforms that miss features and optimizations. This document describes the development of an evaluation tool, with features, integrations and optimizations, that can assess the efficiency of side-channel attacks countermeasures implemented on cryptographic devices.

1.1 Motivation

To perform side-channel analysis it is necessary to collect measurements of a device, apply pre-processing techniques that improve the speed and effectiveness of the method chosen, and evaluate the results by statistical analysis. In order to deploy cryptographic services on devices, it is essential for the researcher to have a platform that can perform side-channel analysis from the beginning to the end. This platform can provide the tools to collect measurements from a target device, preprocess that measurements, perform the attack, and do post-processing analysis that will allow characterizing the attack countermeasures effectiveness.

Actually, most of the side-channel analysis tools available today are commercial tools that focus their compatibility with vendor's hardware. DPA Workstation from Rambus [11] or Inspector SCA [12] from Riscure are two of the most complete software available commercially. However, academically, there has not been so much progress in this area. MATLAB is also one of the most used software to evaluate the security of a cryptographic device, even academically. The lack of tools in this area leads to extensive use of MATLAB software due to their features in signal processing techniques and its popularity, but it is a closed and commercial software that is not focused on side-channel analysis.

This panorama was changed by Colin O'Flynn with the development of the ChipWhisperer [13] project. The aim of the project is to provide to new researchers in SCA the tools for their research. This includes both hardware and software platforms. Although the software developed by Colin was great, it misses features, integrations and optimizations necessary to comply with today's side-channel attacks.

During electromagnetic side-channel attack process, rigorous measurements of EM leaks must be done. This attack can be aimed by placing a EM probe above the chip surface, while a XY table scans the chip while it runs cryptographic operations. Although there are some tools (Universal GCode Sender [14], Benbox [15], etc) to control XY tables, they are not crafted to provide features that can relieve the process of trace collecting.

1.2 Objectives and Requirements

The goal of this project is to develop a flexible and optimized tool that allows evaluating the protection of a cryptographic device against side-channel attacks. To achieve the goal of the project, the development of an open-source tool that offers the following requirements, is needed:

- Support the use of components made in other platforms, through the development of an interface that allows to call external code;
- Development of a tool to control XY Plotter's in order to support electromagnetic attacks;
- Provide compatibility with existing platforms;
- Efficient memory management, in order to support large amounts of data;
- In order to allow the optimization of the attacks, develop features that can provide the assessment of the leakage of a given implementation.

1.3 Contributions

This work resulted in several contributions for the Chipwhisperer platform, namely:

- Add support to use CPA algorithms in Matlab on Chipwhisperer Analyzer software.
- Add support to use a faster CPA implementation on Chipwhisperer Analyzer software.
- Development of a tool to convert Matlab format files to Chipwhisperer, allowing to do analysis of collected power traces in Chipwhisperer.

In the community research, this work also had the following contributions:

- Add support to XY Plotter's software to assist the process of collecting data by controlling the plotter positions.
- Integration of an analysis component in the software.

1.4 Structure of the Document

The rest of this document is organized as follows. Section 2 provides an introduction to side-channel analysis and tools currently available to work this data. Chapter 3 describes the proposed solution and

in Chapter 4 the components developed for the platforms are presented. Chapter 5 presents the results of the implementation of the components implemented in ChipWhisperer. Finally, Chapter 6 concludes this document by summarizing its main points and future work.

2 Related Work

Modern ciphers like RSA [6] or AES [5] used in cryptographic devices are designed to be resistant against cryptanalysis. From a cryptographic perspective, the cryptographic algorithm is secure when it is computationally infeasible for an attacker to compute the secret key, even if the attacker has access to any plaintext/ciphertext pairs. Thus, the cryptographic algorithm acts like a black box that given an input, produces an output, and the intermediate values never leave the black box. However, when these functions are implemented in real world devices, security issues can arise from other layers of implementation, making it possible extracting the key of a cryptographic device.

As depicted in Fig.2.1, when a cryptographic device is performing any operation, there are characteristics that can be observed like power consumption, timing variations, electromagnetic emissions or heat that may leak information. These characteristics are called side-channels and are entry points to perform attacks that can be exploitable to successful recover a secret key. This exploitation is called side-channel attack (SCA).

Side-channels attacks make use of physical observations under the normal functioning of the device to extract secret-dependent information, so they are categorized as passive attacks. On the other hand, attacks who intend to change the normal conditions of the device are called active attacks. The goal of these attacks is to induce malfunctions on the device that can be used for cryptanalytic purposes.

The classification of side-channel attacks can be further divided into three categories:

- **Invasive attacks:** Invasive attacks allow an attacker to perform any action on the device in order to extract a secret key. The package of the chip is typically removed and the attack is performed using micro-probing or reverse engineering techniques. In probing attacks [16] an attacker tries to directly read the information traveling over bus lines. Such attacks are classified as passive. On the other side, if the circuit was modified, we are dealing with an active attack.

- **Semi-invasive attacks:** In these attacks, the package of the chip is removed, but unlike invasive attacks, no contact with the electronic circuit is done.

- **Non-invasive attacks:** Non-invasive attacks use only directly accessible interfaces of the device. The measurement of power consumption or the processing timing of a device is included in this category as passive attacks. On the other hand, tampering with the power line or the clock signal are active attacks.

As described in table 2.1, it is possible to distinguish three classes of implementation attacks:

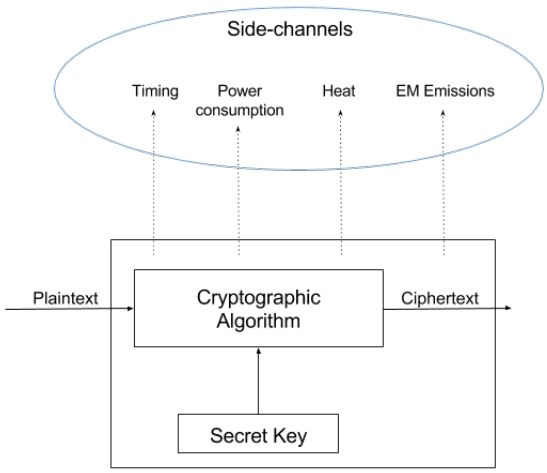

Figure 2.1: Side-channels from a cryptographic device.

	Non-invasive	**Semi-invasive**	**Invasive**
Passive	Side-channel analysis		Probing
Active	Fault Injection		

Table 2.1: Implementation attacks categorized by their nature and invasion degree.

This work focuses on two variants of side-channel analysis: power analysis and electromagnetic attacks. Power analysis attacks were presented to the community by Kocher in 1998 [10], with Quisquater and Gandalfi [17, 18] having coinciding results for electromagnetic analysis. Hence, these two approaches have proven to be highly effective to extract a secret key from a cryptographic device.

2.1 Power Analysis Attacks

Power Analysis Attacks (PAA) are one of the most powerful and common side-channel attack and has received a special attention from the research community since they were presented by Kocher [10]. By measuring the power consumption of a cryptographic operation, and analyzing this relation with the internal state of the cryptographic implementation, an attacker is able to extract a secret key embedded in security devices built in CMOS technology.

These attacks are based on two dependency factors: data-dependency and operation-dependency [10]. These dependencies show that different cryptographic operations have different power consumption and the switch of more transistors that depend on data will lead to higher power consumption. Analyzing the power consumption measurements of a cryptographic device will allow an attacker to make a guess on the secret key used.
A PAA generally comprises four steps :

1. **Equipment Setup** The first step involves the means to communicate with the target device (e.g. a smart card) and record its responses. This is done by using an oscilloscope hooked up to a PC and to a target device via probes, and capturing the power consumption at a determined

sample rate while the target device performs cryptographic operations. The PC is used to store the measurements and perform the analysis.

Figure 2.2 shows a possible configuration of a PA attack using a digital oscilloscope.

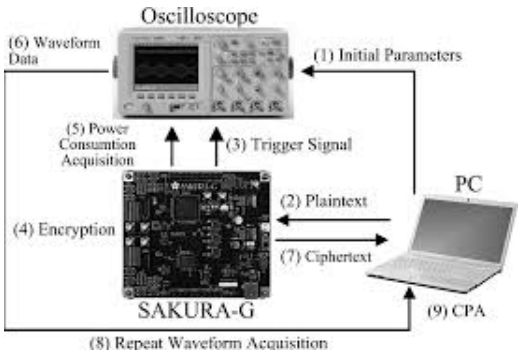

Figure 2.2: Equipment Setup

2. **Measure the power consumption** The collection of measurements can be obtained in a non-invasive manner, which means that the device is attacked only through its interfaces. In this step a set of plaintext (or ciphertext) are sent to the target device in order to encrypt (or decrypt) the value. The corresponding power measurement, also known as trace, is stored on the PC, with their (plaintext,ciphertext) pair. As needed, trace quality and capture efficiency may be improved by adding filters, adjusting bandwidth or sampling rates, and by exploring Simple Power Analysis (SPA) signal characteristics to remove irrelevant regions.

   As depicted in figure 2.2, the oscilloscope and a FPGA board are hooked up to a PC and to target device via probes. The PC has the oscilloscope and FPGA parameters set up, which will be triggered once the FPGA starts processing data. As a result, the oscilloscope will record the power consumption of the FPGA at the sample rate set up before, and send it to the computer. Analogously the FPGA returns the ciphertext produced given the plaintext and the cipher key. This process is repeated until the FPGA finishes processing data.

   In Fig. 2.3 a power trace from a SAKURA-G[1] board performing an AES-128 encryption is represented. It is possible to notice the 10 rounds of AES-128 clearly visible, so the cryptographic algorithm is strongly correlated with the power trace. Although in this figure the rounds are perfectly defined, this is not required for Differential Power Analysis (DPA) or Correlation Power Analysis (CPA) [10, 19] attacks because the use of multiple power traces will allow to reduce/remove noise, using statistical methods (as discussed in the next sections).

3. **Data Analysis** Once captured, the power traces are analyzed and processed by proper software. In this step, preprocessing techniques, e.g. alignment techniques, are applied to the power traces in order to improve their quality. This step also comprises the analysis of the power traces in order to remove irrelevant regions on which evaluation techniques are applied.

[1] A Field Programmable Gate Array (FPGA) board designed for research and development on hardware security.

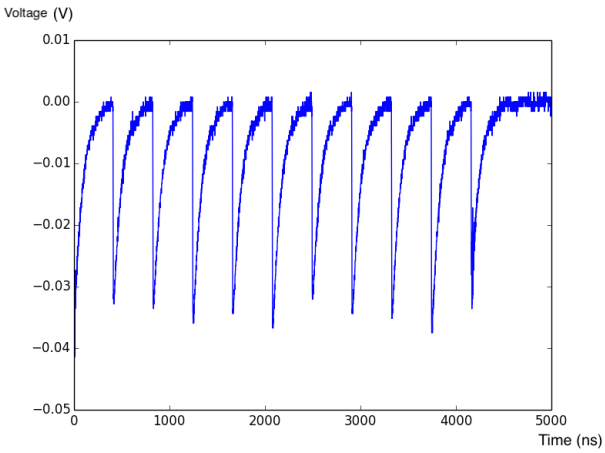

Figure 2.3: Power trace from a SAKURA-G board performing an AES-128 encryption

4. **Evaluation** This step consists in the analysis of the obtained information in order to extract the secret key. Two main methods can be used: Simple Power Analysis and Differential Power Analysis. The main difference is that simple power analysis does not use statistical techniques other than for noise reduction, while DPA does. On the other hand, DPA requires hundreds to millions of traces to be possible to apply these techniques, while SPA only uses a few traces.

Power analysis attacks are non-invasive, are relatively easy to implement and may not require the attacker to have knowledge of the target device implementation.

2.1.1 Simple Power Analysis

SPA [10] attacks try to retrieve the secret key by directly interpretation of the power trace. In this technique, operations from cryptographic algorithms can be identified by analyzing a single power trace or by comparing pairs of traces to find patterns.

SPA is useful where conditional branches depending on a secret parameter occur, since it can reveal the sequence of the instructions executed.

Figure 2.4 shows 2 power traces from Data Encryption Standard (DES). At the upper trace the execution path can be identified, at clock cycle 6, through an SPA feature where a jump instruction is performed. In the bottom trace, the jump instruction is not performed.

Simple power analysis requires a single or very few power traces and it works only with traces which have little to no noise. These attacks are relatively easy to defend against and detailed knowledge about the cryptographic algorithm that is used by the device under attack is required.

2.1.2 Differential Power Analysis

DPA [10] is a statistical method that exploits data-dependent correlation by analyzing power traces. It is more effective, and requires much more power traces than SPA. No knowledge about the crypto-

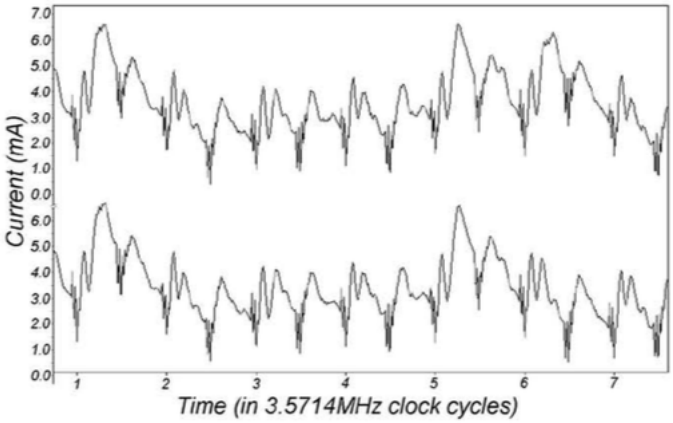

Figure 2.4: SPA traces showing individual clock cycles [1].

graphic device is required and, due to the large number of traces, the noise present in the power trace is less relevant. Also, the use of statistical methods allow the quantification of noise in the measured traces, using the Signal-to-Noise Ratio (SNR) [1]. The greater the ratio, the higher the leakage.

In a difference-of-mean attack, an attacker performs multiple encryption operations with different sets of data and records the power traces and the ciphertexts. Then, a set of power traces is partitioned into two subsets by a selection function and the difference of the averages of these subsets is computed. If this difference will approach a non-zero value, the partitioning into subsets is correlated to the traces measurements.

Let m be the number of encryption operations observed, C the corresponding ciphertext (for this attack, the knowledge of the plaintext is not required), and T the power traces containing k samples each. The power traces collected can be represented as $T_{1:m}[1:k]$. The next step is to use a selection function to split the power traces into subsets. Let $D(C, b, K_s)$ a selection function that computes the value of a target bit b given a ciphertext C and a key guess K_s. The power traces will be partitioned into two subsets either $D(C, b, K_s) = 0$ or $D(C, b, K_s) = 1$.

Assuming that the power consumption is larger when an intermediate value b is $b = 1$ vs $b = 0$. An intermediate value is a value that depends on part of the key (subkey) and plaintext. The power traces are sorted by the previous selection function resulting in two sets. One set will have the power traces with $b = 1$ and the other set will have the power traces with $b = 0$. It is expected than the average of the set containing the power traces with $b = 1$ be higher than the set containing the power traces with $b = 0$ and the difference between these two values should be positive. The next step is to compute a k-sample differential trace $\Delta_D[j]$ by finding the differences of averages between the two subsets. This is done by using the differential power trace formula presented in Kocher's original paper [10]:

$$\Delta_D[j] = \frac{\sum_{i=1}^{m} D(C_i, b, K_s) T_i[j]}{\sum_{i=1}^{m} D(C_i, b, K_s)} - \frac{\sum_{i=1}^{m} (1 - D(C_i, b, K_s)) T_i[j]}{\sum_{i=1}^{m} (1 - D(C_i, b, K_s))}$$
$$\approx 2 \left(\frac{\sum_{i=1}^{m} D(C_i, b, K_s) T_i[j]}{\sum_{i=1}^{m} D(C_i, b, K_s)} - \frac{\sum_{i=1}^{m} T_i[j]}{m} \right) \tag{2.1}$$

If the hypothesis K_s is correct, the average trace for $D(C, b, K_s) = 1$ will be slightly higher at the point of correlation and the average for $D(C, b, K_s) = 0$ will be slightly lower. Therefore, the bit computed by the function D will be equal to the actual value of the target bit b. Hence, $\Delta_D[j]$ should approach zero for a incorrect key guess, and a non-zero value otherwise.

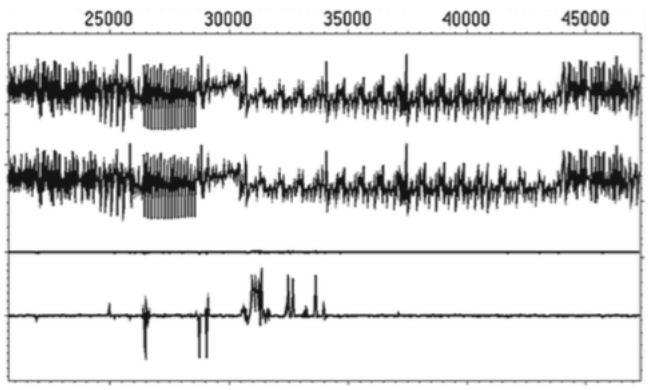

Figure 2.5: Figure illustrating a DPA attack [2].

In Fig. 2.5, the first and second trace represents the average of the subsets where the Least Significant Bit (LSB) of the output of the S-box (the selection function chosen) in the first round of AES is 1 and 0 respectively. The third trace represents the differences of these subsets, and it appears mostly flat because the differences are small. However, when this trace is scaled 15x, as depicted in the last trace, it is possible to notice spikes, meaning that there is correlation between the power consumption and the selection function output.

Although the secret key is unknown, and at first sight, it may require a considerable amount of time to compute the entire key for a modern cipher, this problem becomes minimized because the computation of the intermediate value only requires part of the key (8 bits for a 128-bit key). Thus, for a 128-bit key, the attacker only needs to try $2^8 = 256$ possibilities for each subkey, which is easily computable. The correct subkey will have the largest difference-of-mean. After the first subkey has been determined, this process is repeated until the entire key is successfully recovered.

It is important that the power traces should be correctly aligned. This means that the power consumption of each operation should be located at the same position in each power trace. If this condition is not fulfilled, DPA attacks cannot be properly done, unless alignment algorithms are applied (See section 2.5).

2.1.3 Correlation Power Analysis

This attack is an extension of DPA and exploits the correlation between data and power and it is useful when the number of traces available is limited. In CPA[19], a model of how the power consumption of the device depends on some intermediate value is required, i.e. power model or leakage model. The goal of the model is to approximate the power consumption of the target device during the encryption process. This simulation will be correlated with the actual measured power consumption using a key hypothesis, and the correct key will be the one which maximizes the correlation coefficient. Section 2.3 of this report details the possible power models.

To test the linear relationship between two variables, generally the Pearson's correlation coefficient is used. This value reflects the degree of linear relationship between two variables: a +1 value indicates that there is positive linear relationship between two variables, a -1 value indicates that there is negative linear relationship, and zero shows that there is no linear relationship. In CPA, we are looking for a correlation between the values generated by the leakage model and the measured power consumption.

For every hypothetical intermediate values, hypothetical power values are computed by a power model. If the power model and intermediate values are correct, then the hypothetical power values will have a linear relation with the measured values. For each point j over all the traces D, for each key guess the Pearson's correlation is calculated as in equation 2.2, where **i** denotes the key guess, **t** represents the power traces, **d** denotes the trace number (tnum), **j** represents the index trace point, e.g. traces[tnum][j], and **h** the hypothetical power value.

$$r_{i,j} = \frac{\sum_{d=1}^{D}[(h_{d,i} - \bar{h_i})(t_{d,j} - \bar{t_j})]}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \bar{h_i})^2 \sum_{d=1}^{D}(t_{d,j} - \bar{t_j})^2}} \tag{2.2}$$

This attack is faster and more accurate than DPA, however the computation of the Pearson's formula is computationally heavy. To speed up this calculation, data parallel computing can be used [20].

2.2 Electromagnetic Analysis

Electromagnetic analysis (EMA) is a technique that uses electromagnetic radiation to extract a secret key from a target device [17, 18]. EMA can be applied when the device has countermeasures against power analysis attacks or if it is not possible to install a power tap in the circuit [3].

It has proven to be useful when the circuit under analysis has countermeasures implemented against power analysis, such as adding filters that regulate and stabilize the power signal, as depicted in Fig. 2.6, or when the power pin is connected to other active areas on the chip that add noise to the power signal.

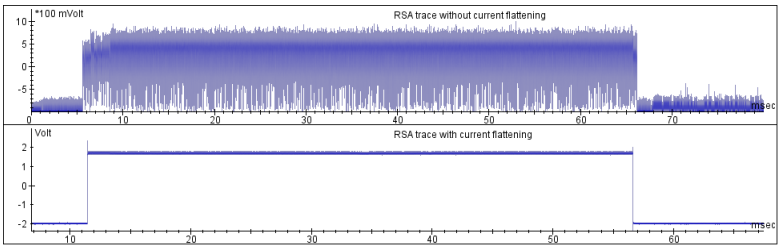

Figure 2.6: Power signal without and with current flattening [3].

2.2.1 EM Probes

An Electromagnetic (EM) probe is a combination of sensor and amplifier. To do an electromagnetic analysis, two types of EM probes can be used: H-field or E-field probes. Both use a coil as a sensor and its characteristics influences all the process.

Figure 2.7: H-field and E-field probes.

H-field probes (with a loop design) respond to magnetic fields produced by current changes and are sensitive to orientation, whilst E-field probes respond to electric field produced by voltage changes. The size of the loop determines the probe sensitivity, and smaller the area, smaller the loop size probe to be used.

2.2.2 Electromagnetic analysis methodology

Electromagnetic analysis methodology differs from power analysis on three points: aliasing, probe positioning, and preprocessing.

- **Aliasing** Aliasing occurs if the analog signal contains components at frequencies above half the sample frequency. Consequently, when the signal is reconstructed from the samples, it will be different from the original one. For high frequencies, the power signal is usually weak while the EM signal remains strong. Aliasing is therefore more relevant for EM signals. However, it can be prevented inserting a low pass filter between the EM probe and the oscilloscope input.

- **Probe Positioning** In EM analysis, a EM probe is placed above the chip surface, while a software controlled XY stage scans the chip while it runs cryptographic operations. When different processes with different clock frequencies run simultaneously, the EM signal should be filtered to reduce the frequency components that are unrelated with the interesting processes. Two criteria

can be applied for the best probe position: Select the strongest signal at the clock frequency for the process, or, when that option is not feasible, select the position that shows a pattern in the EM signal that can be related to the process of interest.

- **Preprocessing - Alignment** One of the methods to do alignment of EM traces is using the low frequency pattern of the envelope of the EM signal. An envelope can be obtained by calculation the absolute value of each sample, followed by a digital low pass filter. This process will allow the correct alignment between the traces. Alignment techniques will be approached in more detail in subsection 2.5.

- **Preprocessing - Downsampling** Downsampling process compresses data and increases the speed of the analysis process. In power analysis, a downsampling process is to average all measured sample within one clock cycle of the process of interest. However, due to the absence of low frequency components in the EM signal, this method should not be applied directly to EM signals because the signals contain approximately equal positive and negative peaks within one clock cycle, resulting on a low average value. Instead, the absolute value of the sample data should be computed.

As demonstrated by Quaisquater and Gandolfi [17, 18], the techniques to extract a secret key using EMA attacks are almost the same as performing PA attacks. EMA is attractive if components on chip filter leakage, add noise to the power signal or if a power tap is hard to install. However, the signals are required to be filtered. In other cases, power analysis is preferable and requires less effort to break a cryptographic device.

2.3 Power Models

A requirement in correlation power analysis is that there is a model which can approximate the power consumption of a target device during an encryption operation. If the target device has an architecture that is known, it is possible to simulate the device in a software environment design and get a precise prediction of the power consumption of the device. However, in most cases, the architecture of the device is not known or it is infeasible to simulate, so a more general power model must be used. Hamming weight [19] and Hamming distance [21] models are the most common ones.

2.3.1 Hamming-Weight

The Hamming weight (HW) model is a basic power model based on the assumption that the amount of power consumed by the device is proportional to the number of bits that are logic ‘1’ during an operation. This model closely correlates to real measurements where the power consumption of the physical implementation depends on the logic value of each bit being stored or moved through the system and can be used when little information is known about the architecture of a target device.

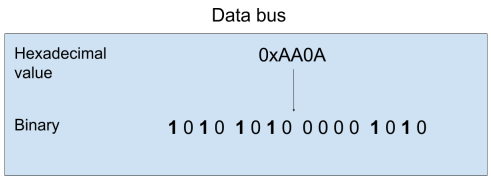

Figure 2.8: Consumes “6” power.

In figure 2.8, a data bus is represented with the value “1010101000001010”. The Hamming-weight of the previous value is 6, reflecting the six digits with value “1”.

2.3.2 Hamming-Distance

The Hamming-distance (HD) model is an extension of the Hamming Weight model and computes the differences between the logic values in a circuit during an operation to determine an approximate power consumption of it.

The power consumption from changing a value R_0 to a value R_1 will be the number of transitions from ‘1’ to ‘0’ and ‘0’ to ‘1’.

$$HD(R_0, R_1) = HW(R_0 \oplus R_1)$$

Figure 2.9: The Hamming distance between two register values is equal to the Hamming weight of both values XOR’d.

Two assumptions are made in this model:

1. The bits which do not change do not contribute to the power consumption of the circuit;
2. Transitions from ‘1’ to ‘0’ and ‘0’ to ‘1’ consume the same power.

This model should be used whenever a change in data can be observable and is more suitable to describe the power consumption in registers or data buses. This model also is generally more accurate than the hamming-weight model, making it preferable [1].

2.4 Countermeasures

The implementation of countermeasures in cryptographic devices aims to create independence between the power consumption of the device and intermediate values of the executed algorithm. The countermeasures can be divided into two groups: masking and hiding [1].

Masking

The idea of masking is to randomize the intermediate values that are processed by the cryptographic device. The big advantage of this countermeasure is that no power consumption characteristics of the device needs to be changed, and the power consumption can still be data dependent.

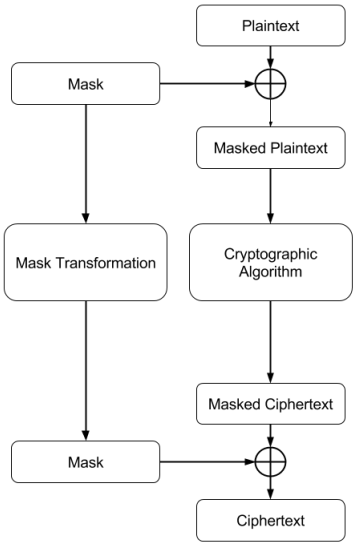

Figure 2.10: Implementation of masking countermeasure.

As depicted in Fig. 2.10 a mask can be applied to the plaintext before the encryption process occurs. This will randomize the intermediate values that will be used in the cryptographic algorithm. When the encryption process finishes, a X'OR operation is again applied in order to remove the mask from the ciphertext.

This countermeasure can be attacked by using High-Order Differential Power Analysis (HODPA) [1], where multiple sources of information can be correlated to determine the key.

Hiding

As opposed to masking, the concept of hiding is to break the dependency between the power consumption and the data processed. The goal is to randomize the device's power consumption or keeping it constant. This can be achieved by introducing random delays, random operations or shuffling during the execution of the algorithm or using special logic gates, or, by other side, building the device in such a way that every operation requires approximately the same amount of energy.

However, in practice the data dependency cannot be removed completely and devices implementing hiding countermeasures process the same intermediate results as unprotected implementations.

2.5 PA Attacks on misaligned power traces

In power analysis attacks the power traces must be aligned in order to apply the techniques described before. This means that the cryptographic operations must match in time in all traces. Clock jitter, absence of a suitable trigger, or hiding countermeasures are some of the reasons why the measurements can suffer from temporal misalignment.

In practice, the alignment of power traces is usually done in two steps: select a pattern in the first power trace and match this pattern in all other power traces.

Having determined the most probable location of the pattern in all power traces, the traces must be shifted in order to match the pattern at the same position.

Some publications deal with this problem using different methods. Charlet and Muijrers use wavelets [22, 23], Homma et al. use the phase-based waveform matching procedure [24], and Woudenberg proposes dynamic time warping [25].

2.6 Existing Plattforms

Currently, the side-channel analysis platform market can be separated into four platforms: MATLAB, ChipWhisperer, JIsca and platforms developed by industry specifically for side-channel analysis. Although there are other industry platforms, like Smart-SIC Analyzer from Secure IC [26], Inspector SCA from Riscure is one of the most complete tools in the area, and has more public information available about the platform, so it was chosen to represent this market. This section provides an overview of each of the platforms, presenting features that are included or missing in each of them.

MATLAB

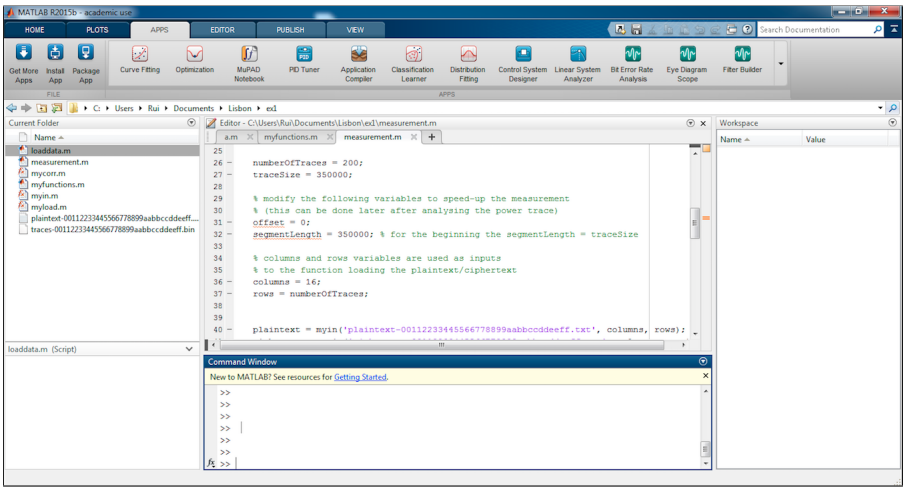

Figure 2.11: MATLAB R2015b.

MATLAB is a powerful mathematical tool to do signal processing but it is not a software focused on side-channel analysis. Through custom scripts it allows to perform analysis and process the power traces, develop new power models to be used statistically or any modifications that the researcher wishes to test at any point of the process. However it does not offer data capture options, or an interface that allows to apply models and techniques that are required for side-channel analysis in an easy way. All the process (data input, preprocessing, attack and postprocessing) must be done writing MATLAB code, which requires some learning curve, it is time consuming, and it is not intuitive. Also, MATLAB continues to be a closed, commercial and paid tool.

ChipWhisperer Plattform

ChipWhisperer [13] is an open-source project developed by Colin O'Flynn that consists in both hardware and software platforms for power analysis. The software has an interface designed to work on side-channel analysis and it is divided into two applications: ChipWhisperer Capture and ChipWhisperer Analyzer.

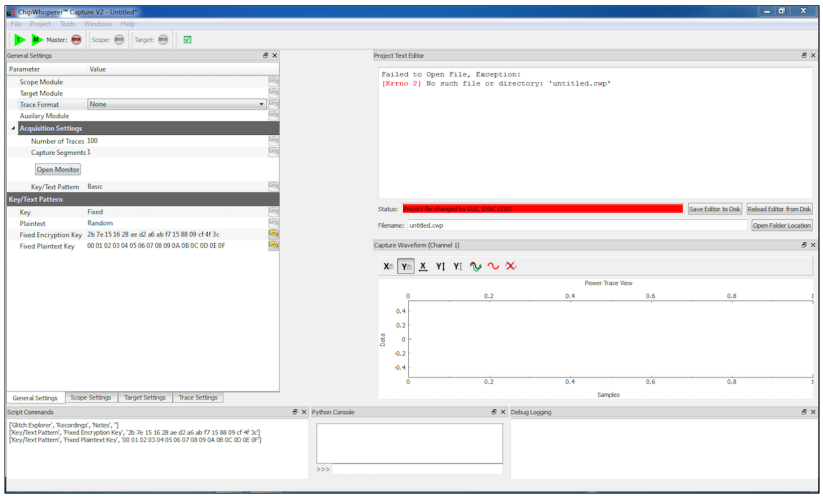

Figure 2.12: ChipWhisperer Capture.

ChipWhisperer Capture provides an interface to communicate with the oscilloscope, adjust its parameters, and the parameters of the target board (Fig. 2.12). A list of oscilloscopes and target boards are supported by default with the application, however modules can be developed by the community if the respective API is available. Define the number of traces, their length, the plaintext and the key to be used in the measurement are also possible in this software. Done the capture, the traces can be saved in ChipWhisperer or DPAContestv3 [27] format and the project can be saved.

On the other hand, ChipWhisperer Analyzer allows to analyze the data traces captured before the evaluation process. It has preprocessing modules that allows to fix misaligned traces, it is possible to choose the leakage model to be used in the attack, and also has post-processing plots where it is possible to see and explore the attack's effectiveness, that can be used to do some readjustments to the attack (Fig. 2.13).

This separation allows researchers with existing attack code to use the capture program and apply

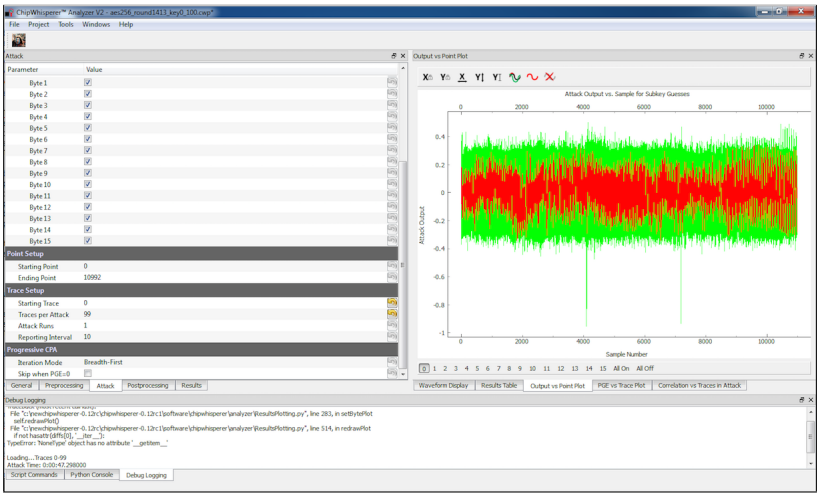

Figure 2.13: ChipWhisperer Analyzer.

it in another software, e.g. MATLAB. On the other hand, it also allows researchers with existing traces to analyze them without using the capture portion.

Several basic preprocessing modules are implemented on ChipWhisperer Analyzer, which operate on the data before passing through the attack. Three types of resynchronization can be found: a sum-of-errors minimizer, peak detect, and cross-correlation. A simple low pass filter is also provided. The waveform display window shows the results after the preprocessing. This is useful to check if the traces are properly resynchronized in the time domain before continuing onto the analysis.

All the software is composed by modules. Thus, the cryptographic model, leakage model, and attack algorithm modules can be changed independently.

However, ChipWhisperer Analyzer takes a considerable time to do statistical analysis, and it misses features that are required to evaluate cryptographic devices that have countermeasures implemented against power analysis attacks.

JISCA

JISCA [28] is a recently publicly available open-source toolbox written by Cees-Bart Breunesse and Ilya Kizhvatov. It consists in a toolbox to do the computational part of DPA. It is a very complete tool that has support to conditional averaging, conditional bitwise sample reduction, multiple aligment algorithms, CPA, parallezation in the computation process, split input and output samples, AES128/192/256 encryption/decryption, etc. JISCA works on two different type of Traces: InspectorTrace (from Riscure products) and SplitBinary, representing and splitting the data and samples used.

Although not part of the toolbox itself, a capture script which works on a Picoscope 5000 model is provided as example to collect traces. On the other hand, Jlsca does not have a GUI to work with, making the learning curve to work with Julia code and with Jlsca time consuming.

Industry Tools - Inspector SCA

In SCA industry, Inspector SCA from Riscure [12] is one very well known player. The software provides features to perform data acquisition as in ChipWhisperer Capture, supports power and electromagnetic analysis, it has signal processing features like filters or trace alignment and supports SPA, DPA/Differential Electromagnetic Analysis (DEMA) and CPA. Although it is a complete solution in this area, it is hard to do changes on some of the modules that are provided by the platform. Also, it is a commercial product, sold on a workstation with the software pre-installed, and it is optimized to work in hardware platforms also developed by Riscure.

Comparison between the tools

This section presents a comparative table that shows which modules or features are supported on MATLAB, ChipWhisperer, JISCA and Inspector SCA.

	MATLAB	**ChipWhisperer**	**Jlsca**	**Inspector SCA**
SCA IDE	-	x	-	x
SPA	Supported by custom scripts	-	x	x
DPA/CPA	Supported by custom scripts	x	x	x
DEMA	Supported by custom scripts	-	x	x
Filters	x	x	x	x
Trace Alignment	x	x	x	x
Type	Commercial	Open-Source	Open-Source	Commercial

Table 2.2: Comparison between MATLAB, ChipWhisperer, Jlsca and Inspector SCA features.

2.7 XY Tables, G-Code and GRBL

XY positioning tables can be used in side-channel analysis to support the probe positioning used at EM methodology. The Device Under Test (DUT) is assembled on a surface under the plotter axis, while it is moved along the device to measure its characteristics. One of the most popular languages used in these machines is G-code [29], a numeric control programming language designed by Massachusetts Institute of Technology (MIT). This language specifies the parameters of a Computer numerical control (CNC) machine job: which coordinates system to use, distance to travel, motor's feed rate, as well as more advanced parameters.

The fast growth of platforms like Arduino and Raspberry Pi allowed enthusiasts to build their own projects, which leads to the development of GRBL: a widely adopted open source controller to allow these devices to support G-code, in 2KB of RAM. This project is licensed under the MIT-license which allows the use of GRBL for any purpose.

As depicted in figure 2.14, the computer sends G-code to Arduino through the USB port, while the GRBL firmware interprets the code and sends it to stepper motor drivers, which transform the signal for stepper motors.

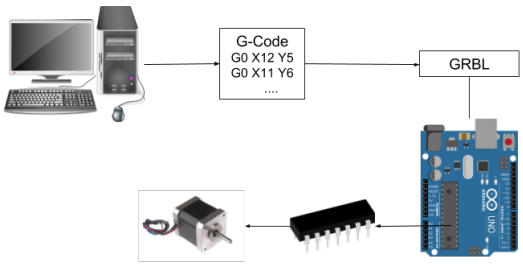

Figure 2.14: Plotter interaction.

2.7.1 Stepper Motor

CNC machines use stepper motors to move over its axis accurately. This type of motor can be moved continuously or one step at a time, forward or backwards, in a very precise increment, while also having holding torque to hold it's position when they are not moving.

Stepper motors convert the electrical pulse into mechanical movements to move an exact amount of degrees, being proportional the speed of the motor with the power being supplied. This provides control over the motor, allowing to move it to an exact location and hold that position until the next command is received. To move a stepper motor, the direction to move, the number of steps and the speed, must be supplied. In our case this is achieved by using Arduino with a GRBL ROM as stated in the previous section.

2.7.2 Existing platforms

This section compares three platforms used to control CNC Machines: Benbox, Universal GCode Sender and GRBL Panel. Benbox is supplied with the Plotter used in this research, and Universal GCode Sender and GRBL Panel are two of the most popular software to control them. The following subsections describe briefly each platform.

Benbox

Benbox is a software supplied with the plotter with the purpose of laser engravement. It is possible to either draw vector graphics in the software and print them or import vector graphics previously converted to G-code. This representation is a set of instructions of how to draw the vectors. Benbox also provides options for jogging the machine, allowing to provide the number of steps, directions and motor feed rate. In laser engravement options it is possible to control the engraving speed, burn time and engraving mode. This software only works with the LX-Nano ROM, which is proprietary and also supplied with the Plotter.

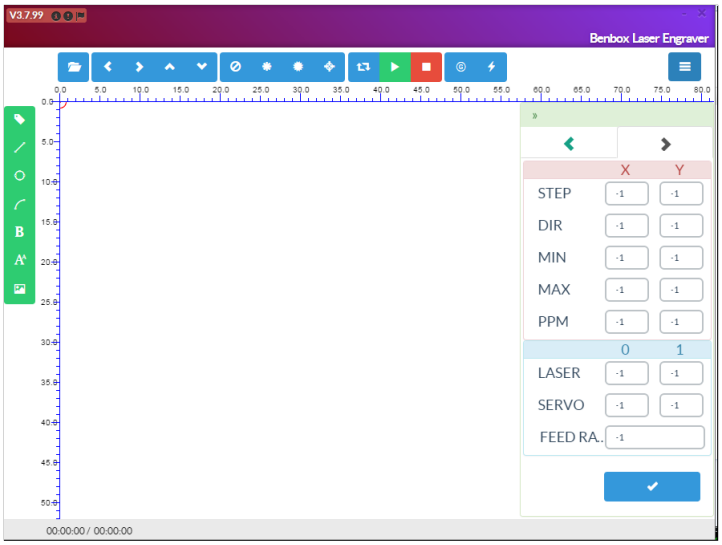

Figure 2.15: Benbox.

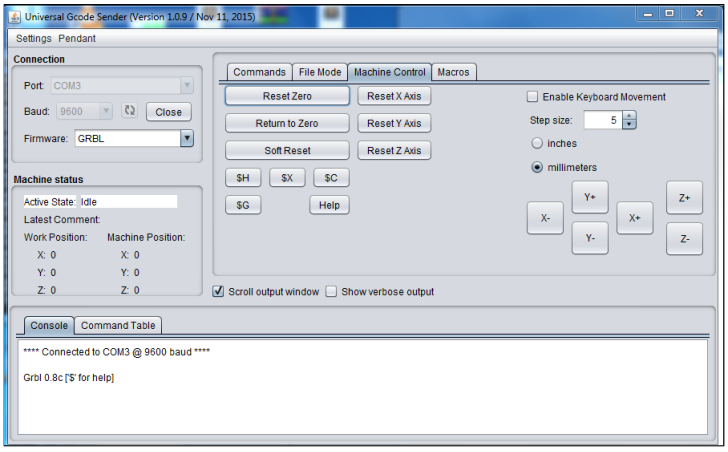

Figure 2.16: Universal Gcode Sender.

Universal Gcode Sender

Universal GCode Sender is one of the most complete solutions to control CNC machines. It is developed in Java, which allows its use in any platform, and it is open-source under the GPLv3 license. Some features of Universal GCode Sender are send G-code commands individually to the device, import files with G-code that describe a job and run it, write macros and store them in the plotter and jog the machine, as well define its properties. The software also allows to reset the plotter axis, allowing to define its zero position at the current machine position, or reset it, reverting its non-permanent properties, and canceling a job currently running. The plotter responses are also displayed at the log window. This software can use GRBL ROM'S which are open-source also under the GPLv3 License.

GRBL Panel

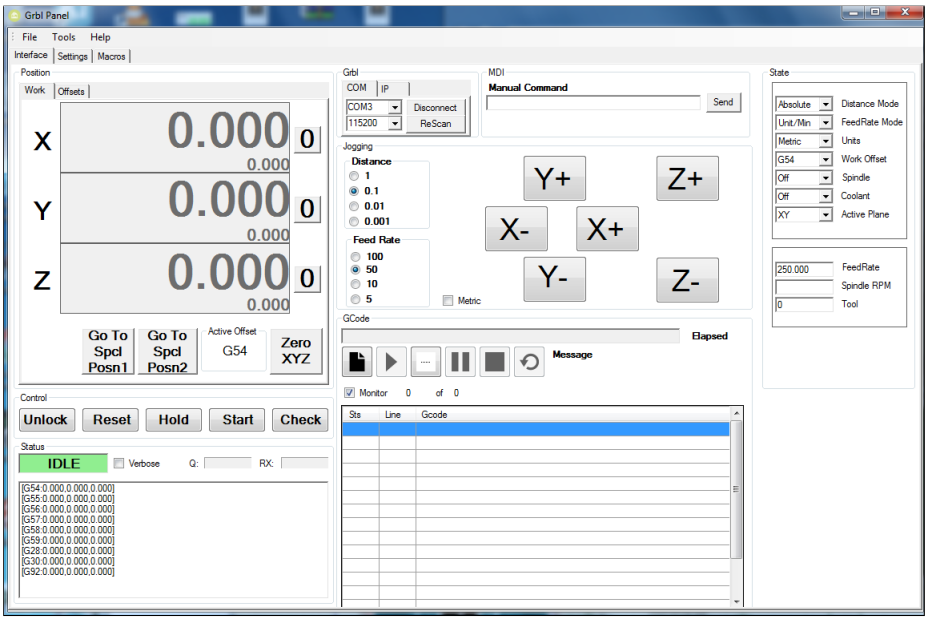

Figure 2.17: GRBL Panel.

GRBL Panel provides identical features as Universal GCode Sender. It allows to jogging the machine, by defining the distance and feed rate intended, load G-Code in order to execute a job, send manual commands or write G-code macros. It's development is in Visual Basic making this software focused on Windows. It is open-source under the MIT License.

Summary

In this chapter three concepts of Power Analysis Attacks: SPA, DPA and CPA were described. Electromagnetic analysis were also presented, describing their methodology. Common platforms were then presented to analyse the information collected in side channel attacks and to control CNC machines. It was also demonstrated that open source software to connect these platforms is currently missing.

3 Proposed Solution

In this chapter two modules to address the objectives stated in section 1.2 are presented. As noticed in the previous chapter, the tools developed to control CNC machines are designed only for that purpose and not for side-channel analysis. To support electromagnetic analysis it is proposed the development of a tool to control a XY plotter, while also having capabilities to capture traces and analyze this information. With this tool it should be possible to specify an area of interest of a cryptographic chip, collect the capture information over each point of that area, and analyze it by running a CPA algorithm with the previous data.

In order to support components made in other platforms it is proposed the expansion of ChipWhisperer Analyzer V3.2.0 with MATLAB support. MATLAB has a strong presence in academic research, and most of the CPA scripts developed by other researchers at Instituto de Engenharia de Sistemas e Computadores - Investigação e Desenvolvimento (INESC-ID) are developed for this platform. However, the data processing that is required before and after CPA computation turns this job time consuming. The fact of ChipWhisperer have an open-source project with an user friendly interface, and have an integration of preprocessing algorithms built-in, leads to the goal of expanding this software to support MATLAB on it.

It is also proposed the implementation of a less memory consuming CPA algorithm on ChipWhisperer Analyzer to support large amounts of data.

3.1 XYPlotter

Electromagnetic analysis is a powerful technique that can be used to complement or be used when it is not possible to use power analysis. This process can be assisted with a XY table. In this case, their objective is to assist in the process of data gathering while a chip runs cryptographic operations. INESC-ID has one plotter with this purpose (figure 3.1), however neither the software that comes with it (Benbox [15]) or open-source software was developed to work in side-channel analysis research.

Thus, the control of the XY table must be implemented. It is then proposed the development of a software with the following requirements:

- Ability to jog the machine using buttons;
- Ability to move the plotter to a point of coordinates XY;
- Specify the steps/mm of the motors;

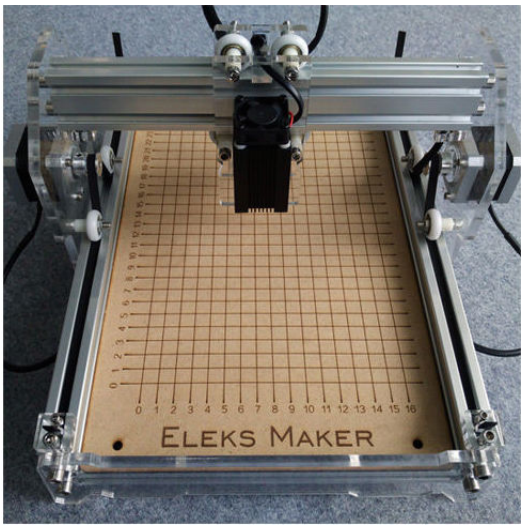

Figure 3.1: XY Plotter.

- Limit the distance traveled by the motors in order to not override the maximum allowed;
- Specify a region of interest to be analyzed;
- Map the region of interest in a number of points;
- Run a capture on each point of the region of interest;
- Analyze the data previously collected or at capture moment;
- Save the capture output and analysis process to a file;
- Save the plotter configurations to a configuration file;
- Load configurations from saved configuration files;
- User friendly interface.
- Automation of the trace gathering process.

The use of configuration files will allow to set and load different plotter parameters or different areas of interest. This makes easier for a researcher to analyze multiple areas without having to set the parameters every time a measure is required. The capture and analysis process is highly tied with the research in progress at INESC-ID, being the capture scripts developed for the equipment used. It is also a requirement that this interaction should need the minimum changes in the code already in use that is used for trace collecting. Despite this requirement, it should be easy to replace of these modules with the intended ones, in order to support different projects.

3.2 Chipwhisperer Analyzer Expansion

One of the most complete open-source project in side-channel analysis available today is ChipWhisperer. ChipWhisperer is developed in Python, it is composed by modules, and has a community of new researchers interested in the project. Taking these factors, this will be the base platform to be considered in this work. MATLAB has a strong presence in side-channel analysis though it does not have an IDE for that purpose. Researchers use MATLAB for development and to work with custom power models. Thus, it is then required the development of a module for ChipWhisperer Analyzer to support it. Supporting this module will also provide ability to do pre-processing techniques over the traces collected with ChipWhisperer, allowing to use this data further in Matlab. This is possible due to Matlab Engine API Python module provided with the Matlab installation, only requiring its installation on Python.

In order to be possible to analyze the traces collected in XYPlotter, as well the traces previously collected before this work, a tool need to be developed in order to convert Matlab captures to a ChipWhisperer project. Additionally, it will also be possible to use the ChipWhisperer data in Matlab.

Currently ChipWhisperer has two CPA implementations: Simple and Progressive. The simple algorithm is a very simple and non-optimized (using libraries to manipulate large data) version of CPA in Python. On the other hand, Progressive uses an optimized version of the algorithm, however their intent is to be used providing a detailed information about the analysis, like correlation vs traces in attack or output vs point plot to see the sample where the known correct key was retrieved. This is done by storing the correlation values at a given report interval. However, for a analysis when this information is not needed, this algorithm is time and memory-consuming. Thus, it is intended to implement a optimized CPA algorithm in order to provide a faster and less memory consuming analysis over the data collected.

Summary

In this chapter the work to achieve the goals proposed was presented. This work will be developed in two phases: The first one will be the development of a software to control a CNC machine and integrate the process of collect and analysis captures on it, with the requirements specified above. The second phase will be the expansion of ChipWhisperer Analyzer software in order to support different algorithms of side-channel analysis developed in Matlab and support a faster implementation of Pearson Correlation attack. ChipWhisperer and MATLAB use two different data structures, so modules to provide conversion between tools will also be developed.

4 Implementation

4.1 XYPlotter

XYPlotter is a software developed to control XY tables with the goal of collect information from DUT analyzed area. It allows to map the cryptographic chip's surface under a number of points, allowing to find the locations where information leaks occurs. Given an interest area, it allows to run captures on each point of it, and run a CPA algorithm over these captures, given regions of interest (traces and samples) and selecting the bytes the researcher is interested on. In this section it is described an overview of XYPlotter and its Graphical User Interface (GUI). Then, technical details of how the system communicates with the plotter are presented, as well how points distribution are made in order to map them. Finally it is then described the capture, analyze and settings modules.

4.1.1 Overview

The lack of software for control CNC machines under side-channel analysis conditions was the reason for the XYPlotter development. The software's purpose is to join the control of a XY plotter while also having capabilities of running external capture and processing scripts for EM analysis. XYPlotter is a tool developed in Python, it uses PyQt as GUI framework and it is composed by three main modules: UI, Control and Connection. UI is responsible for all the user interface modules, like the main application and its components, settings, send commands, and save and load dialog boxes. Control has the business logic related to application and the connection module is responsible for the serial connection with the plotter. The following sections describe in detail each of these modules.

4.1.2 GUI

The purpose of a GUI is to provide a user friendly way of do an action. Two of the more populars GUI frameworks in Python are PySide and PyQT. PyQT 5 was chosen to use in this project due to compatibility reasons with the "Map" area. The UI from XYPlotter is provided from the UI module, which has the most of UI elements, its labels and describe how this elements should be organized. The UI module is instanced when the Main is loaded, making the interaction with the elements possible.

The application is divided into the following sections:

1. **Connection** - It is used to select a serial port and respective baud rate to establish the plotter communication.

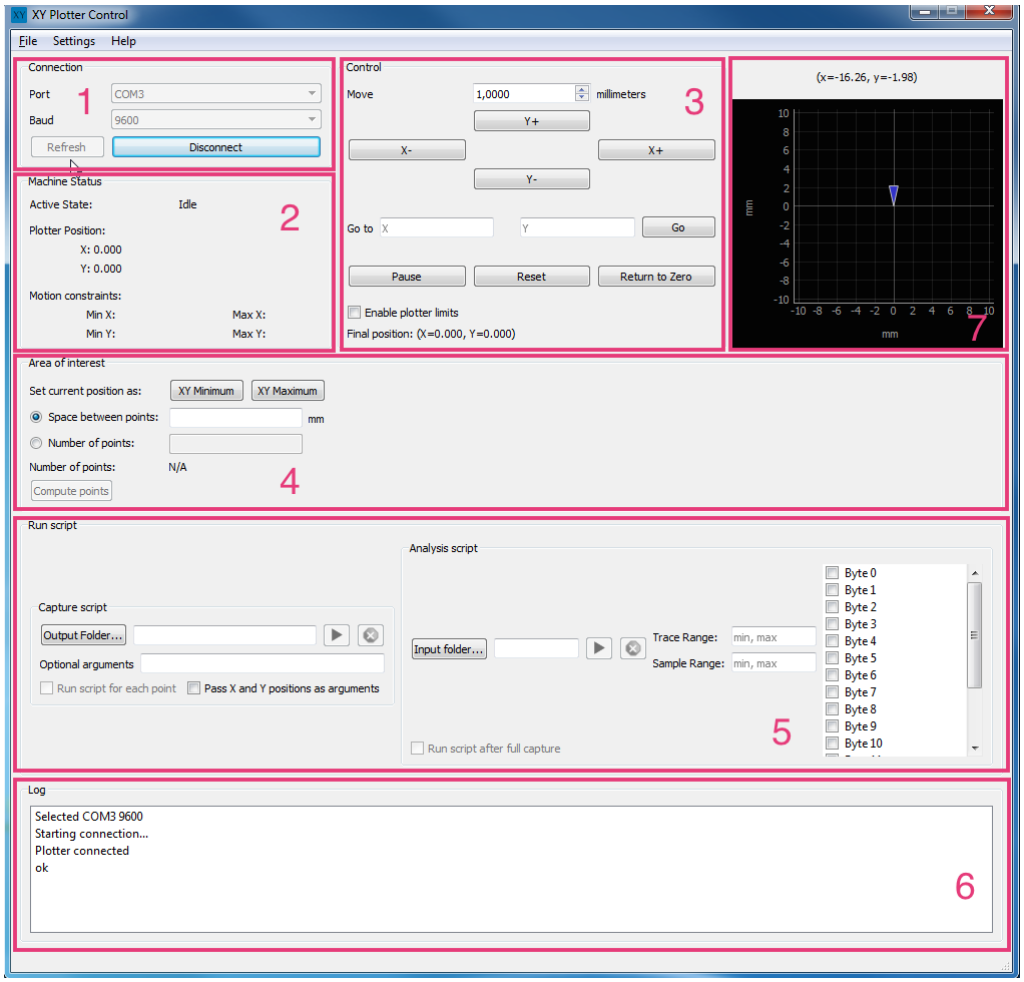

Figure 4.1: XYPlotter main menu.

2. **Machine Status** - It shows the plotter's state (idle, active, paused), its location and motion constraints defined at settings.

3. **Control** - This section implements the plotter control. Jogging commands are displayed in this area, as well the commands to pause, reset, send the plotter to absolute positions, and enable plotter limits.

4. **Area of Interest** - By defining (using 3) the minimum and maximum limits of an interested area, this section shows the region of interest to be analyzed. This region is then divided into points by two available methods (distance between the points or number of points provided as input), and the points are mapped at region 7. This information may be used in the next section.

5. **Run Script** - This section allows to collect and/or run CPA over traces previously collected. It is divided into two independent areas: Capture script and analysis script. The capture script allows to collect traces over the plotter's current position or by selecting a region of interest at 4. The analysis process can run after the capture is complete by checking the "Run script after full capture" checkbox or by selecting an input folder where previous traces were collected. Once the trace, sample range and bytes are selected, the process starts and the capture saved in

Matlab format is converted to Numpy format and, in order to allow the analysis of the capture in ChipWhisperer, a ChipWhisperer configuration file is generated. The Numpy files will be used later in CPA algorithm, and a log with the results is produced. A more detailed description can be found at section 4.1.6.

6. **Log** - This area shows the application log and responses sent by the plotter.

7. **Map** - Plotter position, regions of interest and physical limits are shown in this area.

4.1.3 Interaction between Plotter and GUI

The three main modules of XYPlotter are UI, Control and Connection. The UI module has the logic related to the UI, such as the creation of elements in UI, its behavior, updates in GUI, and dialog boxes. This module connects with the Control Module, which has the most of business logic of the application in the Plotter object. By creating a Plotter instance, the UI module can now get the information needed to update its interface.

The Plotter object stores the attributes related with the Plotter, such as its position, steps, units, and a Serial Instance. This instance is created when a connection starts, allowing to to write and read data from the plotter. When a connection starts (using the connection module and passing the port and baud rate to connect with), information about the plotter configuration present in UI is sent to the plotter. Reads and writes from and to the plotter are made using the send_command and read methods, which encodes the commands before being written to the serial port, and convert the bytes received to string.

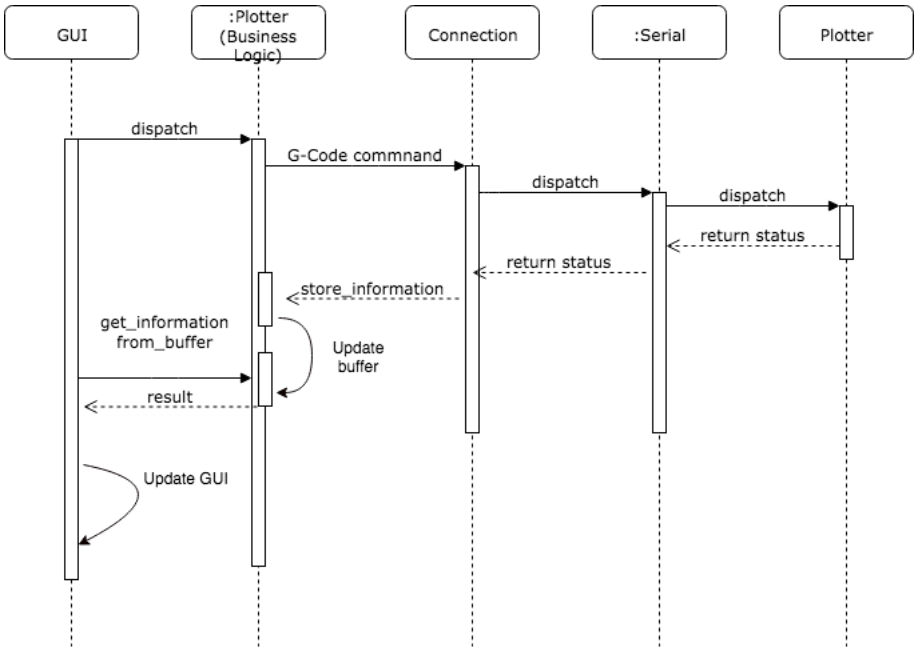

Figure 4.2: Sequence diagram of XY Plotter.

The interaction with the plotter is made by methods containing the structure of GRBL commands and sending this information to send_command. From the moment the connection with the Plotter starts,

two Threads must be started in order to querying the plotter its position and read the data received. The plotter position is obtained by sending the command "?" every 0.3s. In GRBL 0.8c, the response to that command has the following structure: ⟨*Status,Mpos:X_Pos,Y_Pos;Z_pos,Wpos:X_Pos,Y_Pos,Z_-Pos*⟩. The first parameter is the plotter status at that moment (in case of the plotter is running the status is Running) and the X,Y and Z_Pos are the positions of X,Y and Z in machine position and work position. The machine position and work position can be the same if work position is not used, however work position can be used to store a different profile from the machine position. This is a feature from which CNC machines can benefit, but it has no use in this work. After parsing the position, the attributes are updated, and this information is reflected on the GUI. Other responses are stored in a buffer, after checking that the received message has content, and is unique from that command, with the goal to avoid duplicated responses. These responses will be consumed by the GUI and displayed at Log every 0.1s. This interaction is depicted in Figure 4.2.

In order to support custom commands needed to control the plotter, the **Send Command...** dialog box was added to the File menu. This dialog box receives an instance of Plotter and uses the send_-command method from it to send the command to the plotter. The response is treated in the same way as other commands sent by the GUI.

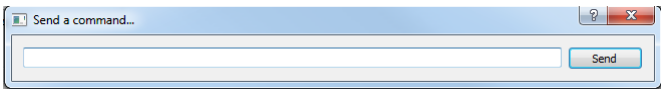

Figure 4.3: Send Command dialog box.

Two types of modes can be used when a plotter movement is intended: Incremental and absolute mode. The incremental mode adds a number of units to the current position, while the absolute mode accepts a position. In GRBL, these two modes have the following structure:

1. **Incremental Mode** - COMMAND = SET_INCREMENTAL_MODE + SET_RAPID_POSITIONING + direction + val
2. **Absolute Mode** - COMMAND = SET_ABSOLUTE_MODE + SET_RAPID_POSITIONING + X + XPOS + Y + YPOS

The constants SET_INCREMENTAL_MODE and SET_RAPID_POSITIONING represent the G-code described in GRBL firmware documentation, and the direction value in the first command can be X+/X-/Y+/Y-, representing the positive and negative motion of the motors. The val parameter is the value of units to move, and XPOS and YPOS represent the exact (absolute) position to move to.

The direction and values are gathered from area 3 in Figure 4.1. Single moves will trigger an incremental move, whilst "Go to" parameter will trigger an absolute move.

Before any movement, checks are made in order to verify that the positions are valid and are inside the plotter physical limits. Some additional checks must be made in points distribution, as described in section 4.1.5.

The map module is an object from **pyqtgraph** Python library containing a grid 20x20 (mm). It contains an arrow representing the plotter's position, which gets this information from the Plotter object. When plotter physical limits are enabled, this information is also provided from the Plotter object.

4.1.4 Serial Communication

In order to communicate with the plotter, the **pyserial** [30] library was used. This library is used in Connection module, which is a wrapper to pyserial. In this module the following methods were implemented:

- **serialports()** - Returns the list of available ports to establish communication with. This is done by trying a connection to each of the 256 ports available [1], and adding to a list the ones which were successful.
- **connect(port, baud)** - Establish a communication with the port at a given baud rate. For the GRBL 0.8c, the recommended baud rate is 9600 bits/s. This method returns an Serial object connection, which will be used to read and write data to.
- **close(connection)** - Closes the connection provided as argument.
- **read(connection)** - Reads the data stored in the buffer of the connection.
- **write(connection, command)** - Writes data to the connection object.

As depicted in Figure 4.2 this is the module which establishes the plotter communication, and it is used to act as a wrapper to pyserial, by adjusting some parameters to the GRBL ROM (GRBL 0.8c) used in this project.

4.1.5 Points distribution

In order to collect information over different locations of a cryptographic chip, a distribution of points must be made over a given area. XYPlotter provides two methods to make this distribution. Both require that minimum and maximum limits should be defined, representing the area in analysis. The first method is d̈istance between pointsïn millimeters - by dividing the limits over N mm/steps provided in the GUI.

The second one is providing the number of points that the researcher wants to analyze. Given this information, we want to generate a uniform random sampling over the surface limits. A uniform distribution was the first approach to achieve this: Given the minimum and maximum limits for each axis, and using the rounded square root of the number of points to generate the same number of points to **x** and **y**.

As noticed in figure 4.4 and 4.5, this approach does not solve the problem because the values are not “well spaced” and the total of points is different of the number of points intended, due to the rounding up used in the square root. In fact, we want to use (pseudo-)random numbers, but not really random numbers. We want to use a random sequence that is low discrepancy, i.e. a sequence where the possibility of the values being very closed (discrepancy) is low and ensuring that the area is covered uniformly. These sequences are called quasi-random sequences [31] or low-discrepancy sequences. In

[1] Maximum number of serial ports on a Windows computer.

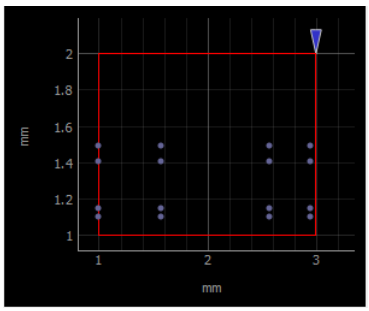

Figure 4.4: Uniform distribution for 12 points.

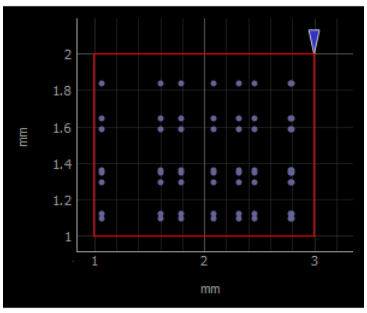

Figure 4.5: Uniform distribution for 50 points.

this work, Halton sequences [32] for dimension 2 were used, which are a generalization of van der Corput sequences to multi-dimension. However, for higher dimensions this sequence starts to deteriorate quickly [33]. To address this problem, Sobol sequences [34] were introduced by Sobol.

To implement the Halton sequences in the project, the ghalton library [35] was used. By specifying the dimension, in this case 2 for X and Y axis, and the number of points intended, the object returns a list of N points, between [0,1] with the specified dimension. Each point is then mapped to the minimum and maximum limits of the respective axis. This is addressed by using the following formula, which maps a point X from an interval [A, B] to an interval [C, D]: **Y = (X-A)/(B-A) * (D-C) + C**. The point Y will be the point mapped to the intended area.

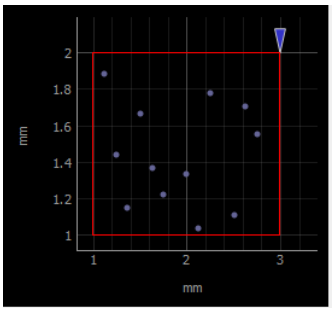

Figure 4.6: Halton sequence for 12 points.

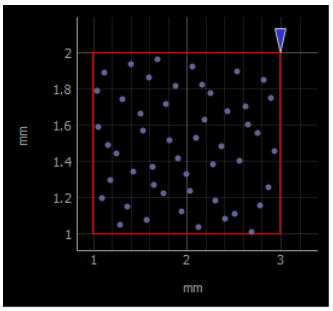

Figure 4.7: Halton sequence for 50 points.

Once mapped, the points will now display having a low discrepancy, and inside the limits defined, as depicted in figure 4.6 and 4.7.

In both methods, before these positions can be used in the plotter, a rounding up must be done due to invalid values between decimal parts of the position and the steps configured for the respective axis.

```
function fix_position(position,steps):
        approximation = ROUND_UP(Decimal_part(position) * Steps) / Steps
        return Integer_part(position) + approximation
```

For example, using the fix_position function for the position 2.05 and steps configured in 250 steps/mm, the following correction will be done: The decimal part of 2.05 is 0.05 which results in 0.05 * 250 = 12.5 steps. This result will block the plotter, because there is no half steps. In order to fix this, the step value is rounded to 13, resulting in 13/250 = 0.052. This value will be the decimal part used for this position. This rounding will be applied to all the values whose number of steps are not an integer number, preventing the plotter from blocking.

After this process, the points are represented in the visualizer module (Map in Figure 4.1).

4.1.6 Capture and Analyze

With the capture module, a capture can be obtained easily, making possible to just run it for the plotter's current position or for a distribution of points previously defined. The analysis module makes it possible to run a CPA script over a given capture, and also convert the capture from Matlab to a ChipWhisperer project allowing to be analyzed in this platform. Both run external scripts, being the capture script already in use at INESC-ID, containing the acquisition parameteres and requiring minimum changes. Further, as a last step, both modules share the fact of a subprocess for the respective script is spawn, registering the PIDs for pause, continue, or kill the process options. Threads are also used to start the subprocesses in order to prevent the GUI from blocking. These are the simple processes of run an analysis over a folder containing traces, or run a capture for just a single position. The following sections describes each module more detailed for a more complex analysis:

Capture

When a capture is initialized, the first step is to register the timestamp for the respective capture. This will be the name of the folder inside the output folder chosen, and will have the following format: **yyyy-mm-dd_HH-MM-SS**. The positions where the capture was acquired are also added to the folder name, as last parameters. This will allow to sort the folder name by capture date. The capture module also allows to run the script analysis after the full capture is complete, by checking the respective checkbox in GUI. If the checkbox is checked, the analysis arguments (trace range, sample range, bytes) will also be used.

To start a capture, the following method of Plotter object is used, and a log file is created, replacing the current standard output:

- **start_analyze**(file, args, opt_args, pass_positions=False, run_each_point=False, file_analysis=None, arg_analysis=None)

File and file_analysis indicate the filepath of capture and analysis scripts respectively; args represents the output folder, and opt_args the optional arguments that can be used inside the script; pass_positions indicates if the plotter position should be used as argument to the capture script, and run_each_point makes possible to run the capture script for each one of the points obtained in 4.1.5.

When run_each_point is enabled, two Threads must be synced in order to move the plotter to the point and hold on until the capture is complete. The capture can only start when the plotter finishes his movement. If pass_positions or arg_analysis is used, the capture script will also receive these parameters, which can be used for additional operations. The start_analyze method is also used when an analysis is required after a capture and run for each point is disabled. The capture script is then triggered, containing the oscilloscope and FPGA configurations.

When the captures for each point are complete they are stored in the respective folder. In order to write to the log file pending data, a flush is done to the standard output and the plotter returns to its zero position.

Analyze

This module uses a modified version of the Python CPA script implemented in Chipwhisperer, described later, and provides a GUI to it. In order to run a correlation power analysis script, four parameters are supplied: the traces file containing the samples gathered from the oscilloscope, the plaintext(s) file(s) used, the ciphertext(s) output and the key(s) used to cipher it. The capture script stores these files in the following format: the traces have the Matlab format, and the remaining files have txt extension. Each of these files is stored in a specific folder: trace files are stored in “traces”, plaintexts in “input”, ciphertexts in “output” and keys in “keys”. The respective filename also contains its type (e.g. output_Random_100_-traces_part_1.txt).

It is our goal to generate a ChipWhisperer project from this information, and to achieve this, it is used the module described in 4.2.1. As input the referred module receives the location of the previous files. When the analysis process comes from the capture process (i.e. “Run script after full capture” is checked), the files location is obtained by getting the folders which contains the timestamp registered and the traces are stored in a queue. Each of these folders is used as argument to a method (e.g. get_-file_plaintexts_txt(path);get_file_traces_mat(path), etc) which returns the list of files of that type. These files will be used as argument to **convert_files_to_cw**, which returns the location of tracefile.npy and plaintexts.npy produced - the ChipWhisperer format.

These files, along with the trace and sample ranges and bytes checked will be used in CPA script, which will do compute CPA for the specified bytes, over the traces and sample range intended. As “Capture”, this information will be stored in a log file. In order to avoid the cost of the process of file conversion if two consequent analysis are done, the last analysis is saved, registering its path, the tracefile and the plaintexts file. If this information is the same, only the CPA script will run. In case of any of these files or folder is missing, numpy files will be produced again.

4.1.7 Settings

In order to configure the plotter and some application settings, there is a window when this customization is defined. It is on “Edit - Settings...” menu and it allows to set the plotter limits, define its zero position (i.e. the plotter current position will be the position 0), the units in which the plotter will work, the number of motor steps which will be equal to 1mm, and define if the plotter motor axis will be inverted or not. This window is shown in Figure 4.8.

It is important to have plotter limits defined because it prevents the plotter from force a movement where it reaches its physical limits. When enabled in area number 3 of figure 4.1, a green contour representing the limits is displayed on map, and every movement of plotter is verified before the plotter proceeds with the command. An alternative to this software verification, will be the use of switch triggers

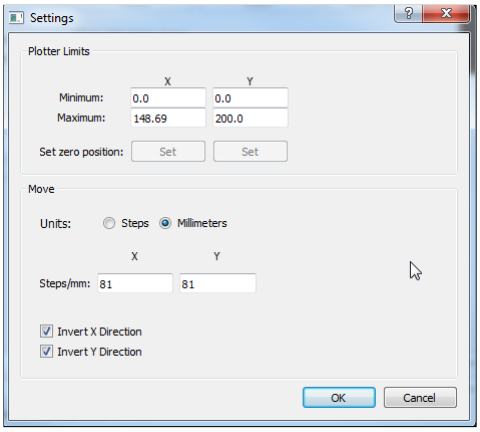

Figure 4.8: XY Plotter settings.

near the end of travel of each axis. This option is available starting from version 0.9 [2] of GRBL, however the 11 and 12 pinouts (as depicted in figure 4.9) are switched in this version and do not match with the plotter controller.

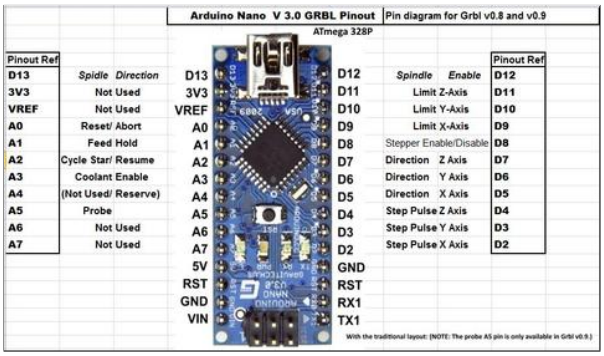

Figure 4.9: GRBL Pin Diagram 0.8.

The plotter is provided from Eleks Maker, which uses Eleks Mana SE, a customized Arduino Nano version, in which the motors are connected with.

The plotter settings (units, steps, maximum and minimum limits, and inverted positions) are attributes from Settings object, which are used to register the new configurations that will be applied.

Listing 4.1: “Settings dialog box.”

```
    def change_parameters(self):
        unit_selected = self.ui.units_buttonGroup.checkedId()
        try:
            self.plotter.settings.change_x_steps(self.ui.x_steps.text())
            self.plotter.settings.change_y_steps(self.ui.y_steps.text())
            if unit_selected:
                self.plotter.settings.unit_mm = True
            else:
                self.plotter.settings.unit_mm = False

```

[2] https://github.com/grbl/grbl/wiki/Configuring-Grbl-v0.9

```
            self.plotter.settings.set_plotter_limits(self.ui.limits_min_x.text(), self.ui.
                limits_max_x.text(), self.ui.limits_min_y.text(), self.ui.limits_max_y.text())

            self.plotter.settings.set_inv(self.ui.checkbox_inv_x.isChecked(), self.ui.
                checkbox_inv_y.isChecked())
            self.plotter.apply_settings(self.plotter.settings, True)
            ...
```

The previous code is triggered when “Ok” button is pressed on Settings dialog box. When changes are made in UI, the respective settings are changed in Settings. The last step is apply the settings to the plotter, by providing a new Settings object to the Plotter object. If no errors occur, the Plotter object will replace their values by the ones from Settings object.

This independence allows to use files to save and load the plotter configuration, without the need to define over again the parameters stated early. This is achieved using the PyYAML Framework [36] which allows to dump a Python object to a YAML [37] document and produce a Python object recurring to it. YAML is a way of serialize data which is readable by humans and its use is adequate to lists, hashes or simple variables. When the user saves the plotter’s configuration, a YAML document is produced with the following structure:

Listing 4.2: “YAML document produced using the Settings object.”

```
!!python/object:settings.Settings
_max_x: null
_max_y: null
_min_x: null
_min_y: null
_plotter_limit_max_x: 148.69
_plotter_limit_max_y: 200.0
_plotter_limit_min_x: 0.0
_plotter_limit_min_y: 0.0
_x_steps: 81
_y_steps: 81
inv_pos:
  x: true
  y: true
unit_mm: true
```

The first line represents the data type to reconstruct when the data is deserialized. In this case it is a Python object of type Settings. The next four lines represent the values for an region of interest for capture purposes. Next, the maximum and minimum limits for plotter traveling are also registered as well the steps for each axis. Lastly, it is registered if the axis should be inverted and which units are in use. With this information a YAML document with the .cfg extension is produced. This file also allows its modification outside the application, which can be useful if multiple settings must be used, reducing the time cost of this operation in the software.

When imported, a Settings object is created from the YAML document. The values from the parameters are verified and if no error occur, the previous Settings instance will be replaced, applying the values to the Plotter object.

4.2 Chipwhisperer Analyzer Expansion

As stated before, ChipWhisperer Analyzer is a software intended to work in SCA. In the following subsections it is described the expansion in order to support Matlab, and a less memory-intensive CPA version. It is also described two modules which convert the traces from Matlab to Chipwhisperer and vice-versa.

4.2.1 Matlab to Chipwhisperer Project

The goal of this tool is to convert in a easy way the traces recorded in Matlab to the ChipWhisperer format, by converting the files to Numpy and generating a ChipWhisperer format configuration. When captures are collected, by using the script used at INESC-ID, they have the structure described in 4.1.6. In order to be possible to analyze the captures in ChipWhisperer and do not break the compatibility with other scripts that make use of the same structure, this module needed to be developed.

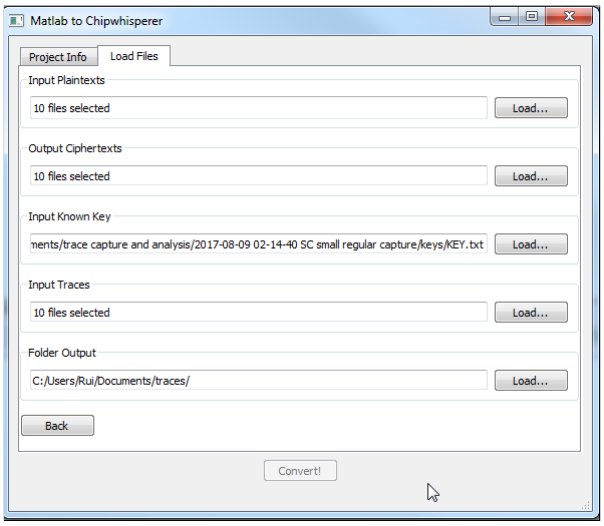

Figure 4.10: Matlab to Chipwhisperer GUI.

Listing 4.3: “CWP Project File.”

```
[Trace Management]
tracefile0 = EM_part1_data/traces/config_2018.09.09-22.02.09_.cfg
enabled0 = True
[ChipWhisperer]
[[General Settings]]
Project Name = EM_part1
Program Name = ChipWhisperer
Project File Version = 1.00
Project Author = INESC-ID
Program Version = V3.5.4
```

A ChipWhisperer Project has the following structure:

- A .cwp file containing the properties of the project and the location of the .cfg file.

- A .cfg file with information regarding the traces (number of traces, number of samples, and prefix).
- One .npy file for each of the following objects: keylist, knownkey, textin, textout, traces.

The prefix used in .cfg must match with the one used in .npy files and should be in the same folder.

With this module it is possible to produce .npy files, along with the configuration files regarding the project from multiple .txt (plaintexts,ciphertexts,keylist) and .mat files (traces).

The GUI depicted in figure 4.10 has 2 tabs. The first one contains information regarding the project, such as project name, project author, number of traces and number of files. The second allows the user to select the files to be used and the output folder.

The information regarding the project (Project Name, Project Author, Project File Version, Program Version) and traces are stored in ordered dictionaries, which allows to write this information to the files in the order required. The prefix used for the numpy and .cfg files is the timestamp of when the file is generated, in the format **yyyy.mm.dd-HH.MM.SS**.

The next step is to save the plaintexts, ciphertexts, keys and traces in numpy format. In traces this is done using the **loadmat** method from the scipy.io python library. For each file selected, the matlab keys are iterated, and a copy of the content of the variable with this information is appended to a list. Other keys contain information about the .mat file, such as version, header and globals. This information is not intended, and can be differentiated by starting with characters __. When all Matlab files selected are traversed, the traces are stacked vertically in a numpy vstack structure, allowing to be accessed in the format traces[trace_number].

For the processement of .txt files, the process is similar. The plaintexts, ciphertexts and keys are stored in hex format and each entry (i.e. trace number) has the previous structured splitted by newlines. Thus, for each type, each file is traversed, copying as decimal format each line and storing the results in a list. Once done, the information will also be written into vstack arrays, which also allows the access in the format above.

When each type is handled, the last step is to save the vstack array into .npy. This is done by using the numpy save method, providing the selected output folder chosen early, along with the timestamp from the .cfg file and the type of data (textin, textout, knownkey or keylist). This information is crucial because it is the way the ChipWhisperer identifies each element.

This tool was integrated in ChipWhisperer Analyzer, and can be found in “Tools - Matlab to Chipwhisperer Converter”.

4.2.2 NPY to Matlab

This module is a simple Python script which provides a way to convert .npy files to a Matlab file. In its original version [3] the .npy files are placed inside an “input” folder. By iterating over each file, the structured is saved recurring to the scipy.io savemat native library, and using a dictionary structure where

[3] https://github.com/ruitome/npy_to_matlab

the key is the name of the .npy file, and consequently the variable of the .mat file. In ChipWhisperer this implementation is used to provide Matlab support, as described in 4.2.3, and there is a slightly difference: instead of using a folder to place the files to be converted, the list of files is passed as argument to npy_-to_matlab, and once the process is completed, the .mat file location is returned.

4.2.3 Matlab Support on ChipWhisperer Analyzer

The Matlab Support on ChipWhisperer provides the ability to use custom power models in Matlab, with the minimum changes in ChipWhisperer's source code. This is achieved by using the Matlab Engine for Python [38], which provides an API to call Matlab functions.

To use this feature, the engine library must be installed in the same Python version as ChipWhisperer is running, whose process is described in Matlab documentation, or else this module won't load.

The script attacks are loaded when the CPA class is loaded by getting the scripts which are in **chipwhisperer.analyzer.attacks.cpa_algorithms**. These scripts inherit from **AlgorithmsBase** and the method **addTraces(self, traceSource, tracerange, progressBar=None, pointRange=None, tracesLoop=None)** must be implemented. This method will also be the interface from which it is possible to get the traces, trace range, and sample range.

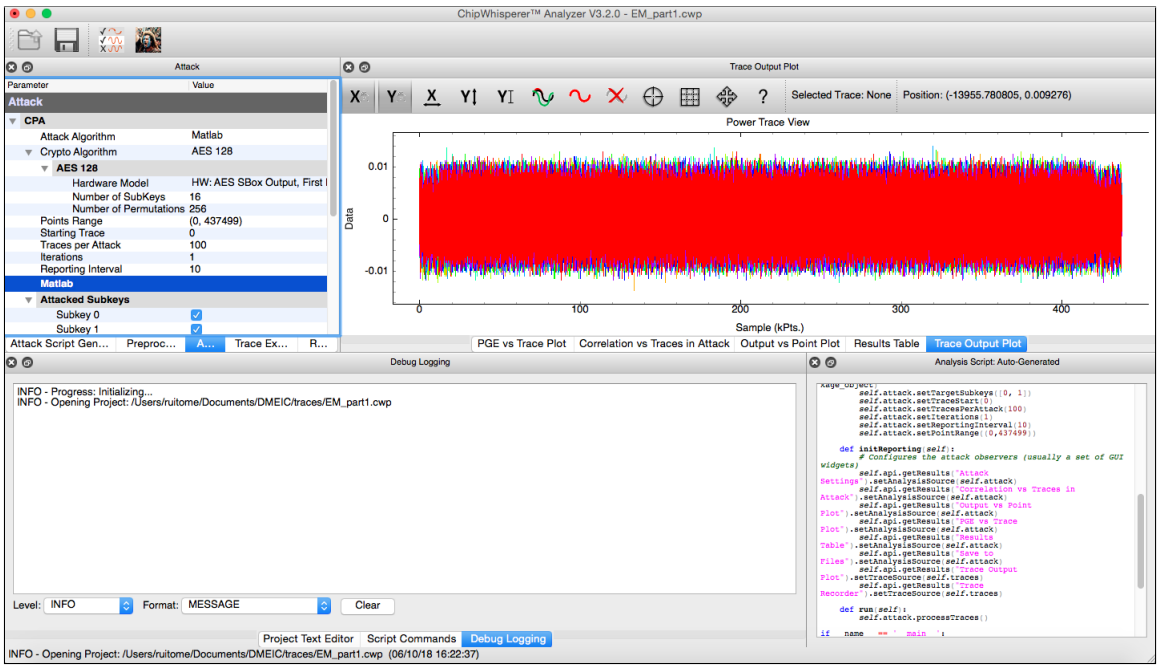

Figure 4.11: Matlab Support on ChipWhisperer Analyzer.

The Matlab support is achieved recurring to the NPY to Matlab module (4.2.2) by writing traces, plaintexts, and keys to a .mat file, and calling the Matlab function with the file location and parameters (trace/samples range and bytes) in analysis (depicted in Figure 4.11).

Another alternative instead of using a .mat file will be to pass the information stored in memory directly to the Matlab function, however the Matlab API needs to convert the data to a Matlab format.

This process is memory consuming, because two instances of the same object must remain in memory, and is extremely slow as shown in the Evaluation section.

The Matlab function has the following declaration:

- **function CC = cpabload(filename, tracerange, pointsrange, byte_list)**

As noticed, the Matlab function receives the path of the .mat file, which has the traces, plaintexts and keys, the trace range, the sample sange and the byte in analysis, and computes the correlation between the Power Hypothesis and the Traces, returning this result to ChipWhisperer. In order to provide feedback to ChipWhisperer (Updating the labels and progress bar) in which byte the correlation is at, only a single byte is computed each time.

For each byte the correlation is computed, and the result is updated in ChipWhisperer stats, which will build the statistics to the Results Table.

- data = correlation_matlab.correlation(self.eng, (0, numtraces-1), (0, points-1), [bnum], traceSource.getMatfile())
- self.stats.updateSubkey(bnum, data, tnum=tracerange[1])

It can be observed above that the trace range and sample range starting from is always 0, and the maximum range and maximum points are the numbers of sample and points respectively. This is done because trace range and sample range are already applied to the data received.

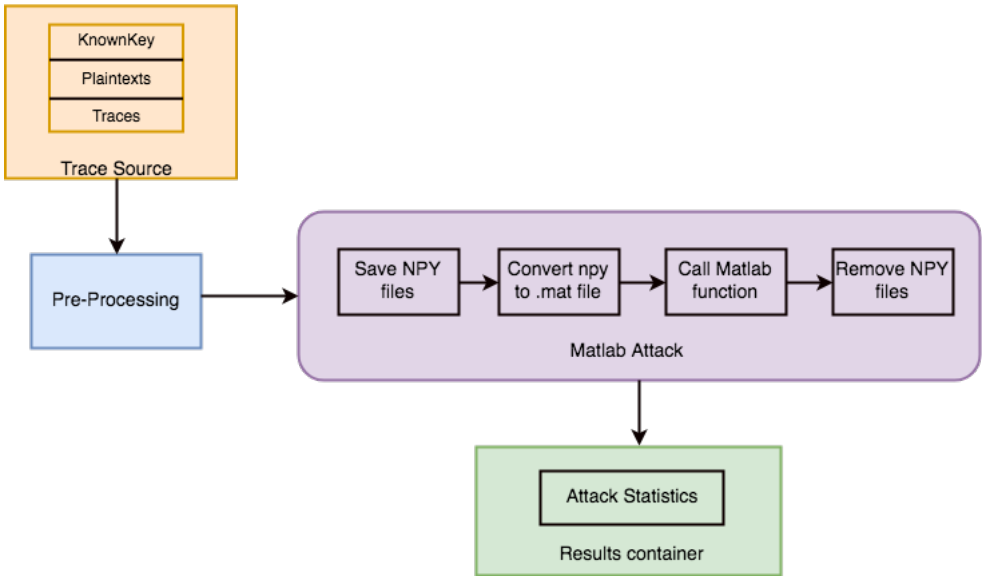

Figure 4.12: Matlab Attack Process.

The Figure 4.12 sums up the Matlab Attack Process. Plaintexts, traces and keys (data) are provided from TraceSource. If pre-processing modules are enabled, they will operate over that data. In order to convert the information to Matlab, the next step is to get the data from the TraceSource, and

create numpy files with this information. The filepath from the .mat produced is then stored in the TraceSource object, allowing to access this information to further purposes. This was done by implementing two methods (**getMatfile** and **setMatfile**) on **PreprocessingBase** class (which inherits from all preprocessing modules), and to **TraceManager** class, which inherits from TraceSource, and it is used when no processing modules are used. Once converted, the Matlab function is called and once finished the results are returned to ChipWhisperer, which will handle the data. The .mat file produced contains data after the preprocessing module is applied, allowing its use outside of ChipWhisperer. The last step is to remove the numpy files created previously.

4.2.4 A less memory-intensive version of CPA script

ChipWhisperer has two methods of computing CPA: recurring to a progressive mode, which stores values at a given report interval necessary to build some statistics about the traces, or a simple CPA script which does not make use of libraries optimized for these operations. Thus, the goal of this module is to allow analysis where a progressive mode is not intended, due to it's memory consuming, and a implementation faster than simple CPA.

This algorithm has the same structure as the analysis script used in XYPlotter - Analysis (4.1.6), however only the correlation is computed and due to the reasons stated above, only a single byte is computed each time. In XYPlotter this file accepts a list of bytes and the top 5 keys (the keys with higher correlation values) are displayed for each byte.

The file contains a simple implementation of Pearson Correlation in Numpy, and its implementation can be found below, along with the Pearson Correlation:

$$r_{i,j} = \frac{\sum_{d=1}^{D}[(h_{d,i} - \bar{h_i})(t_{d,j} - \bar{t_j})]}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \bar{h_i})^2 \sum_{d=1}^{D}(t_{d,j} - \bar{t_j})^2}} \tag{4.1}$$

Listing 4.4: "Numpy Pearson Correlation implementation"

```
def corr(hypo,traces):

    xr, xc = np.shape(hypo)
    yr, yc = np.shape(traces)
    mean_x = np.mean(hypo, axis=0, dtype=np.float64)
    mean_y = np.mean(traces,axis=0,dtype=np.float64)
    x = hypo - np.tile(mean_x,(xr,1))
    y = traces - np.tile(mean_y,(yr,1))
    hyp = np.dot(np.transpose(x),y)

    sqrt_x = np.sqrt(np.sum(np.multiply(x,x),axis=0))
    hyp = np.divide(hyp, np.transpose(np.tile(np.transpose(sqrt_x),(yc,1))))

    sqrt_y = np.sqrt(np.sum(np.multiply(y,y),axis=0))
    hyp = np.divide(hyp, np.tile(np.transpose(sqrt_y),(xc,1)))
```

```
    return hyp
```

In order to take advantage of this implementation, a new attack module, named CPAMod, was added to **chipwhisperer.analyzer.attacks.cpa_algorithms**. The process is similar to the Matlab Support, however in this process no numpy or .mat files are created, and the data from TraceSource is used as argument without any conversions, allowing to reduce the memory consumption and saving time in disk writings. This module is not intended to replace the Progressive attack implemented, so detailed analysis as "Correlation vs Traces in attack", among others, will not be available. Instead its purpose is to provide a analysis where this information is not intended, providing a faster alternative to get the keys for the trace range in analysis.

Summary

In this chapter we have been through the design and implementation of the XYPlotter software. This tool allows to combine the process of collecting traces from a device under test with the control of a XY Plotter CNC machine, by mapping a region of interest and collecting the data in each point of the given area. Then, it is either possible to run an analysis script inside XYPlotter or save the data gathered to analyze in other platforms. An overview of the XYPlotter was described, followed by the description of each component developed.

In ChipWhisperer Analyzer Expansion, two tools to convert data between different formats were developed: Matlab to ChipWhisperer Project and, Npy to Matlab. The two last sections describe how the Matlab support on ChipWhisperer was achieved, and how the correlation function was implemented as alternative to other algorithms already implemented in the platform.

5 Evaluation

This evaluation covers the implementation of Matlab support and the new CPAMod component integrated in ChipWhisperer Analyzer. Different number of samples were considered in the process, depending on the number of traces and the respective approach used. The tests were performed on a virtual machine with the following characteristics allocated: AMD Ryzen 1700 with 8 CPU's (running at 3.00Ghz), 28GB of RAM DDR4 and running Windows 7 x64. It was used an Samsung Evo 860 Solid State Drive (SSD) to store the data, with a sequential read of 550MB/s and a sequential write of 520MB/s. In both modules it was evaluated the time taken to compute the Pearson Correlation algorithm and the memory consumed. Two approachs were used when considering the use of Matlab in Python: Pass the traces and plaintexts in memory as arguments to Matlab function's vs write data to a Matlab file and pass the respective filepath as argument. The tests were taken using the cProfile and memory_-profile Python profilers recovering the first 10 bytes and the 16 Bytes of the full AES 128 key and using Matlab R2016a. Then, four attack implementations of ChipWhisperer are compared for the same group of traces, comparing the time taken for each approach. Two groups of samples (traces x number of points) were evaluated from captures previously collected:

- 1k x 100
- 100 x 1k
- 10 x 10k
- 100k x 10k
- 100k x 1k
- 10k x 10k

The traces, plaintexts, ciphertexts, and keylist are stored in the Numpy format, which allows the use of Python scientific libraries optimized for this data.

5.1 Call Matlab from Python - Memory vs File vs Only Python

Two approaches were considered when using Python to call a Matlab function: Pass the traces and plaintexts in memory as arguments to Matlab function's vs write data to a Matlab file and pass the respective filepath as argument. In the first approach, Matlab requires that the arguments should be converted to a Matlab type, in this case, **matlab.double**. This type only accepts lists as arguments, so the traces and plaintexts must also be converted to lists before using them. Once this process is complete, it is possible to call the Matlab function that will perform the Pearson's Correlation computation. The following results were taken outside ChipWhisperer, using the Python script provided in the platform and calling the Matlab function. The Python alternative approach used in analysis in XYPlotter, and integrated in ChipWhisperer, was also added to the comparison.

50 samples of each of the following captures were analysed, and the average of the results was computed:

- 1k x 100 (578 KB)
- 100 x 1k (409 KB)
- 10 x 10k (393 KB)

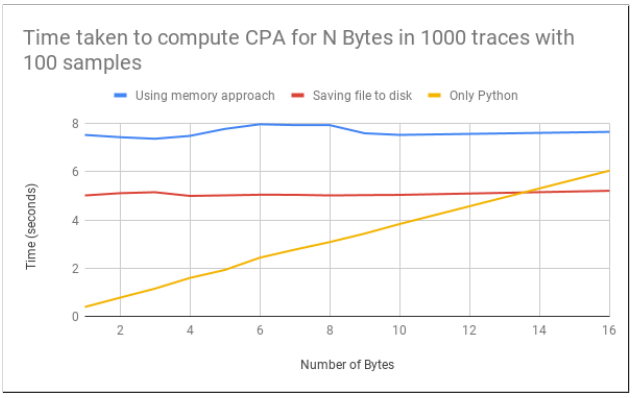

Figure 5.1: Time taken to compute CPA for N Bytes in 1k traces with 100 samples.

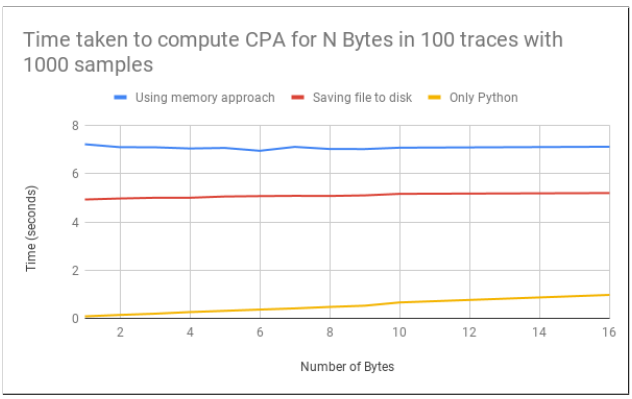

Figure 5.2: Time taken to compute CPA for N Bytes in 100 traces with 1k samples.

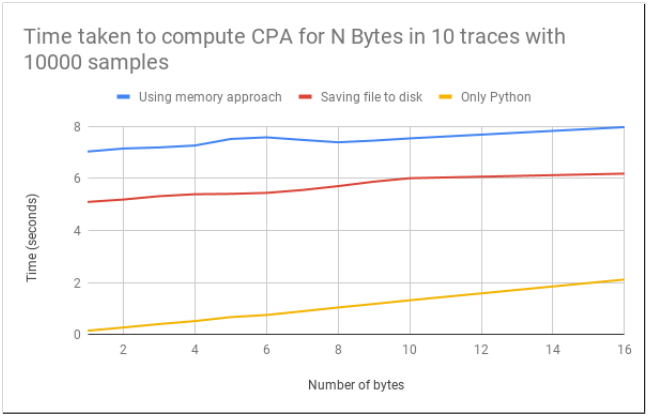

Figure 5.3: Time taken to compute CPA for N Bytes in 10 traces with 10k samples.

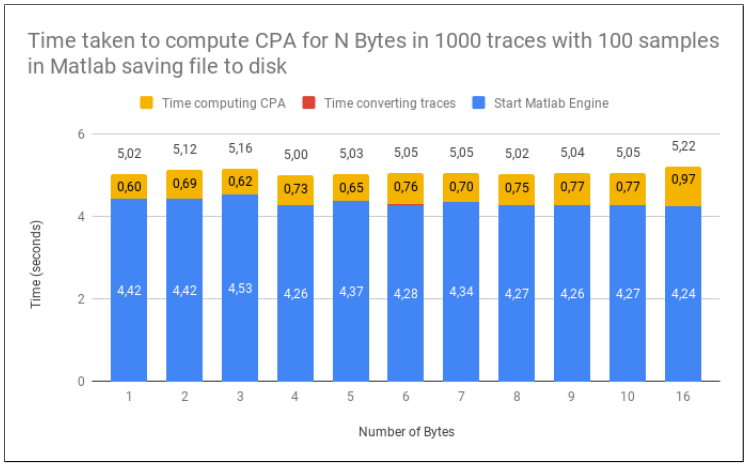

Figure 5.4: Time taken to compute CPA for N Bytes in 1k traces with 100 samples in Matlab.

Figures 5.1, 5.2 and 5.3 show the average time taken to compute CPA in this group of samples, comparing the time when downsizing the number of traces and increasing the number of samples in study. This group has a reduced number of traces/samples and can be compared to a power analysis attack on a smartcard without counter-measures implemented. As noticed, the time taken when passing the traces and plaintexts in memory from Python to Matlab is higher than other options, evidencing the conversion of the traces to the matlab.double type. On the other hand, it can be noticed that there are cases where it can be useful to choose saving a Matlab file to disk instead of Python exclusive approach. When comparing these three captures, it can be noticed that using this method is ideal for captures above 1000 traces and a number of bytes $\geq$13. However, for this group of traces, the figure 5.4 shows that this approach has a cost (in time) of initialization of the Matlab engine that is higher than the cost of converting the traces ($<$ 0.05s) and computing CPA. The Matlab engine in this approach is used when calling the module to load the file converted. For the remaining captures the behaviour is identical, as the time of initilization is independent of the capture used. As expected, the approach that only uses memory to compute CPA has the highest cost in time and memory.

- 100k x 10k (3,74 GB)
- 100k x 1k (399 MB)

- 10k x 10k (383 MB)

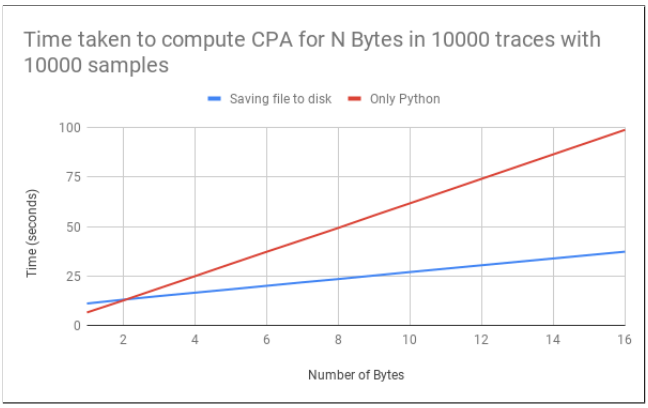

Figure 5.5: Time taken to compute CPA for N Bytes in 10k traces with 10k samples.

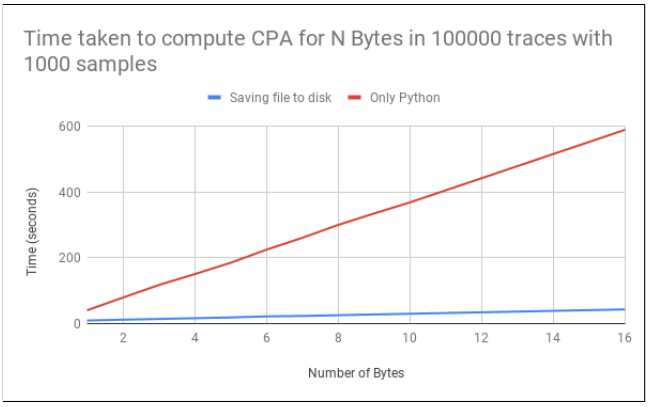

Figure 5.6: Time taken to compute CPA for N Bytes in 100k traces with 1k samples.

In this group of traces it is even more noticeable the time differences between the Python exclusive approach and the usage of a Matlab file to read data from. Figures 5.5 and 5.6 confirm that the usage of the Matlab file reduces significantly the time of computing CPA. It is important to refer that for this group the method of using all the data in memory was not considered because it is much slower and consumes much more resources that is out of scale for any comparison with the other approaches. Additionally, for 100k traces x 10k samples (Figure 5.7), the exclusive python approach has a cost much higher than the alternative approach, so only the first 5 bytes were computed. In both methods the time consumption is nearly linear, i.e. any additional byte we want to compute CPA, the cost increases the time of 1 byte only. However, as stated before, in the approach of saving file to disk there's time involved in starting the Matlab engine and converting traces (which is not noticeable for small group of traces/samples) that is nearly constant. By deducting that time we can notice linearity. This is observable in Figure 5.8.

To profile the memory used during the CPA computation it was used the memory profiler module included in Python distribution and the profiler present in Matlab. In these results it is defined the term memory used as the peak memory during the process execution - i.e., the maximum amount of allocated memory during the function execution. For the approach which makes use of Matlab to compute the CPA by providing the Matlab file, these results do not include the memory used in the conversion process

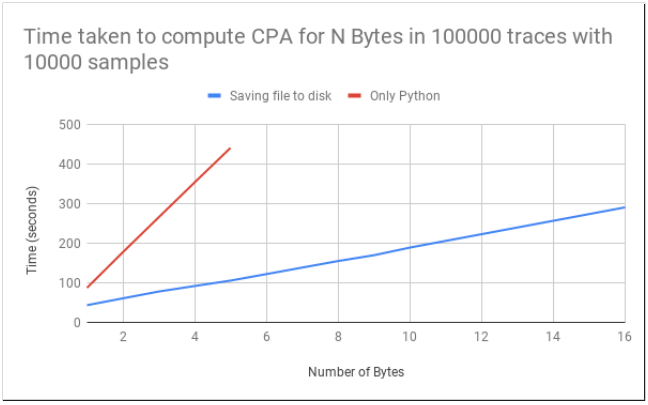

Figure 5.7: Time taken to compute CPA for N Bytes in 100k traces with 10k samples.

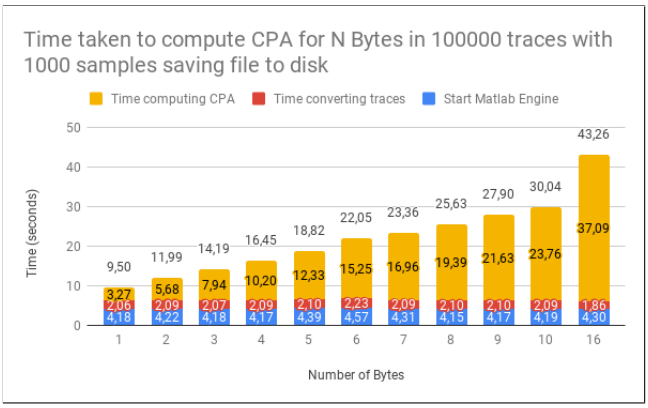

Figure 5.8: Time taken to compute CPA for N Bytes in 100k traces with 1k samples saving file to disk.

from the numpy file format to the Matlab format, which will always be the same size of the file given as input. Once that process, the Matlab file will be used during the call to Matlab code.

- 100k x 10k (3,74 GB)
- 100k x 1k (399 MB)
- 10k x 10k (383 MB)

Figures 5.9, 5.10 and 5.11 depict the memory required to compute CPA using Python by saving the file to disk and calling Matlab to do computation. During the evaluation two analysis were made: The first one was to measure the consume of memory during the execution of the Python code. The second was measuring the memory used during the CPA computation in Matlab. Thus, the last one will give us an overview of the memory used in each function. It is important to state that these results were obtained directly from the profiler report included in Matlab and correspond to the most relevant functions related with CPA computation: load file, computation of power hypothesis, and computation of Pearson's correlation. The remaining functions and variables, and respective child function calls are refered as "Other lines" and are also included in the figures.

As stated before, the memory consumed corresponds to the peak memory of the process, thus, the memory in use is independent (has very little contribution) of the number of bytes in analysis. As

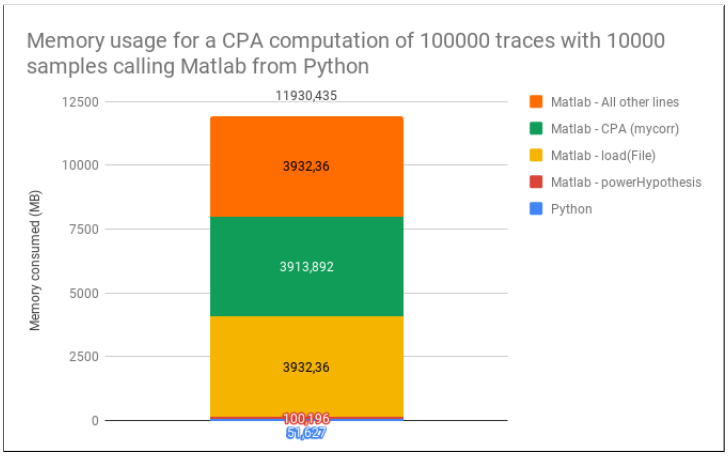

Figure 5.9: Memory consumed during the computation of CPA in 100k traces with 10k samples saving file to disk.

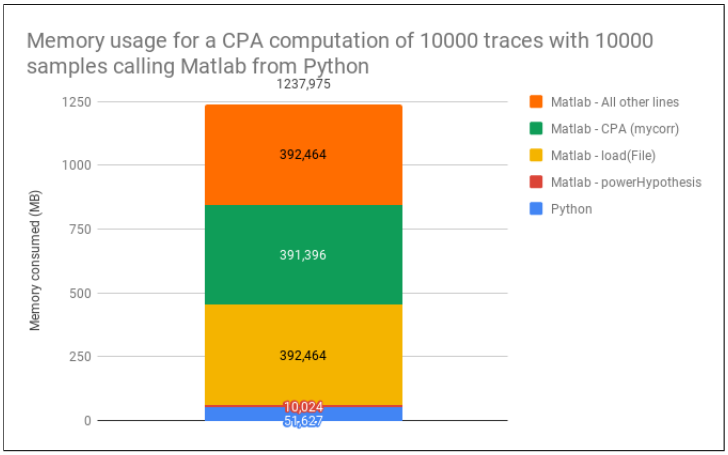

Figure 5.10: Memory consumed during the computation of CPA in 10k traces with 10k samples saving file to disk.

observable, the memory usage of the Python file is constant. After traces conversion, the traces in Matlab format will be used in the call of the Matlab function (which will perform the CPA). The traces conversion to Matlab format is made by loading the numpy file in memory and write the same data to the Matlab format. As previously mentioned, this process is not included in the benchmarks, because the conversion occurs before any Matlab call, and it is intended to evidence the maximum amount of memory (i.e. the memory required) consumed during all the process.

In section 4.2.4, it was presented the CPA code implementation in list 4.4 used in the approach of using only Python. In table 5.2 it is shown a comparison of the memory consumption for the first code block (which consumes higher memory). The variables of the remaining code blocks that have a cost less than 1 MB are not present in the table. A detailed example for the group of 10k traces with 10k samples can be found in list 6.1 of Appendix. The table rows represent the code variables and the columns represent the trace group in analysis. The last row represents the maximum memory consumed in the process.

The ‘traces’ variable contains the loaded traces from the .npy file. ‘xr’ and ‘yr’ represent the number

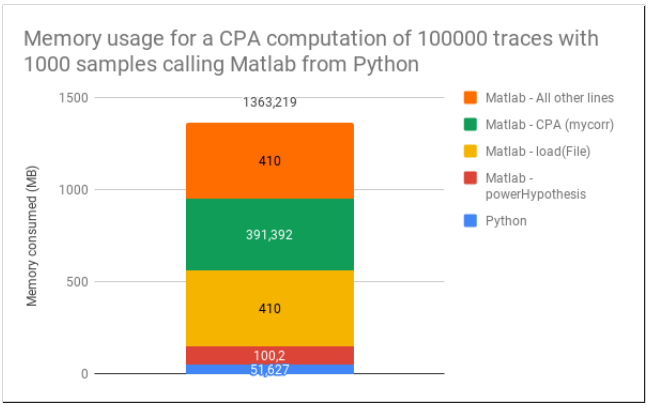

Figure 5.11: Memory consumed during the computation of CPA in 100k traces with 1k samples saving file to disk.

of traces. The remaining variables are part of the Pearson's Correlation formula present in equation 2.2, and have the following association:

Variable	Formula representation
mean_x	h_i
mean_y	$\bar{t_j}$
x = hypo - np.title(mean_x,(xr,1))	$\sum_{d=1}^{D}(h_{d,i} - \bar{h_i})$
y = traces - np.tile(mean_y,(yr,1))	$\sum_{d=1}^{D}(t_{d,j} - \bar{t_j})$
hyp = np.dot(np.transpose(x),y)	$\sum_{d=1}^{D}[(h_{d,i} - \bar{h_i})(t_{d,j} - \bar{t_j})]$

Table 5.1: Variable representation in Pearson's Correlation formula.

Variable (MB) \Trace (traces_samples)	10_10k	100_1k	1k_100	10k_10k	100k_1k	100k_10k
traces	0,477	0,453	0,473	382,312	382,344	3822,273
mean_x	0,027	0,023	0,027	0,02	0,02	0,02
mean_y	0,141	0,066	0,066	0,07	0,004	0,004
x = hypo - np.title(mean_x,(xr,1))	0,051	0,34	1,98	19,594	195,715	195,719
y = traces - np.tile(mean_y,(yr,1))	1,523	1,594	1,512	764,496	764,445	7644,367
hyp = np.dot(np.transpose(x),y)	21,906	6,098	2,973	27,324	6,719	27,387
Total	71,027	37,926	36,973	1261,32	1584,375	11922,086

Table 5.2: Memory consumption of variables with higher costs in correlation computation and total of memory consumed by the program.

The results are the expected according to the code. Only the number of traces has influence in the summation represented by **x** because the hypothesis generated only use the number of traces, while both number of traces and number of points have impact in **y** variable, which will be reflected in the memory consumed by the algorithm.

5.2 ChipWhisperer Integration

In this section it is compared the time of processing correlation power analysis in ChipWhisperer Analyzer between the methods described in 4.2.3 and the ones provided by ChipWhisperer for the same group of traces of the previous section.

As default, ChipWhisperer provides two methods to compute CPA: Simple and Progressive. The Simple method is a simple approach to compute CPA. When comparing to Progressive mode, it uses less memory because it provides feedback of the actual attack status and it stores information about the attack that can be used to view other information about the attack progression.

For the evaluation 10 attacks were made for the following group of traces, recovering the first 10 Bytes and the 16 Bytes of the full AES-128 key, and the average of the results was computed:

- Number of traces x number of samples
- 100k x 10k
- 100k x 1000
- 10k x 10k
- 1000 x 100
- 100 x 1000
- 10 x 10k

The processing time using Matlab is expected to be higher than the results in the previous section. This increase is caused by the traces conversion to Matlab file, which needs to be created every attack as the pre-processing modules may have been applied to the traces. When the conversion is complete, the CPA computation will be computed over that data.

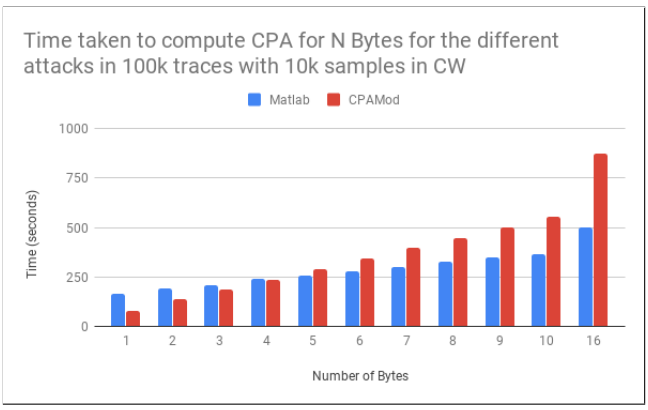

Figure 5.12: Time taken to compute CPA for N Bytes in 100k traces with 10k samples in ChipWhisperer.

As previously stated, the use of Matlab to compute CPA has better results where multiple bytes are in analysis, taking advantage of the initialization cost of the Matlab engine. In figures 5.12, 5.13 and 5.14

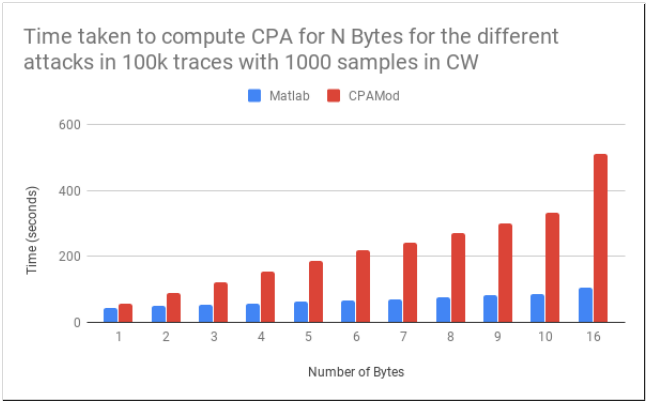

Figure 5.13: Time taken to compute CPA for N Bytes in 100k traces with 1k samples in ChipWhisperer.

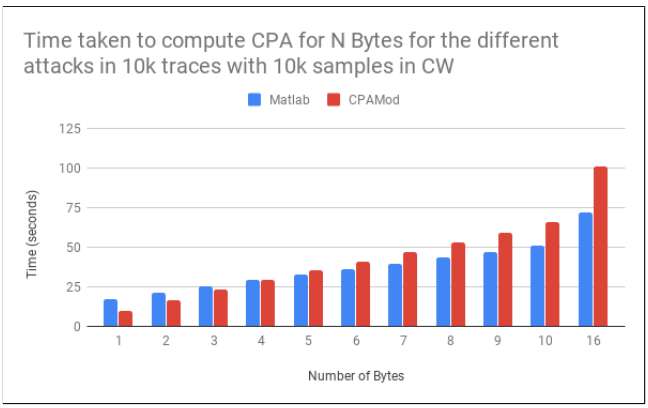

Figure 5.14: Time taken to compute CPA for N Bytes in 10k traces with 10k samples in ChipWhisperer.

it is observable that CPAMod attack in ChipWhisperer has performance issues with high number of traces. The advantage of Matlab is noticeable, where optimizations in Matlab platform are made to work with these sample data. Despite of the use of numpy Python library in CPAMod, Matlab has advantages for these cases. In these figures, the attacks Simple and Progressive are very slow, with results out of scale, so for this reason they were omitted.

In both figures 5.15 and 5.16 it is noticeable that the Progressive mode is a method that is slower than the Simple method, however, when completed, it provides information about the attack progression. CPAMod is faster in these group of traces because it only uses Python, as we are analyzing a reduced number of traces/samples and it is faster than initialize the Matlab engine, as opposed to data where exists a large number of traces. Figure 5.17 denotes that the Progressive mode performance is highly tied with the number of traces in analysis, getting better results when this number is lower.

As CPAMod uses Python to compute CPA, does not have a performance degradation when implemented in ChipWhisperer whilst Matlab does due to the conversion cost. However the use of Matlab allows to compute a large number of traces in a reduced time. On the other hand, this support also improves the performance to compute CPA when the researcher is interested in analyze a large number of bytes, even for few traces.

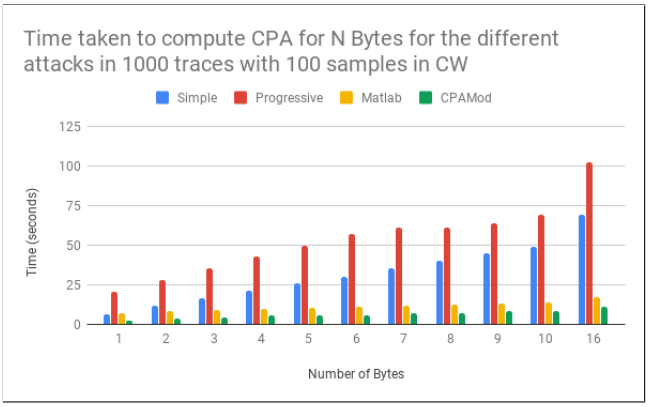

Figure 5.15: Time taken to compute CPA for N Bytes in 1k traces with 100 samples in ChipWhisperer.

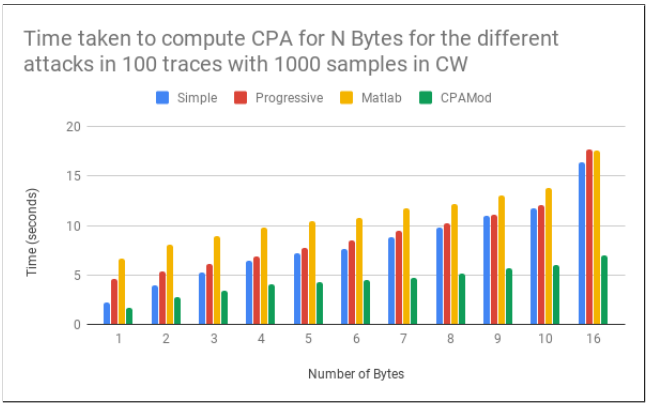

Figure 5.16: Time taken to compute CPA for N Bytes in 100 traces with 1k samples in ChipWhisperer.

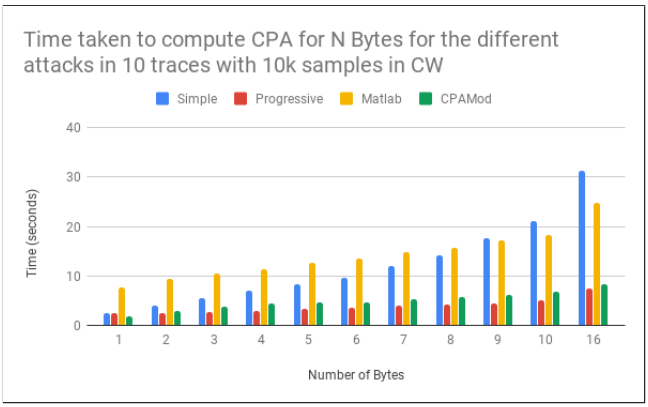

Figure 5.17: Time taken to compute CPA for N Bytes in 10 traces with 10k samples in ChipWhisperer.

Summary

In this chapter we introduced the experimental evaluation made to ChipWhisperer Analyzer Expansion and its results. It is compared, in memory and time, the cost of call Matlab functions from Python code, using Numpy data in memory, versus writing this data to Matlab files, versus use only Python to compute the same correlation function.

After the integration in ChipWhisperer Analyzer of Matlab support and a customized CPA Python script, the timing cost of calling each algorithm for N bytes was compared for the algorithms in the platform. We conclude that both the implementations have faster results than the algorithms provided originally in ChipWhisperer Analyzer. When comparing a reduced group of traces, the Python exclusive approach is generally faster than the other algorithms implemented. When comparing to other approaches, the fact of Matlab has performance optimizations makes the CPA analysis faster when working with very large samples, improving the time taken on ChipWhisperer to compute these datasets. However, when working with Numpy traces, a conversion process must occur before working with these data, consuming time, memory and disk space. The process of the Matlab engine initialization occurs always when the Matlab algorithm is called in ChipWhisperer Analyzer, contributing to an additional time in the process and taking longer than other algorithms to compute CPA with small group of traces. The next chapter finishes this thesis by presenting the conclusions regarding the work developed and also introduces some directions in terms of future work.

6 Conclusions

6.1 Conclusions

In this project, a set of tools were developed to allow efficient side channel attack evaluation of cryptographic devices. XYPlotter was developed with the purpose of integrating side channel analysis features with the control of a XY positioning table, allowing to collect electromagnetic traces over a given cryptographic chip area and also allow runs analysis algorithms on it. This analysis can be used to find where higher leaks occur, i.e., the leakage spatial distribution.

Support for both Matlab and conversion modules was added to the ChipWhisperer platform, allowing to compute CPA in a large number of traces and allowing researchers to test different correlation techniques whilst maintaining compatibility with existing environments. An optimized version of Pearson's Correlation was added to ChipWhisperer, name CPAMod, allowing it to have a faster alternative than existing algorithms when few traces are being analyzed.

Timing and memory tests were performed in different Python calling Matlab code approaches, passing traces and plaintexts in memory to Matlab's function vs write the data to file and pass the respective filepath, vs a customized CPA implementation in Python, showing the best option for the trace size in analysis.

To conclude, the requirements and objectives set for this Thesis were achieved: a user friendly SCA tool was developed, allowing to continue the work in SCA research. In order to improve the use of one of the most complete software in SCA new tools were also developed, providing compatibility with the past and future work to be developed.

6.2 Future Work

One of the biggest pleasures along of this project is its flexibility and the different features that can be implemented, having plenty of room for improvements. The software XYPlotter started from zero and today is one of the softwares used to work with the plotter at INESC-ID. As future work, it would be interesting the visualization of the traces in XYPlotter, support for displaying locations with higher information leaks, support to choose which CPA script to use and its parameters in the GUI. In order to improve the processing time of CPA in XYPlotter, and support multiple features provided by the toolbox, the support of Jlsca would be another interesting path. As an extension to ChipWhisperer Analyzer it would be interesting to support different attack profiles on it. This would allow to associate an attack with a certain configuration with the most suited analysis algorithm. Another interesting direction would be

the use of deep learning in trace analysis, allowing to have a success rate of key recovery higher when the researcher has few traces available or there are protection mechanisms on the device under test.

Bibliography

1. Stefen Mangard, Elisabeth Oswald, and Thomas Popp. *Power Analysis Attacks - Revealing the Secrets of Smart Cards*. Springer US, 2007.

2. Paul Kocher, Joshua Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011.

3. Fred De Beer, Marc Witteman, Bartek Gedrojc, and Yijun Sheng. Practical Electro-Magnetic Analysis. *Non-Invasive Attack Testing Workshop NIAT-2011*, pages 1–18, 2011.

4. Fabien A P Petitcolas. *Encyclopedia of Cryptography and Security*, chapter Kerckhoffs, page 675. Springer US, Boston, MA, 2011.

5. N Fips. 197: Announcing the advanced encryption standard (AES). *Technology Laboratory, NIST*, 2001.

6. R L Rivest, A Shamir, and L Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

7. Thomas S Messerges, Ezzy A Dabbish, Robert H Sloan, Thomas S Messerges, and Robert H Sloan. Investigations of power analysis attacks on smartcards. *USENIX Workshop on Smartcard Technology*, 1999.

8. Joan Daemen and Vincent Rijmen. Resistance Against Implementation Attacks A Comparative Study of the AES Proposals. *Second AES Candidate Conference*, pages 343–348, 2000.

9. Eli Biham and Adi Shamir. Power Analysis of the Key Scheduling of the AES Candidates. *Proceedings of the second AES Candidate Conference*, 1999.

10. Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential Power Analysis. *Advances in Cryptology—CRYPTO'99. Springer Berlin Heidelberg*, pages 388–397, 1999.

11. Rambus. DPA Workstation Platform.

12. Riscure Inc. Inspector SCA, 2016.

13. Colin O'Flynn and Zhizhang Chen. ChipWhisperer: An Open-Source Platform for Hardware Embedded Security Research. *Constructive Side-Channel Analysis and Secure Design. Springer International Publishing*, pages 243–260, 2014.

14. UGS. Universal GCode Sender.

15. Benbox. Benbox.

16. H Handschuh, P Paillier, and J Stern. Probing attacks on tamper-resistant devices. *Cryptographic Hardware and Embedded Systems*, pages 303–315, 1999.

17. Jean-Jacques Quisquater and David Samyde. ElectroMagnetic Analysis (EMA): Measures and Counter-measures for Smart Cards. *Smart Card Programming and Security*, pages 200–210, 2001.

18. Karine Gandolfi, Christophe Mourtel, and Francis Olivier. Electromagnetic Analysis : Concrete Results. *Cryptographic Hardware and Embedded Systems — CHES 2001. Springer Berlin Heidelberg*, pages 251–261, 2001.

19. Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. *Cryptographic Hardware and Embedded Systems-CHES 2004. Springer Berlin Heidelberg*, 3156:16–29, 2004.

20. João Amaral, Francesco Regazzoni, Pedro Tomás, and Ricardo Chaves. Accelerating Differential Power Analysis on Heterogeneous Systems. *Proceedings of the 9th Workshop on Embedded Systems Security*, 2014.

21. M Joye, P Paillier, and B Schoenmakers. On second-order Differential Power Analysis. *Cryptographic Hardware and Embedded Systems–CHES 2005. Springer Berlin Heidelberg*, 3659:293–308, 2005.

22. Xavier Charvet and Herve Pelletier. Improving the DPA attack using Wavelet transform. *NIST Physical Security Testing Workshop*, 46:1–15, 2005.

23. Ruben A. Muijrers, Jasper G J Van Woudenberg, and Lejla Batina. RAM: Rapid alignment method. *Smart Card Research and Advanced Applications. Springer Berlin Heidelberg*, pages 266–282, 2011.

24. Sei Nagashima, Naofumi Homma, Yuichi Imai, Takafumi Aoki, and Akashi Satoh. DPA Using Phase-Based Waveform Matching against Random-Delay Countermeasure. *Circuits and Systems, 2007. ISCAS 2007. IEEE International Symposium on. IEEE*, pages 1807–1810, 2007.

25. Jasper G J Van Woudenberg, Marc F Witteman, and Bram Bakker. Improving Differential Power Analysis by Elastic Alignment. *Topics in Cryptology–CT-RSA 2011. Springer Berlin Heidelberg*, pages 104–119, 2011.

26. Secure-IC. Smart-SIC Analyzer.

27. DPA Contest. DPA Contest v3, 2016.

28. Ilya Kizhvatov and Cees-Bart Breunesse. JuliaSCA.

29. Thomas R Kramer, Frederick M Proctor, and Elena Messina. The NIST RS274NGC Interpreter -Version 3. Technical report, 2000.

30. Chris Liechti. pySerial Library.

31. George Levy. An introduction to quasi-random numbers. *Transport*, 2010.

32. J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, 1960.

33. Henri Faure and Christiane Lemieux. Generalized Halton sequences in 2008. *ACM Transactions on Modeling and Computer Simulation*, 2009.

34. I. M. Sobol'. On the distribution of points in a cube and the approximate evaluation of integrals. *USSR Computational Mathematics and Mathematical Physics*, 1967.

35. Francois-Michel De Rainville. Generalized Halton Number Generator.

36. PyYAML. PyYAML Framework.

37. Oren Ben-Kiki Clark Evans, Ingy döt Net. YAML.

38. Matlab. Matlab Engine for Python.

Appendix 1

Listing 6.1: “CPA memory consumption in Python for a trace file containing 10000 traces / 10000 samples”

```
Filename: cpa_bytelist_2.py

Line #    Mem usage    Increment   Line Contents
================================================
    44   457.734 MiB  457.734 MiB   @profile
    45                              def corr(hypo,traces):
    46
    47   457.734 MiB    0.000 MiB       xr, xc = np.shape(hypo)
    48   457.734 MiB    0.000 MiB       yr, yc = np.shape(traces)
    49   457.734 MiB    0.020 MiB       mean_x = np.mean(hypo, axis=0, dtype=np.float64)
    50   457.734 MiB    0.070 MiB       mean_y = np.mean(traces,axis=0,dtype=np.float64)
    51   477.309 MiB   19.594 MiB       x = hypo - np.tile(mean_x,(xr,1))
    52  1241.746 MiB  764.496 MiB       y = traces - np.tile(mean_y,(yr,1))
    53  1261.316 MiB   27.367 MiB       hyp = np.dot(np.transpose(x),y)
    54
    55  1261.316 MiB    0.004 MiB       sqrt_x = np.sqrt(np.sum(np.multiply(x,x),axis=0))
    56  1261.324 MiB    0.008 MiB       hyp = np.divide(hyp, np.transpose(np.tile(np.transpose(sqrt_x),(yc,1))))
    57
    58  1261.324 MiB    0.027 MiB       sqrt_y = np.sqrt(np.sum(np.multiply(y,y),axis=0))
    59  1261.320 MiB    0.000 MiB       hyp = np.divide(hyp, np.tile(np.transpose(sqrt_y),(xc,1)))
    60
    61  1261.320 MiB    0.000 MiB       return hyp


Filename: cpa_bytelist_2.py

Line #    Mem usage    Increment   Line Contents
================================================
    63    27.199 MiB   27.199 MiB   @profile
    64                              def computecpa(traces_file, plaintexts_file, trace_range, sample_range, byte_list):
    65
    66    27.199 MiB    0.000 MiB       start_time = time.time()
    67   409.512 MiB  382.312 MiB       traces = np.load(traces_file)[trace_range[0]:trace_range[1]+1, sample_range[0]:sample_range[1]+1]
    68   410.117 MiB    0.605 MiB       plaintexts = np.load(plaintexts_file)[trace_range[0]:trace_range[1]+1]
    69   410.117 MiB    0.000 MiB       numtraces = np.shape(traces)[0]-1
    70   410.117 MiB    0.000 MiB       numpoints = np.shape(traces)[1]
    71
    72
    73                                   #hypothetical values for each associated plaintext with the current guess
    74   410.121 MiB    0.004 MiB       hyp = np.zeros(shape=(numtraces+1,256))
    75   457.734 MiB    0.000 MiB       for byte in byte_list:
    76   457.734 MiB    0.000 MiB           for kguess in range(0,256):
    77   457.734 MiB    0.328 MiB               for tnum in range(0,numtraces+1):
    78   457.734 MiB    0.020 MiB                   hyp[tnum][kguess] = HW[intermediate(plaintexts[tnum][byte],kguess)]
    79
    80   457.734 MiB  457.734 MiB           cpa = np.abs(corr(hyp,traces))
    81
    82                                       #max_index = np.unravel_index(np.argmax(cpa, axis=None), cpa.shape)
    83   457.734 MiB    0.008 MiB           display_top_keys(byte, cpa)
    84   457.734 MiB    0.000 MiB       print(time.time()-start_time)
```