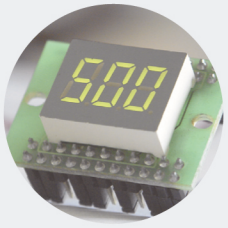
INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA ELECTROTÉCNICA E DE COMPUTADORES






Desenvolvimento de um Sistema de Ataques Side-Channel

JOÃO PEDRO MARTINS DE OLIVEIRA
Julho de 2022

POLITÉCNICO
DO PORTO

Politécnico do Porto
Instituto Superior de Engenharia do Porto

Development of a Side-Channel Attack System

João Pedro Martins de Oliveira

Master in Electrical and Computer Engineering
Specialization Area of Autonomous Systems

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto

July, 2022

This dissertation partially satisfies the requirements of the Thesis/Dissertation course of the program Master in Electrical and Computer Engineering, Specialization Area of Autonomous Systems.

Candidate: João Pedro Martins de Oliveira, No. 1170547,
1170547@isep.ipp.pt

Scientific Guidance: Veríssimo Manuel Brandão Lima Santos,
VMS@isep.ipp.pt

Company: Continental Engineering Services Portugal, Unipessoal Lda.
Advisor: Pedro Miguel Oliveira,
pedro.miguel.oliveira@conti-engineering.com

Departamento de Engenharia Eletrotécnica
Instituto Superior de Engenharia do Porto
Rua Dr. António Bernardino de Almeida, 431, 4200–072 Porto

July, 2022

Aos meus pais, avós, tios, irmãos e amigos

Acknowledgements

In the accomplishment of my project, I counted on the support of several people and institutions to which I am deeply thankful.

To my family and friends for the support and understanding during the accomplishment of the project and throughout my academic life.

I would like to thank my supervisor Pedro Oliveira and CES for providing this dissertation thesis whose topic was of such interest to me because it involved a lot of critical thinking and analysis, and as my supervisor, he always helped me through all the stages of this project. Thank you for the opportunity given to me and for the fantastic environment and conditions provided during the realization of the project

Lastly my advisor Veríssimo Santos for the accompaniment during the project and clarification of any doubt I might had.

Abstract

Nowadays consumers expect their IoT devices and data to be adequately protected against any vulnerability. As such, the implementation of protection layers should no longer be taken into account once the device is fully developed. The most common method of ensuring the security of the devices is based on the encryption of the communication sent and received by the device. Regardless of the complexity of the algorithm and the theoretical protection against brute force attacks, the attackers have evolved their strategies. Despite the developers' best efforts to secure and encrypt the device's communications, there will always be some leakage of information somewhere in the device. Similarly, the attackers have now started to exploit and analyze these leaks in order to successfully break into the so-called secure devices. By its very nature, these leaks of information will always exist, and consequently, the developers should find countermeasures to either confuse the attacker with worthless information or somehow decorrelating the leaked information from the truth. In this context, the work presented in this report presents the development of methods to verify the difficulty of decryption of the different AES 128-bit modes through power analysis, and an application developed to simplify this task for future use. Lastly, the results of the attacks performed on different targets are presented. These include a Raspberry Pi 4 and an Arduino Nano which were not successful due to the overpowering existing noise, and the ChipWhisperer Lite ARM target with 5 different AES 128-bit modes which were successfully attacked, even with countermeasures implemented.

Keywords: Side-channel Attack, Power Analysis, AES 128-bit, ChipWhisperer, Raspberry Pi, Arduino

Resumo

Atualmente, os consumidores esperam que os seus dispositivos IoT e respetivos dados sejam adequadamente protegidos contra qualquer vulnerabilidade. Como tal, a implementação de camadas de proteção deverá deixar de ser tido em conta uma vez que o dispositivo esteja completamente desenvolvido. O método mais comum para garantir a segurança dos dispositivos é baseado na encriptação das comunicações do dispositivo. Independentemente da complexidade do algoritmo usado e a proteção teórica contra-ataques por força bruta, os atacantes evoluíram as suas estratégias. Apesar dos melhores esforços dos criadores para proteger e codificar as comunicações do dispositivo, há sempre alguma fuga de informação algures no dispositivo (informação *side-channel*) em forma de vibrações, flutuações na alimentação do sistema, radiação eletromagnética, etc. Os atacantes já começaram a explorar e analisar estas fugas de modo a invadir com sucesso os dispositivos e devido à sua própria natureza, estas fugas de informação existirão sempre. Consequentemente, os criadores dos sistemas devem desenvolver e implementar contramedidas para confundir o atacante com informação inútil ou de alguma forma descorrelacionar a informação libertada da verdade. Neste contexto, o trabalho apresentado neste relatório apresenta o desenvolvimento de métodos para verificar a dificuldade de descodificação dos diferentes modos AES de 128 bits através da análise da alimentação e uma aplicação desenvolvida para simplificar esta tarefa para utilização futura. Finalmente, são apresentados os resultados dos ataques realizados aos diferentes alvos. Estes incluem um Raspberry Pi 4 e um Arduino Nano os quais não foram bem sucedidos devido ao ruído excessivo existente, e o alvo ARM do ChipWhisperer Lite com 5 diferentes modos AES 128-bit que foram atacados com sucesso, mesmo com contramedidas implementadas.

Contents

List of Figures

Listings

Acronyms

AES *Advanced Encryption Standard*

API *Application Programming Interface*

CPA *Correlation Power Analysis*

DDoS *Distributed denial-of-service*

DNS *Domain Name System*

DPA *Differential Power Analysis*

DTW *Dynamic Time Warp*

EM *Electromagnetic*

GUI *Graphical user interface*

IoT *Internet of things*

IV *Initialization Vector*

NMOS *N-channel metal–oxide–semiconductor*

PCB *Printed Circuit Board*

PMOS *P-channel metal–oxide–semiconductor*

SAD *Sum of Absolute Difference*

SCA *Side-Channel Attacks*

SoC *System on Chip*

SPA *Simple Power Analysis*

UART *Universal asynchronous receiver/transmitter*

Glossary

ACK

Acknowledgment (ACK) is a signal that is passed between communicating devices to signify the receipt of a message.

Ghost Peaks in DPA

Ghost peaks in DPA is a problem when the algorithm doesn't have an efficient way to deal with multiple strong guesses for a certain key value.

S-box

The s-box is a lookup table, in which, an 8-bit input is mapped into an 8-bit output given by the table (for AES it is the Rijndael S-box).

Shunt resistor

A shunt resistor is used to measure electric current. This is done by measuring the voltage drop across the resistor.

Statistical Significance

Statistical significance is the assertion that the results of an experiment are due to an underlying reason, as opposed to chance.

Time complexity

Time complexity is an estimation of the amount of time it takes to run an algorithm. It's commonly estimated by counting the number of elementary operations performed by the algorithm, supposing that each elementary operation takes a fixed amount of time to perform.

Chapter 1

Introduction

The accelerating growth of connected devices and the sensitive information they generate poses a complication for manufacturers seeking to protect their devices from attack. Today, consumers expect their IoT devices and data to be properly protected against any vulnerabilities or various exploits, thus inferring that multiple layers of protection should be considered as a design parameter rather than a post-development implementation. Historically, protection solutions that included encryption with one key were considered secure. However brute force attacks became virtually impotent due to the length of the decryption key. As a result, a category of attacks was found that purely ignore the mathematical operations of a cryptographic system, thus making the focus of their attacks, the hardware implementation of the system itself. From an information security angle, the protection of sensitive data requires the implementation of algorithms that are resistant to theoretical attacks. Despite that, taking a purely mathematical approach or in other words, abstracting away from the physical (hardware/software) implementations of the device usually results in unintended outputs such as execution time, power consumption, EM radiation, among others. The existence of said undesired outputs, usually known as side-channel information, does not directly compromise the sensitive information. The branch of knowledge that explores is formally known as Side-Channel Analysis (SCA). As will be presented ahead with different requirements come different attacks, and as such, there is a practical trade-off between the effort spent on some attacks and the benefits gained from a successful attack. Nevertheless, there are dedicated countermeasures against SCA that can aid developers to create a safer device

but come at their own expense in terms of increased execution time or resource demand.

1.1 Motivation

Traditional cryptoanalysis handles the input and output pairs but assumes no knowledge of the internal states of the device under attack. However, the appearance of side-channel attacks reveals that even a cryptographic device can leak critical information. By monitoring these manufacturing undesired leaks, the attackers can gain information about the internal data or operations and extract the cryptographic key used to encrypt the communications with the device without mathematically breaking the encryption itself. While new methods of tampering and newer attacks are being proposed and developed, designing a secure system becomes increasingly difficult. Owing to the fact that the attacker only needs to succeed once out of many attacks, the designer has to prevent all the applicable attacks and variants simultaneously. Furthermore, the countermeasures of one attack might benefit another. Consequently, keeping up with the most recent developments in the field of attacks and corresponding countermeasures is an ever-lasting task. Currently, newer generations of chips do not include on-chip flash for manufacturing cost reasons. They make use of a serial interface available for initialization (e.g. CAN) and large amounts of DRAM. As such the system is open to a serious problem because with a serial programmer it is possible to read and manipulate the information sent. The natural solution to this would be to encrypt the implicit data. In this context, the work presented in this report proposes as its main challenge, to verify the difficulty of decryption of the aforementioned communications and the creation of a system to simplify such tasks for future use.

1.2 Company

This dissertation was developed together with Continental Engineering Services (CES) in favor of a solution to the proposed problem in 1.1. Created in 2006, the company has evolved from serving the automotive sector into also partnering with customers in diverse spaces such as aerospace, construction machinery, agriculture, among others. With over 1800 employees in 23 locations worldwide, CES's mission is to generate technical solutions for all challenges. The following corporate values are fundamental to all employees and form the roots of the corporate culture: Trust, Passion to Win, Freedom to Act and For One and Another.

1.3 Objectives

The main goals for the problem at hand are the following:

- Study and Research
  - Side-Channel Attacks
  - AES-128 encryption/decryption
  - Hardware capabilities
- Development
  - Attack on AES-128
  - Graphical User Interface (GUI)
- Testing and Validation

1.4 Document structure

This document is composed of 6 distinct chapters. In the first chapter, Introduction, we contextualize this dissertation, describe the problem, followed by a brief presentation of the company, and lastly introduce the objectives proposed. The second chapter, the State of the Art, was used to analyze tools that could be used in this project's scope and to compare each one. The third chapter was used to introduce some of the research made, by displaying some fundamentals essential to the development of this project. The fourth chapter, Implementation, describes the technical details used during the process of development, and the implementation of the solution The fifth chapter, Results, presents the outcome of the tests conducted on the project. The sixth chapter, Conclusion, presents the outcome of the project, an outlook on future development, and lastly its strengths and limitations.

Chapter 2

State of the Art

The rapid growth of IoT devices has resulted in an estimated 46 billion connected devices and it's expected to reach 83 billion by 2024. The majority of them are very likely to be unsecured. Such potential dangers of these poorly protected IoT devices have been demonstrated by the massive DDoS attack on the Dyn DNS[1]. Consequently, the population of homes with other IoT and internet-connected devices creates much more worrying thoughts. Most IoT devices can communicate with each other, creating an inadvertent channel of communication that circumvents the internet. The big problem arises from the fact that even the industry-standard cryptographic techniques used to obfuscate the messages can be abused by computer criminals to spread infectious malware to neighbors or just gain control of the whole network. An introduction to IoT-related security and privacy problems and potential solutions can be found in [2][3][4]. An example of a worm that rapidly spreads over large areas through the infection of the very popular Philips Hue smart lights system can be seen in [5]. Many other exploits can endanger the automotive space [6], the industrial environment, intellectual property theft or reverse engineering,[7] and even take control of an airplane [8]. And more recently, in [9] introduces a new family of side-channel attacks on the very commonly used 8th to 11th generation of Intel CPUs, and Zen 2 and 3 from the AMD side.

One of the products responsible for aiding in the exploitation of such problems is the ChipWhisperer. It started as a very successful Kickstarter and has since been presented at conferences such as DEFCON and Black Hat among others. Its initial goal was to revolutionize the embedded security industry by linking the realms of

academic research and practical engineering by offering the market a cheap and accessible solution. According to NewAE Technology Inc. (currently responsible for ChipWhisperer), "ChipWhisperer is an open-source toolchain that makes learning about side-channel attacks easy and affordable. It also serves as a platform for performing side-channel research in a well documented, cost-effective, and repeatable way" [10] and at the moment has several products such as learning kits and development boards that make it easy to get into the study of most variants of SCA [11]. With aid of this hardware, there is an abundance of paper such as [12][13] demonstrating its decryption power and exploit capabilities. Some attacks include the aforementioned Philips Hue Smart lights project and common IoT device exploits such as routers [14], Smart card readers [15] and many others that can be easily found since there is a very active and prominent community behind this product.

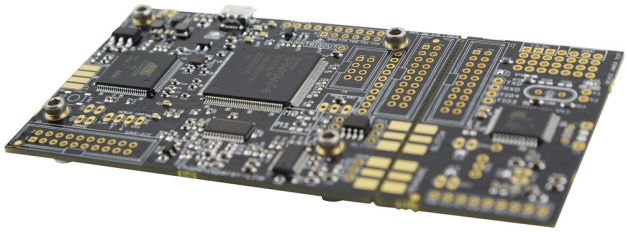


Figure 2.1: ChipWhisperer Lite

One other method of performing an SCA involves injecting errors through an electromagnetic radiation injector, also known as Fault Injection Attack [16]. Previous to the development of the product in question, the standard product on the market for such applications was the ChipSHOUTER, which was too expensive to be considered an attractive solution.[17]

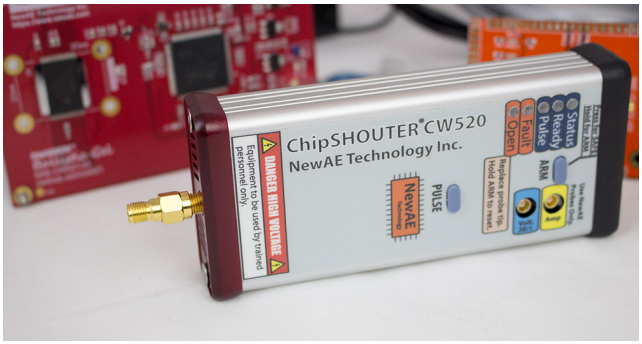


Figure 2.2: ChipSHOUTER® (CW520)

SiliconToaster, created by Karim M. Abdellatif and Olivier Heriveaux, presents a cheap solution relative to the market when it comes to the injection of electromagnetic radiation.[18]

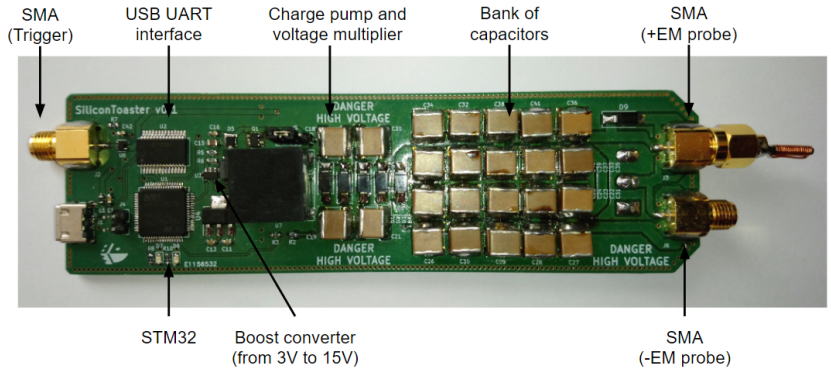


Figure 2.3: SiliconToaster

Finally, for a more costly package, Rambus provides a wide range of hardware and visualization software to perform tests and analysis of cryptographic vulnerabilities of chips and systems to SCA of origin in their power supply or electromagnetic origin. Normally, the target interest in the platform in question is government organizations, testing labs, and others since such a product includes a required training period and a higher initial cost [19].

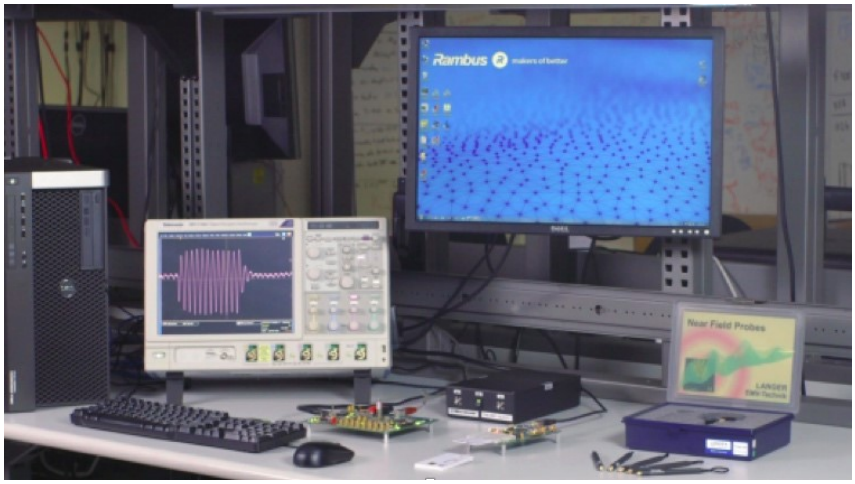


Figure 2.4: Rambus DPAWS

Chapter 3

Fundamentals

3.1 Origin of leakages

The concept of extracting data from a computer by analyzing side-channel information, such as its power consumption or electromagnetic emissions, is known since World War II [20]. The existence of such attacks is are based on physical changes that happen and can be observed during the execution of computing operations on electronic devices. As previously mentioned, these observed physical phenomena can be manifested in a variety of ways. In this chapter, the focus of attention will be on power consumption. Given that Complementary Metal Oxide Semiconductors gates (CMOS) are the basis of digital circuits and the major contributor to the implied power leaks, this introduction will also clarify this technology. Under standard conditions, static CMOS is very power efficient since they dissipate nearly zero power when idle. Nevertheless, the power dissipation on such a device can be divided into two subcategories: [21] [22] [23] [24]

3.1.1 Static dissipation

The static mode of operation occurs when the input signals are stable. Static power dissipation normally refers to the dissipation of power arising in the static mode of operation or in the active mode of operations in the case of idle.

- Sub-threshold Leakage - In both transistors (NMOS and PMOS), when the gate-source voltage is below the specified threshold value, the so-called sub-threshold current doesn't suddenly cut off, but instead plunges exponentially to zero. Due to constraints in the development of increasingly smaller devices, the value of this current is no longer insignificant.
- Gate leakage - The ongoing down-scaling of chips results in smaller gate oxides which cause an increase in the electrical field across the oxide layer, which in return increases the probability of electrons tunneling through the oxide layer in both directions.

3.1.2 Dynamic/Active dissipation

- Short-circuit power - There is always a slope in the input wave on the transitions due to the capacitance of the gate. During this very short period, there is a direct path from the supply rail to ground. The appearing short-circuit current is directly dependent on the switching frequency since it rests on the rise and fall times of the input signal.
- Switching Power - During the transition of states, a dissipation of power is created by the charging and discharging of capacitive nodes.

Countermeasures to the presented issues can be seen in [24] [25].

3.2 Side-Channel Attack

A Side-Channel Attack is a security exploit that seeks to extract secrets from a chip or a system by bypassing the theoretical strength of cryptographic systems. All systems (including cryptographic ones) leak information about the internal processes. This means that attackers can use this leaked information, and with the aid of various techniques, extract the decryption key and other information from the device. Through measurements or analysis of various physical parameters such as supply current, execution time, and electromagnetic emissions from the system it is possible to gather information of influence the program execution. It's a form of cryptoanalysis that exploits the weaknesses found in certain implementations of the cryptographic routine. Some examples of SCA can include analyzing the power supply to the system, analyzing the electromagnetic radiation emited by the system, or even by creating errors through the injection of electromagnetic radiation. Side-channel attacks can be diversified into two main categories: evasive/non-evasive and passive/active.

- Non-evasive vs Evasive: Evasive attacks require depackaging the different layers of the chip in the target system. Depending on the depth, (number of layers

removed of the chip) the attack can be considered semi or fully evasive. For more in-depth see [26]. Non-evasive attacks only exploit (the unintentional) information leaked, such as running time, and power consumption.

- Passive vs Active: Passive attacks do not require manipulation of the device's inputs/outputs or working environment. On the other hand, Active attacks manipulate the device's operation by inducing abnormal behavior caused by the injection of various types of faults (electrical, optical, etc.) or by engaging in glitching attacks.

Since these attacks are based on the physical properties of the device, and not (only) on mathematics, there are numerous types of attacks such as timing attacks [27][28] power analysis attacks (SPA, DPA, template attacks) [11][14], and EM attacks (Electromagnetic attacks) [16].

Depending on the security of the device and the sophistication required there are available a lot of techniques ranging from low to high complexity and attack potential.

3.3 Power analysis

Among the techniques available for obtaining the secret keys of a system, one of the easiest and most effective ways is using power analysis. In this method, the variation in power consumption is used to determine the information contained in the device. There are two types of power analysis.

3.3.1 Simple Power Analysis (SPA)

Simple Power Analysis examines the device's current consumption over a period of time. Given that different operations demonstrate distinct power profiles, one can determine the function type at a given time (e.g., multiplications consume more current than additions). SPA does not rely on mathematical methods but instead involves directly interpreting the power traces. It is most useful when the data-dependent features are apparent in the power traces, but when there is significant noise in the system, SPA is not practical.

3.3.2 Differential Power Analysis (DPA)

Differential Power Analysis is a statistical method for analyzing power consumption with the intent of recognizing data-dependent correlations between sets of traces from multiple cryptographic operations.

Through enough traces, even small correlations can be identified regardless of the noise, since it's canceled during the averaging.

DPA focuses on the processing of chosen key-dependent intermediate values and analyses multiple measurements of this particular operation to test a set of hypotheses. For such an attack, the following assumptions must hold:

- The attacker must be able to measure the power consumption of the target device as it performs several runs with a fixed key and random input data.
- The attacker must know either the plaintext or the ciphertext.

The biggest problem revolves around the large amounts of traces needed to break AES, and it can suffer from ghost peaks

3.3.3 Correlation Power Analysis (CPA)

This method replaces checking the average power consumption over many traces to instead check if the guessed sub-key also has a linear relation to the device's power consumption across a set of traces. Like DPA, this method will need to be repeated for each point in time long the power trace. The correlation of the power consumption is based on a chosen model. In our case, the chosen model is the Hamming Weight. The relationship between the two datasets is calculated through Pearson's correlation coefficient [29].

$$\rho(X, Y) = \frac{Cov(X,Y)}{\sigma_x \sigma_y}$$

Where:

- Cov(X,Y) - the covariance between X and Y
- σ_x - the standard deviation of the variable x

The value of Pearson's coefficient will always be between [-1, 1], and the stronger the relationship between the variables, the closer the value will be to these limits.

3.4 Advanced Encryption Standard (AES)

The Advanced Encryption Standard (AES) has been adopted worldwide as a standard for securing information. A high-level description of the algorithm can be found in 3.1

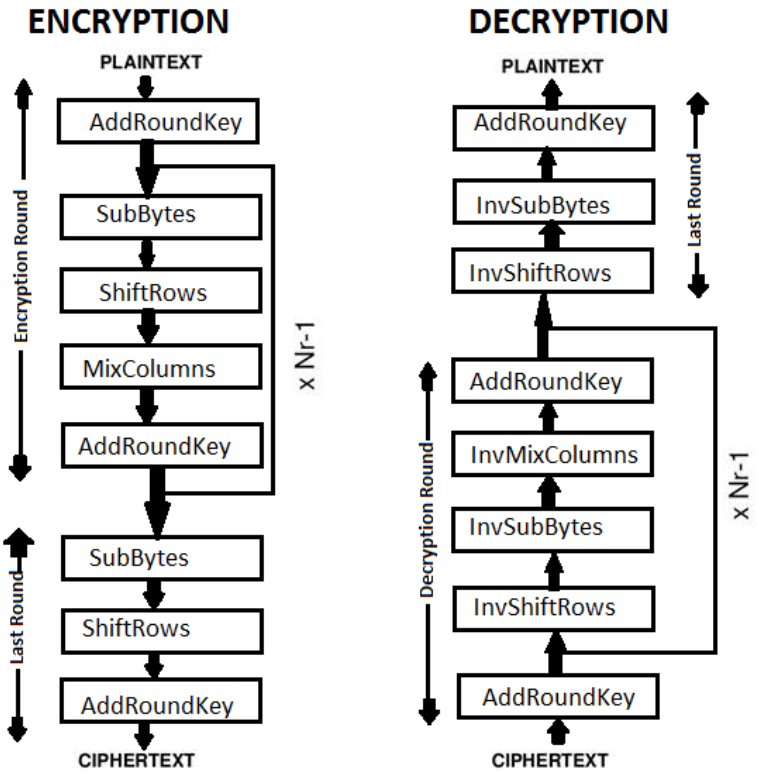


Figure 3.1: Diagram of AES architecture [30]

where *Nr* corresponds to the number of rounds (in the case of the AES-128 it's 10), and the remaining operations can be described as:

- AddRoundKey - each byte of the state is XORed with a byte of the round key
- SubBytes - each byte is replaced by another according to a lookup table
- ShiftRows - shift cyclically the last three rows of the state
- MixColumns - combining the four bytes in each column

Based on the recommendation of the National Institute of Standards and Technology (NIST), there are five main modes of operation[31]: Electronic Code Book (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), the Output Feedback (OFB), and the Counter (CTR).

Electronic Code Book Mode

This mode is the simplest of all and is not used anymore. The message is divided into blocks, and these are encrypted separately.

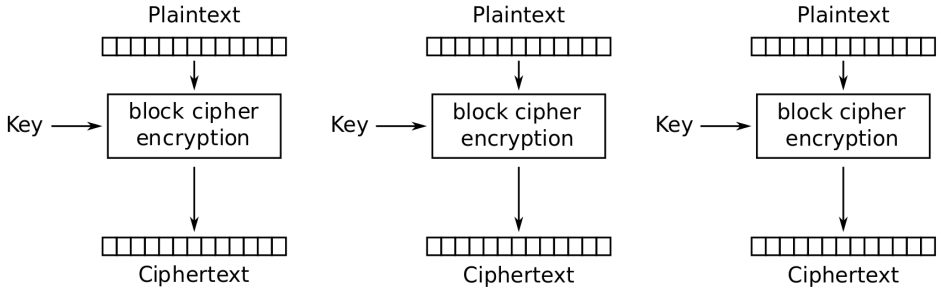


Figure 3.2: Diagram of Electronic Code Book (ECB) mode of encryption [32]

The massive flaw in this mode is the clear lack of diffusion, and as such, identical plaintext blocks are encrypted into identical ciphertext blocks

Cipher Block Chaining Mode

In this mode, each block is XORed with the previous ciphertext before being encrypted. This way, each ciphertext block depends on the previous ones. The first block is XORed with a random/pseudorandom number, the initialization vector (IV).

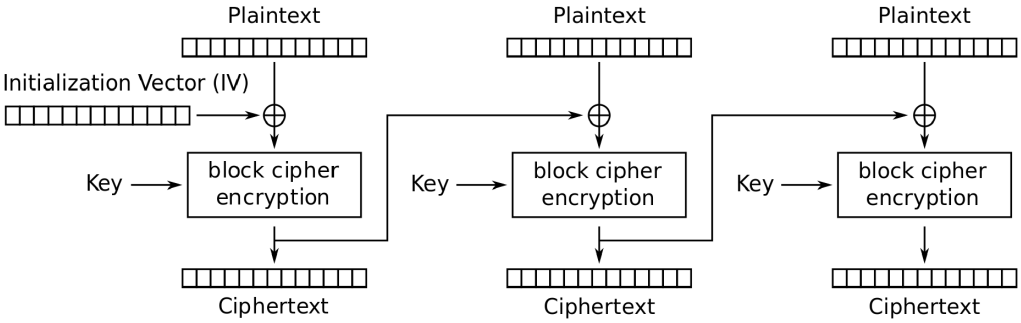


Figure 3.3: Diagram of Cipher block chaining (CBC) mode of encryption [32]

This has been the most commonly used mode of operation but has some flaws. The encryption is sequential, and as such cannot be parallelized, and the message must be padded to a multiple of the cipher block size.

Cipher Feedback Mode

This mode of operation uses the encryption block as a stream cipher.

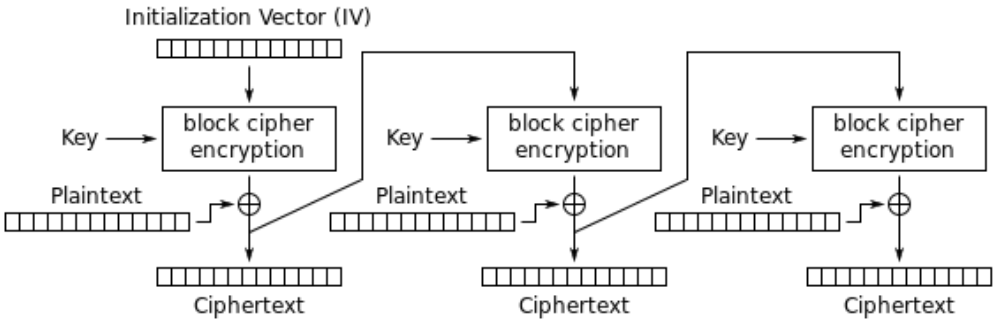


Figure 3.4: Diagram of Cipher Feedback (CFB) mode of encryption [32]

It doesn't need to pad the data, since it doesn't encrypt the plaintext directly, and it could decrypt data in parallel, but not encryption. Similar to CBC, if there is a broken block, it affects the following block.

Output Feedback Mode

Very similar to the CFB mode, but it always encrypts the IV

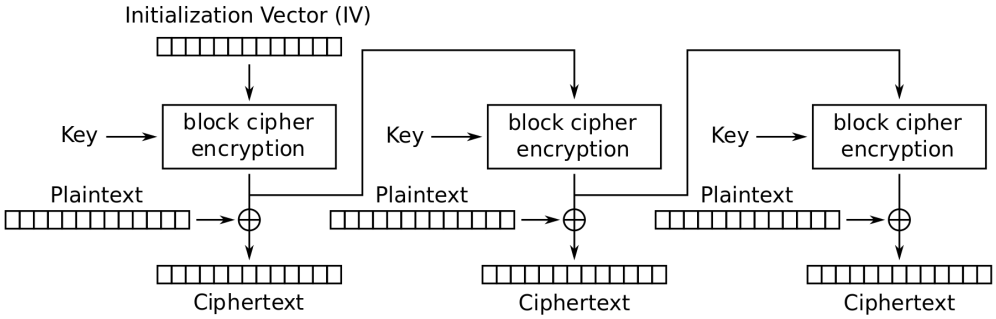


Figure 3.5: Diagram of Output Feedback (OFB) mode of encryption [32]

It also cannot encrypt and decrypt in parallel but is not affected by a broken block.

Counter Mode

Similar to the OFB, the Counter mode turns a block cipher into a stream cipher but generates the next keystream block by encrypting successive values of a counter.

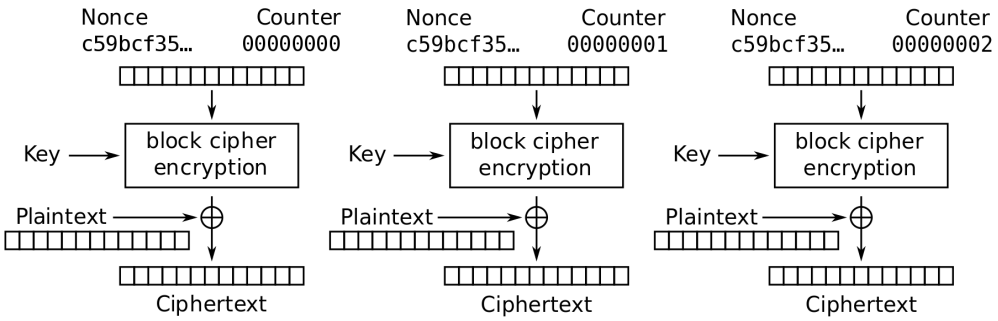


Figure 3.6: Diagram of Counter (CTR) mode of encryption [32]

This mode is not affected by the broken block, like OFB. But it uses a counter to be encrypted every time instead of the IV. It also can encrypt/decrypt data in parallel.

3.4.1 CPA attack on AES-128

The attack on a system with AES encryption is made through guesses to the encryption key, though these guesses will not take the complexity of the key 2^{128}. Since we will be guessing one byte of the key at a time, the complexity will become 16×2^8 which equates to a total of 4096 guesses that are very easily calculated by a computer nowadays. From a side-channel attack standpoint, the smaller details of each step are not important. The focus of the attack will be on the first block of *SubBytes* 3.1 although the other blocks can be used. The guess of the current key-byte will be passed with the same byte of the input to the specified encryption model, and thus we hypothesize the output of the first block of *SubBytes*. As explained in section 3.1, there is a small but significant power dissipation, which is manifested in the transition of the gates in a chip. By the same nature, the temporary storage of data of bytes will prompt spikes in the power consumption of the target device. Depending on the design of the device it is possible to see a draw or a drop in power usage, either way the value will be directly related to the number of 1's in the binary representation of the byte in question (*Hamming weight*). The Pearson correlation function [29] is used to assign a correlation value between the hypothesized *SubBytes* output and a point of a trace. Iterating over all traces captured and all possible guesses for one byte of the key, we assume as the correct guess the one with the largest correlation value.

3.4.2 Common countermeasures to SCA

Countermeasures can fall into two main categories:

- Eliminating/reducing the release of side-channel information - Some countermeasures can include jamming the emitted channel with noise, more specifically random delays or spikes in amplitude, and implementing a silicon-based hardware root of trust (state machine designed to perform a specific set of functions like data encryption, validation of certificates).

- Eliminating the relationship between the leaked information and the secret data - Some examples include altering the algorithm's input into unpredictable states to prevent leakage and implementing a mask (this is widely used in practice, but is also considered an empirical solution and its effectiveness are rarely proved [33]). One of the best approaches consists in designing hardware with constant power consumption, but this is very expensive to implement. Shamir [34] proposes a de-correlation of the external power supplied to the internal power consumption of the chip. The addition of capacitors on the power supply path will filter/smooth out the power consumption trace, essentially making it constant in time.

Chapter 4

Implementation

This chapter discusses aspects of the development of the analysis algorithms, the created leakage models, and the algorithms used to bypass implemented countermeasures.

4.1 Implementing DPA

The first step in the development of a project of this nature was to understand deeply the implementation of the attack. As such, a script was developed for analyzing the AES 128-bit ECB mode using only the *NumPy* [35] library for simplifying array operations (the theory of the attack was explained in more detail in 3.4.1).

```
1  def aes_internal(inputdata, key):
2      return sbox[inputdata ^ key]
3
4  t_bar = np.sum(trace_array, axis=0)/ len(trace_array)
5  o_t = np.sqrt(np.
       sum((trace_array - t_bar)**2, axis=0))
6
7  cparefs = [0] * 16 # key byte guess correlations
8  bestguess = [0] * 16 # key byte guess
9
10 for bnum in  range(0, 16):
```

```
11      maxcpa = [0] * 256
12      for kguess in  range(0, 256):
13
14          hws = np.array([[HW[aes_internal(textin[bnum],
                kguess)] for textin in textin_array]]).
                transpose()
15          hws_bar = mean(hws)
16          o_hws = std_dev(hws, hws_bar)
17          correlation = cov(trace_array, t_bar, hws,
                hws_bar)
18          cpaoutput = correlation/(o_t*o_hws)
19          maxcpa[kguess] =  max( abs(cpaoutput))
20
21      bestguess[bnum] = np.argmax(maxcpa)
22      cparefs[bnum] =  max(maxcpa)
```

Listing 4.1: Implementation of DPA from scratch [36]

4.2 Implementing leakage models

The following step, naturally, was to expand this script to analyze the other AES-128 modes stated in 3.4. From here on, the ChipWhisperer library was used in order to simplify the implementation of the remaining code and to provide additional information regarding the analysis.

Analyzing the provided leakage model used to crack the ECB mode we can observe the *leakage* method returns the output of s-box with the hypothesized guess for the key byte, as explained in 3.4.1.

```
1 class SBox_output(AESLeakageHelper):
2     # Leakage model for the ECB encryption mode
3     name = 'HW: AES SBox Output, First Round (Enc)'
4
5     def leakage(self, pt, ct, key, bnum):
6         return self.sbox(pt[bnum] ^ key[bnum])
```

Listing 4.2: Leakage model of ECB mode [36]

Furthermore, when implementing the leakage models for the remaining modes, it should first be determined the viable and accessible vulnerabilities.

4.2.1 CBC mode

Now, looking back at the previously shown Fig 3.3, we note that the input to the encryption is not just the plaintext as illustrated in the ECB mode in Fig 3.2. In this case, the input is the plaintext XORed with the ciphertext of the previous encryption (except the first time, which is replaced by the initialization vector). And, since in the current implementation we have knowledge of the IV and have access to the ciphertext of every trace, our leakage model needs to store the ciphertext of one trace, to be used in the next one.

```
1 class AesCBC(cwa.AESLeakageHelper):
2     def __init__(self):
3         self.prevct = [0] * 16 # IV
4     def leakage(self, pt, ct, key, bnum):
5         final = self.sbox(pt[bnum] ^ self.prevct[bnum]
              ^ key[bnum])
6         self.prevct = ct
7         return final
```

Listing 4.3: Leakage model of CBC mode

4.2.2 CFB mode

The leakage model for this mode is analogous to the ECB one, however, the plaintext is replaced by the ciphertext of the previous iteration. Similar to the previous mode, it will be required to store the ciphertext, as it is the input of the encryption. But unlike the previous models, we do not require any information regarding the plaintext.

```
1 class AesCFB(cwa.AESLeakageHelper):
2     def __init__(self):
3         self.prevct = [0] * 16 # starts as IV value
4     def leakage(self, pt, ct, key, bnum):
5         final = self.sbox(self.prevct[bnum]^key[bnum])
6         self.prevct = ct
7         return final
```

Listing 4.4: Leakage model of CFB mode

4.2.3 OFB mode

Following the previous, this leakage model will also need to store some information, since one iteration of this mode is reliant on the previous iteration, though this one will be somewhat different. The information to be stored in this case will be the plaintext XORed with the ciphertext, resulting in the output of the encryption block.

```
1 class AesOFB(cwa.AESLeakageHelper):
2     def __init__(self):
3         self.prevoutput = [0] * 16 # starts as IV
              value
4     def leakage(self, pt, ct, key, bnum):
5         final = self.sbox(self.prevoutput[bnum] ^ key[
              bnum])
6         self.prevoutput = ct ^ pt
7         return final
```

Listing 4.5: Leakage model of OFB mode

4.2.4 CTR mode

The implementation of this final mode of operation is quite distinct from the others. Since we have no clear input to this mode, we will have to approach this leakage model from the other end. When XORing the ciphertext with the plaintext, we get the output of the encryption block. Passing this value through a reverse s-box presents the output of the 10th round of the encryption. Our leakage model will be evaluating this value with the output of the 10 rounds with our current key guess.

```
1 class AesCTR(cwa.AESLeakageHelper):
2     def leakage(self, pt, ct, key, bnum):
3         aa = ct[bnum] ^ pt[bnum]
4         st9 = self.inv_sbox(aa ^ key[bnum])
5         return st9
6     def process_known_key(self, inpkey):
7         return key_schedule_rounds(inpkey, 0, 10)
```

Listing 4.6: Leakage model of OFB mode

It should be noted though, that the output of this model is the best guess for the yield of the 10th round of the encryption. To retrieve the original key, we will be taking advantage of the ChipWhisperer API, which has the method “key_schedule_rounds()” which can reverse the rounds, returning the encryption key.

4.3 Bypassing countermeasures

Following the implementation of the different leakage models, the next logical step would be to develop some approach in case the target device had countermeasures implemented to prevent these types of attacks. In this project, there were implemented scripts to evade the addition of random delays and amplitudes (jitter) explained in 3.4.2.

The addition of jitter can be implemented as a simple procedure to deter attacks or even just encountered as a result of unwanted jittery triggering on the UART communication. The most commonly implemented processing technique used to resynchronize the captured traces is through Sum of Absolute Difference (SAD) also known as Sum of Absolute Error. In this pre-processing, a segment of the traces is used as a ’benchmark’. This segment is then “slid over the ’input window’ for each trace, and the amount of shift resulting in the minimum SAD criteria is selected as the shift amount for that trace” [37].

Random delays are implemented as a method of discouraging timing attacks. This artificial "noise" forces the attacker to collect more measurements. It should be noted although, that standalone noise introduction is incapable of sufficiently masking side-channel emissions. In this case, the attacker can effectively bypass this countermeasure by resynchronizing the captured traces with Dynamic Time Warp (DTW). DTW is an algorithm for measuring similarity between two temporal sequences and also calculates an optimal match between the given sequences. The ChipWhisperer API utilizes the FastDTW algorithm which greatly reduces the time of the resynchronization process. Normal DTW has a time complexity of O(NM) where N and M are the lengths of the two input sequences, but the FastDTW [38] implementation provides near-optimal alignments with O(N) time, through the use of a multilevel approach that recursively projects a solution from a coarser resolution and refines the projected solution.

4.4 Capture settings

When working with foreign hardware (outside of the ChipWhisperer), it is almost required to have control over the parameters of the capture of the traces. With aid of the ChipWhisperer API, we gain the possibility of modifying the parameters of the capture in the ChipWhisperer board. These include the amount of traces and

points in a trace, the settings of the scope such as gain, clock speed, the triggering input for the capture, the possibility to provide the plaintext to be used in the capture, and a lot more [39]. One classic example of the need for access to these settings will be presented in the following chapter where it is documented an attack on an Arduino. This hardware usually runs at a 16 MHz clock, and according to the *Nyquist Theorem* [40], the sampling rate should be at least double the target speed. Since by default, the sampling rate does not follow the aforementioned theorem, this value should be modified in accordance.

4.5 Graphical User Interface

With the purpose of facilitating the verification of the encryption process of a developed project, a GUI was created. Using the library PyQt5 [41], the interface is able to capture traces from the target device and save them in a ChipWhisperer project (.cwp) and consequently analyze these files, with the desired encryption mode and also assuming some countermeasures were implemented. Since the target audience for this project would be the engineers at CES, the following application was created with the desires and requirements of these developers in mind. The main window of the application created displays 4 distinct quadrants which provide the following tools:

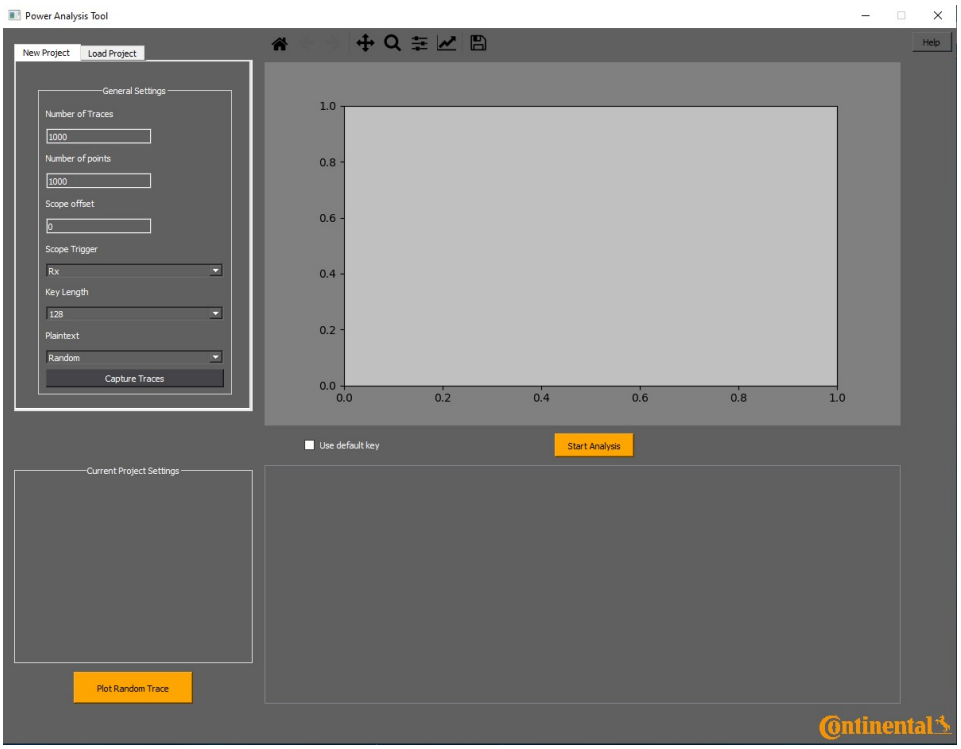


Figure 4.1: GUI Main window

- The top left Widget incorporates 2 tabs, that when pressed display 2 different features:
  - The first tab is designated for the creation of a new project. In this tab, it is possible to select the desired features in the capture of the traces of a new project. Other parameters include the number of traces to capture and the number of points in each trace, the offset applied to the scope, the trigger method for reading the traces, the length of the encryption key, and an option to provide the plaintext to be used in the moment of capture. Finally, at the bottom end of the tab, a button to start the capture of the traces is provided.
  - The second tab is designated for the loading of an existing project and the selection of an encryption model to be used in the analysis. Additionally, it's possible to select the algorithm to apply for the resynchronization of traces. It also provides a button to easily export the current loaded projects' plaintext and ciphertext as text files.

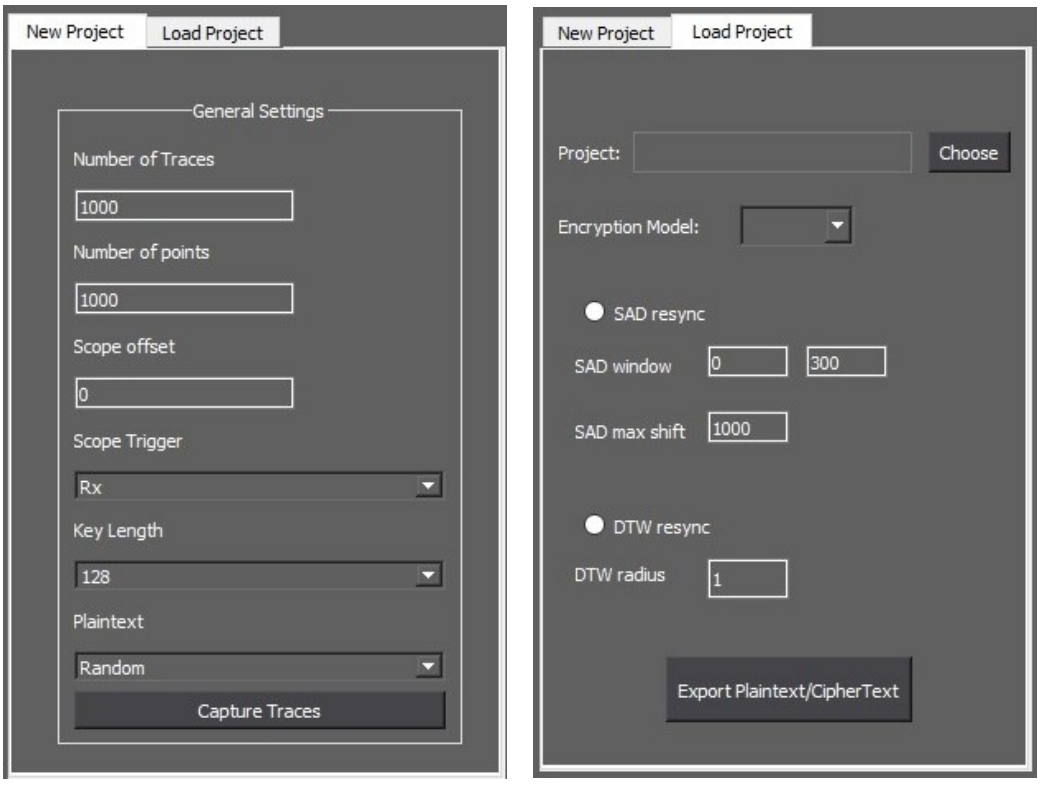


Figure 4.2: New Project tab (left) and Load Project tab (right) in the main window

- Top-right Widget includes a frame for plotting waves of a currently loaded project and its respective navigation toolbar. Since this Widget is based on

the Matplotlib library, the toolbar includes actions such as pan, zoom, being able to export the canvas as an image, and much more.

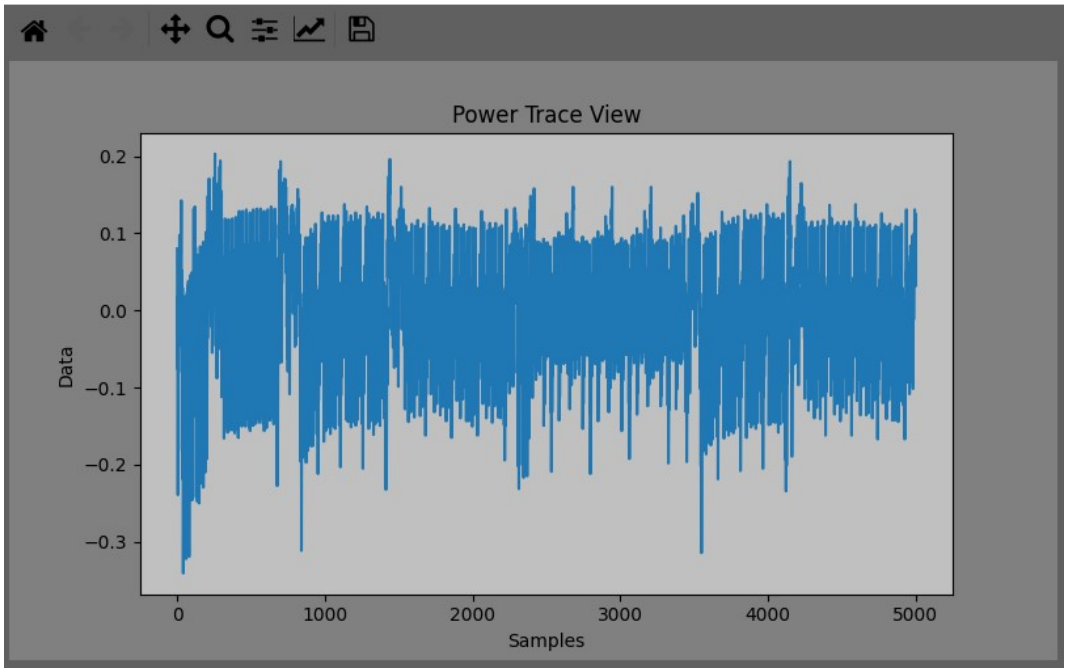


Figure 4.3: Example of a plot in the canvas of the main window

- Bottom-left Widget integrates a large text box to exhibit the metadata of the currently loaded project. This Widget is automatically updated once a project is loaded with its information and prints the full path of the project, the number of traces, and the number of points in a trace. Under this box, a button is presented which, when pressed plots in the canvas a random point of the currently loaded project.

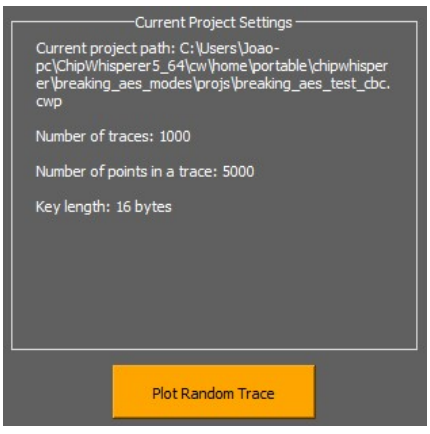


Figure 4.4: Example of the text box displaying information regarding the currently loaded project.

- Bottom-right Widget completes the GUI by providing a button to start the analysis and a table to watch the analysis in real-time. In order to maintain the usability of the developed application, this Widget was implemented in a QThread (regular thread with better integration with the Qt library). In the analysis, once a defined number of traces are analyzed the analysis invokes a callback, sending a signal from this thread to the main window with the currently updated table, to be updated. To use the ChipWhisperers API for analyzing projects, custom table models and classes were developed. Once the analysis is complete, a text file is exported with the full table.

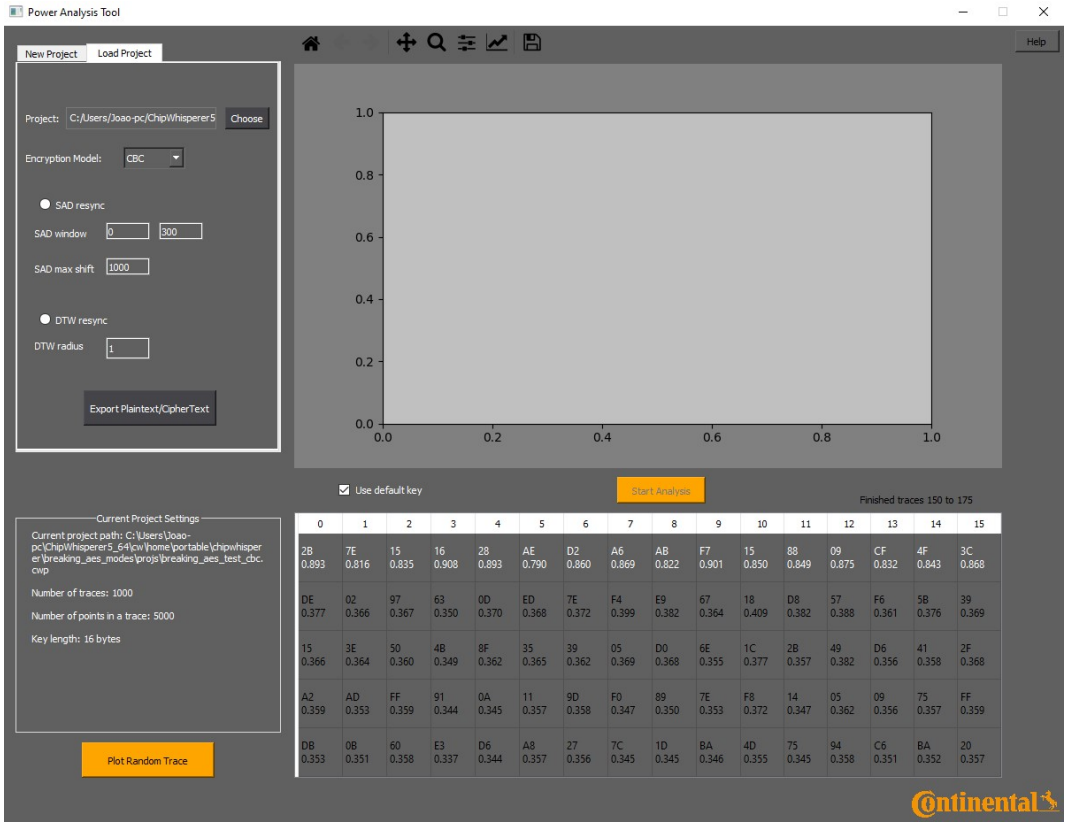


Figure 4.5: An application running an analysis. The white text in the table indicates the correct byte for that column according to the provided key text file.

Additionally, on the top right corner, a help button was implemented with the intent of providing the helper window. This window provides all the information regarding every observable parameter in the main window and some additional back-end information. Some of this information includes the meaning of every parameter in the main window, the list of the required hardware for the acquisition of traces, explanations of the desired formats for input files (default key, plaintext), and more.

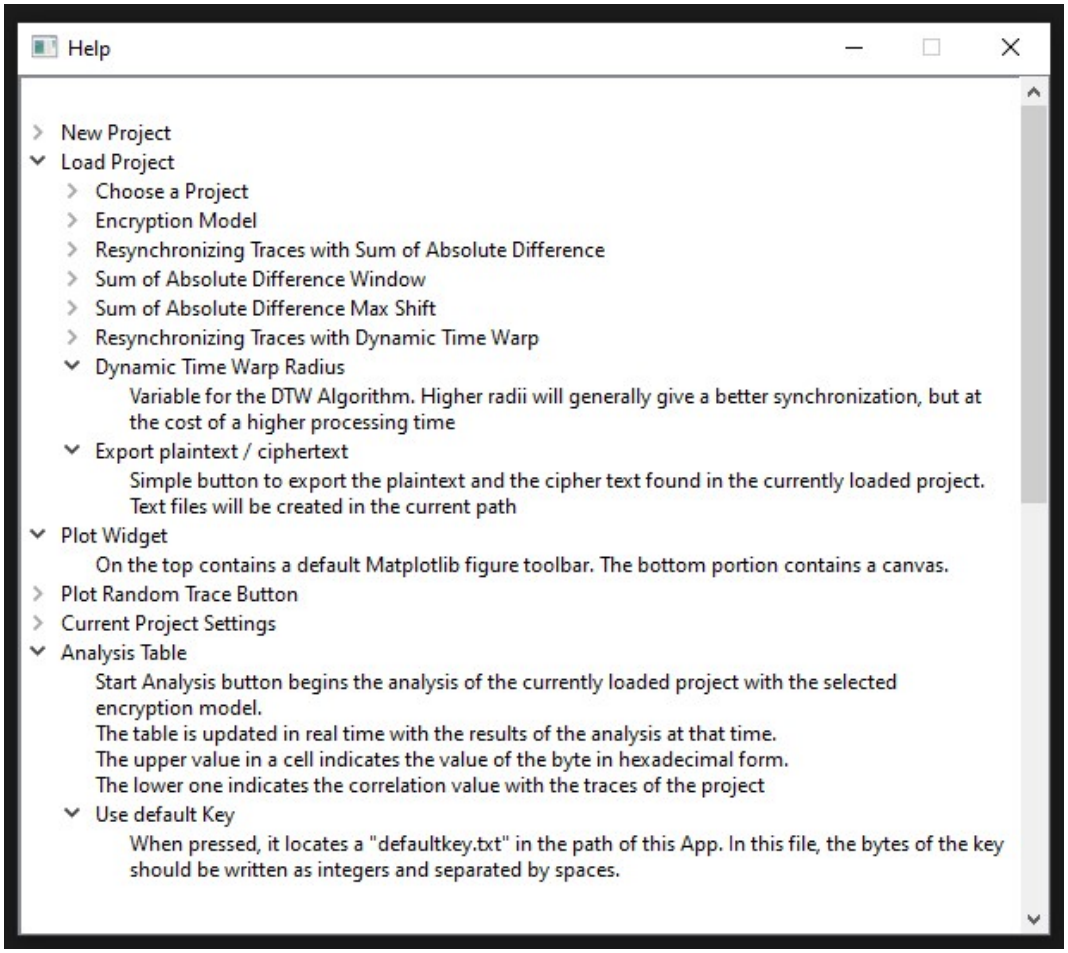


Figure 4.6: Helper window of the Application

Chapter 5

Results

As we will observe in the following images, some technical difficulties were encountered due to the noise found which obfuscates the correct assessment by the system. The attacks on external hardware will have somewhat similar problems, but with rather different approaches and distinct practical solutions. The custom scripts running on the foreign hardware are portrayed by the following:

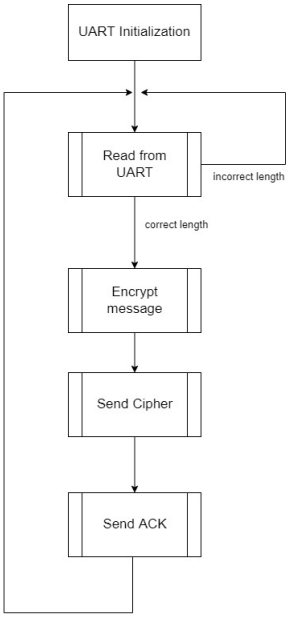


Figure 5.1: Flow chart of the implementation on both the Raspberry (in C++ and Python) and the Arduino devices

5.1 Raspberry Pi

The first attack on foreign hardware was with the Raspberry Pi 4b. Naturally, even during the preparations, it was realized that the attack would be quite unlikely since it is a much faster and noisier target. Nevertheless, it served as an example of a very complex target with background tasks. The development of the communication between the Raspberry and the ChipWhisperer would have to be created since the Raspberry Pi is not currently supported by the SimpleSerial communication protocol[42]. As such scripts were created, but with only the strictly necessary sections (Annex A.1).

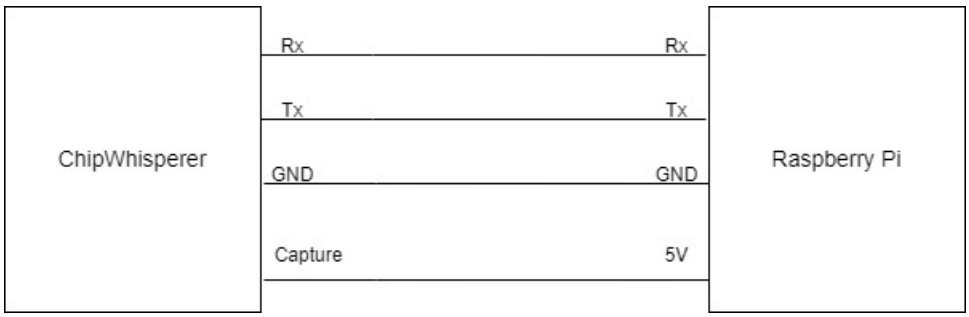


Figure 5.2: Diagram of Raspberry and ChipWhisperer connections

5.1.1 Problems encountered

The possibility of this attack being successful was rather low from the start, due to two problems. The first is regarding the clock speed of the target device. According to the Raspberry PI technical specifications, the default clock speed of the device is 1.5 GHz, which is far superior to the ChipWhisperers' maximum sample rate of 105 MS/s. The second problem is the Raspberry Pis hardware power distribution system. As informed by a forum on the Raspberry Pi official forums, the target power demand passes through a switch-mode regulator, whose job is to smooth out the power. Ideally, an attacker would try to bypass the power regulation to capture directly the power pins of the target device. In this case, there was no expertise present to aid with this process, thus producing another deadlock. Nevertheless, since reducing the clock to the required speed would not be possible and the aforementioned soldering proficiency was impossible at the time, the objective was to observe the captured traces for both Python and C++ implementations and attempt to find some correct information regarding the encryption key and also any significant differences between both implementations.

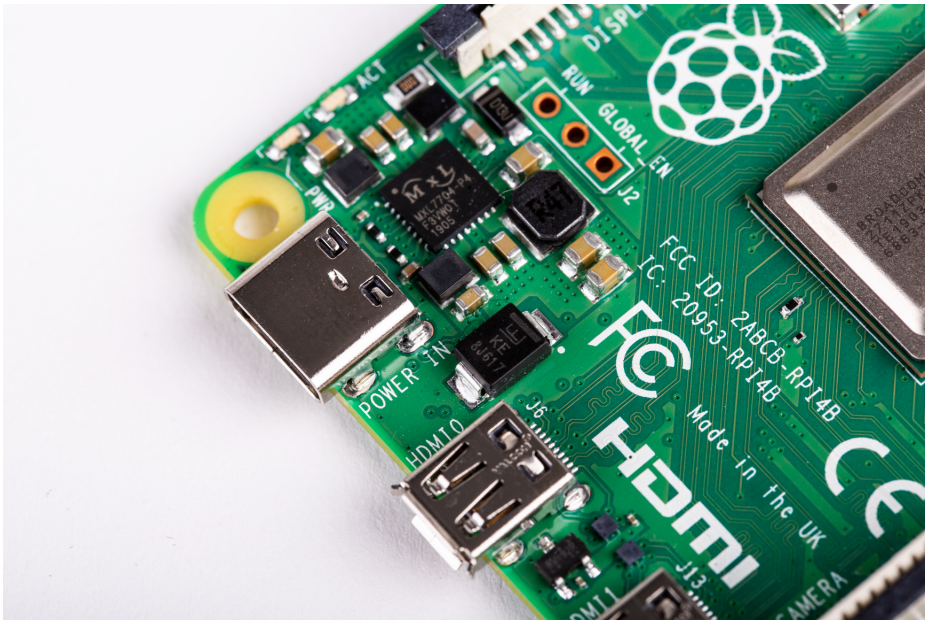


Figure 5.3: A figure of the power traces and regulators on a Raspberry Pi 4.

As can be seen in the Figures 5.4 and 5.5, both implementations seem to display large amounts of noise in the traces. By the lack of ability and knowledge about the wiring of the device, the capture was done through the GPIO, thereby introducing additional noise.

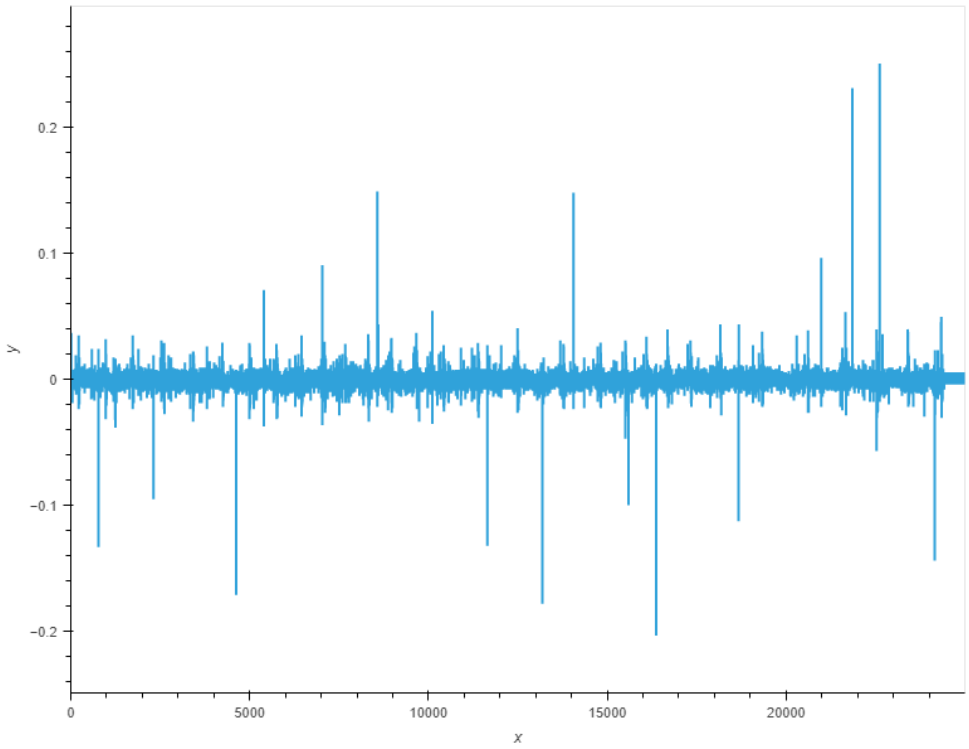


Figure 5.4: Example of a trace in the C++ implementation of the script in a Raspberry Pi 4

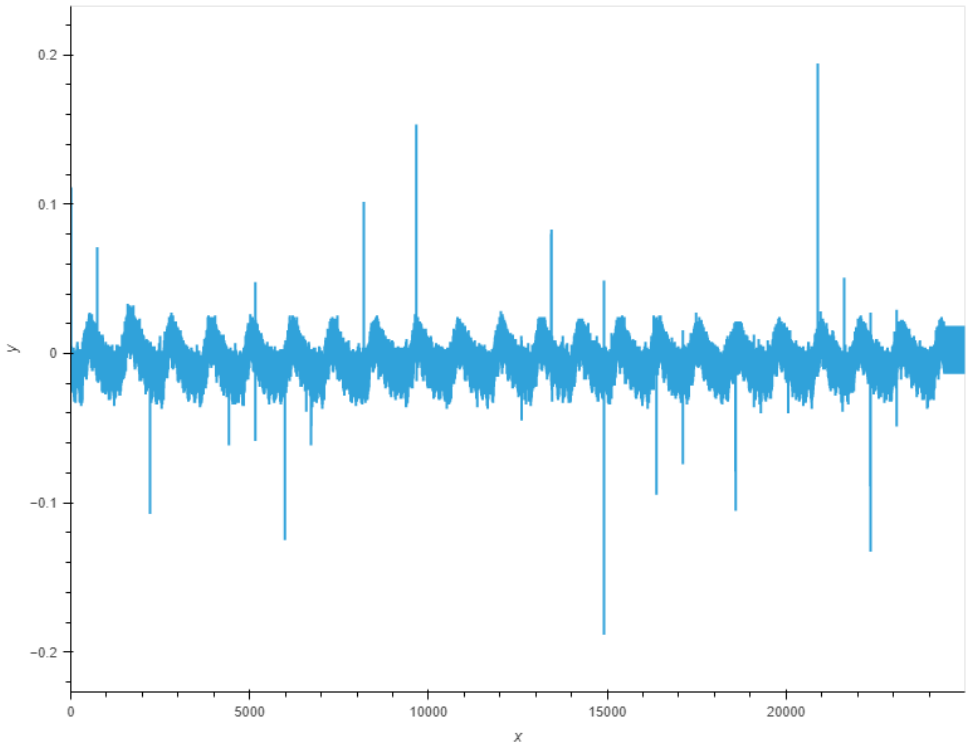


Figure 5.5: Example of a trace in the Python implementation of the script in a Raspberry Pi 4

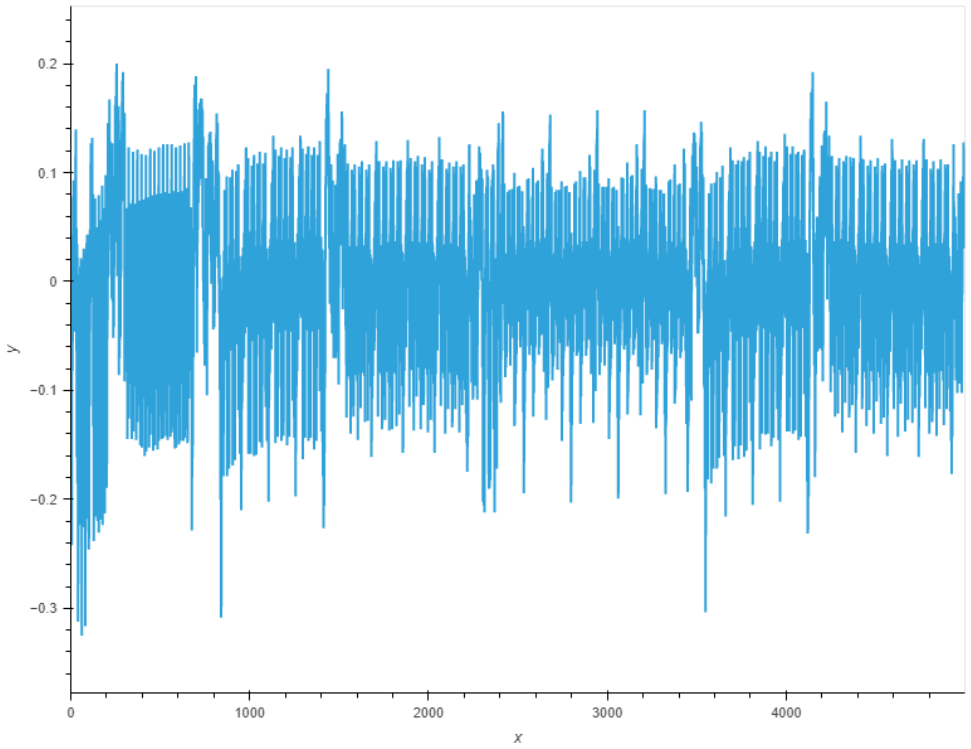


Figure 5.6: Example of a usable trace captured with the ChipWhisperer Target in ECB mode

The big differences between the examples of the implementations are the result of the large distinction between both programming languages. As Python is a more abstract language, it is possible to observe a more cyclic repetition during operation. In contrast, in the C++ implementation, it is clear there is no apparent periodical segment, as this language is compiled and optimized it's often faster and more efficient. This can be observed by the reduced fluctuation and lack of periodicity. Figures 5.4 and 5.5 represent an example of two different ChipWhisperer projects, and both of them were created with the scope trigger as the Rx pin of the UART. Figure 5.6 is an example of an ECB implementation on the ChipWhisperer target device with the GPIO 4 as the scope trigger.

5.1.2 Potential Solutions

To successfully attack a Raspberry Pi target, the following points should be addressed:

- Given that the SoC (System on chip) contains not just the core which is doing the encryption but also other cores and other blocks doing the processing for the Ethernet stack, the USB stack the display, the analog audio PWM and has many blocks being derived and distributed there will be an excess of noise in the captured traces. For that reason, replacing the target device model with

a Raspberry Pi Zero should be considered, preferably underclocked to the 5 MHz range, producing a more apparent and evident noise.

- This uncorrelated noise due to the background processes can consequently be filtered with statistical techniques, given enough data.
- Finally, on the hardware modifications part, it should be clear that any change shouldn't prevent the Pi from working. The traces should be captured as the voltage drop on a shunt resistor in line with the power regulation circuit and before the chip itself.

5.2 Arduino

The second attack on external hardware was on the Arduino Nano. As this target device has a low clock speed and an open-source development, an implementation would be far less obstructed (see Annex B.1). The initial step was to add the shunt resistor to capture the traces and remove some capacitors to make the power oscillations more evident. The preparations for this attack included the same connections as the previous attack, increasing the ChipWhisperers ADC clock to 32 MHz (double the 16 MHz of the Arduino) and also configuring the scope trigger in the Rx pin.

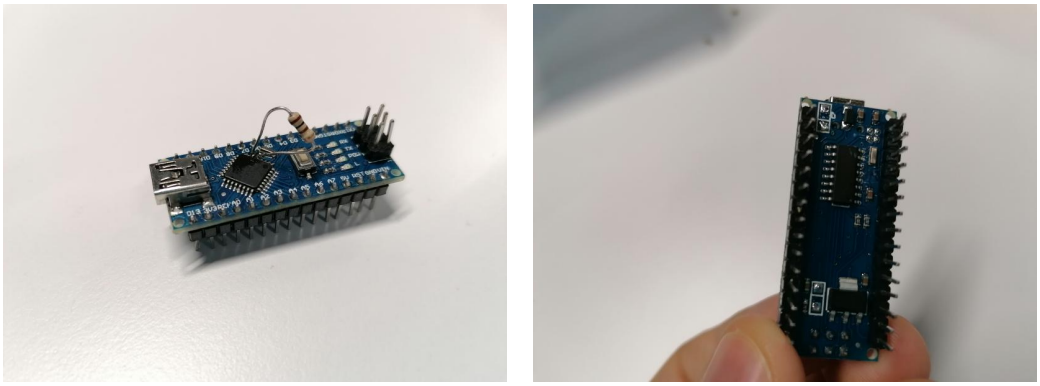


Figure 5.7: Arduino hardware modifications. On the left, is the top side of the Arduino Nano with the visible shunt resistor. On the right, is the evidence of the removed capacitors.

The idea was to insert a resistor between the chip's power pin and its pad on the PCB (Printed Circuit Board). This operation while very delicate, was implemented and the device continued to operate normally. Nevertheless, the structural stability of the modification was not perfect, and minor handling of the device would place strain on the already feeble connection. This problem would certainly create noise, which is sustained by Figure5.8.

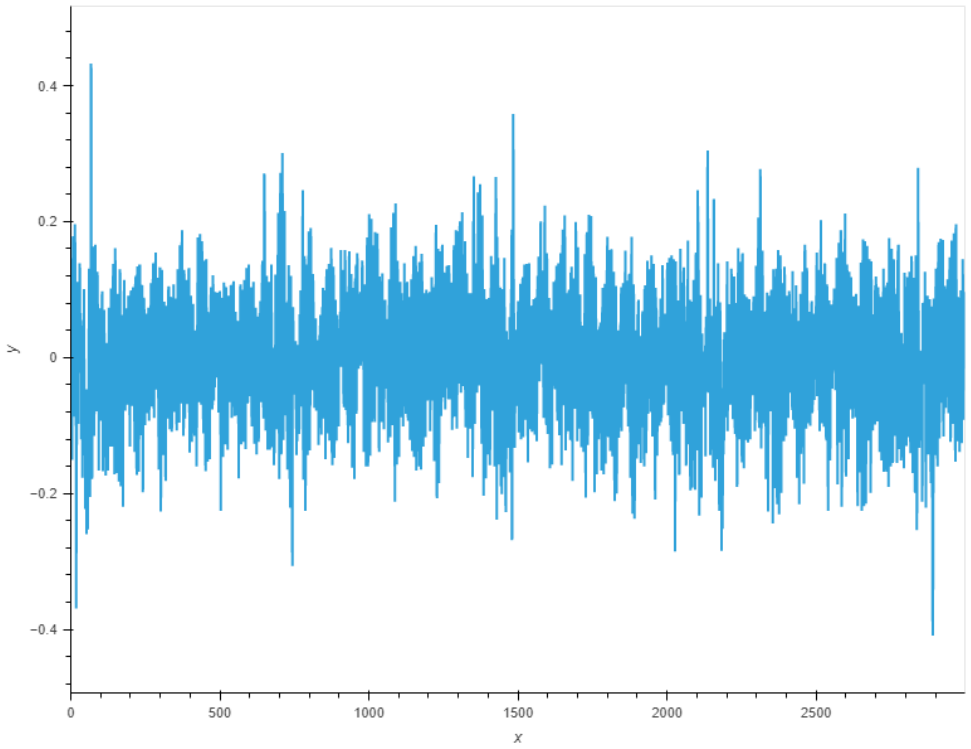


Figure 5.8: Example of a trace captured from the Arduino Nano

5.2.1 Potential Solutions

Potential solutions capable of solving the noise problem encountered include:

- Replace the current shunt modification with a more reliable, less noisy approach that accomplishes the same purpose
- Remove the few lasting capacitors on the back side of the device as they might be contributing to some smoothing of the power
- Reduce the Arduinos clock and increase the ChipWhisperers. Assuming there is no impact on the device's functional capabilities, there should be a better depiction of the power wave.

It's also probable that some attackers if given the same results as presented here, might try to transplant the device's chip into a custom board with a more easily accessible, reliable, and less noisy implementation. An example of a custom, board with the same chip (Atmega328p) is known as the Notduino (Figure 5.9), and it's sold alongside the ChipWhisperer as a target board.

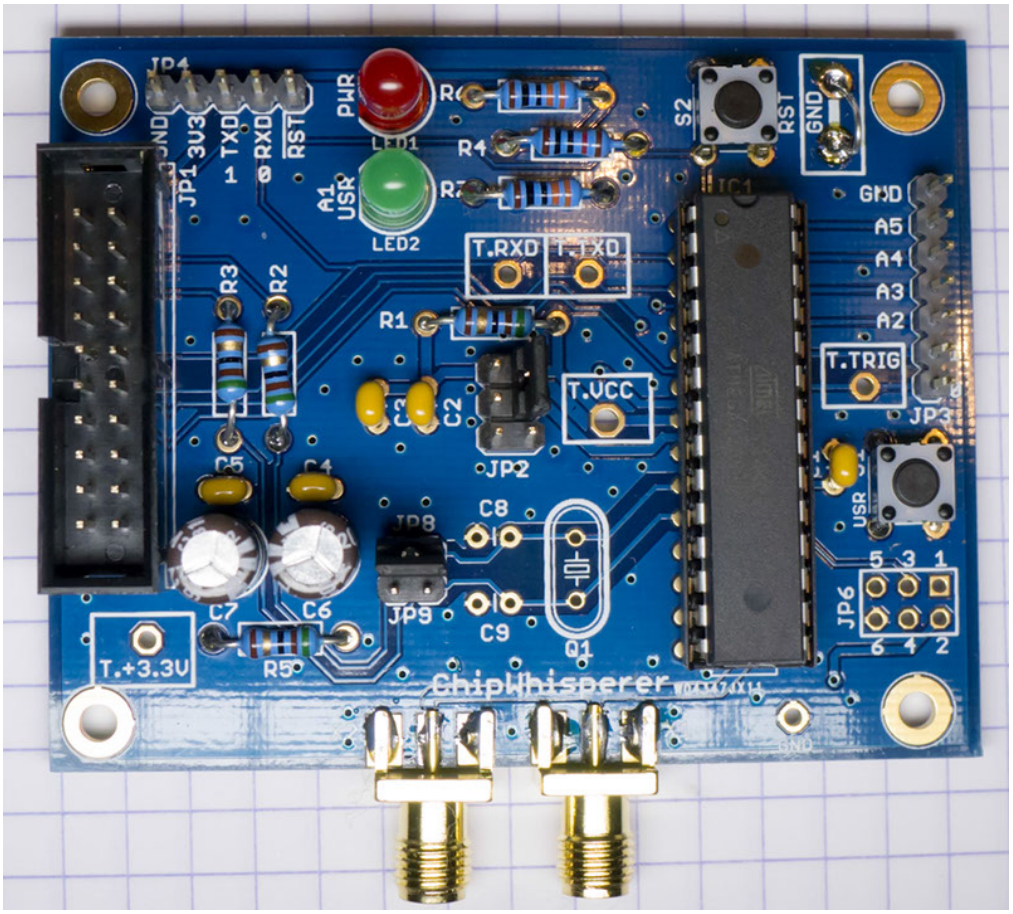


Figure 5.9: Notduino target, from Newae at [10]

5.3 ChipWhisperer ARM Target

Even though the previously mentioned targets had far too much noise in the capture of the traces, there was a collective of very successful attacks on the Arm target device of the ChipWhisperer. These attacks are the result of 10 different firmware implementations comprised of the five AES modes of operation and with and without the addition of random noise. The firmware for the target was provided from [36], and these attacks were conducted with the same connections as the previous ones.

The correct outcome of these attacks was due to the implementation of this target being far more reliable, and as such, it didn't present the same problems exhibited in the previous targets. Although lacking the hardware difficulties, this target was useful to demonstrate the effectiveness of the analysis scripts even when presented with countermeasures implemented.

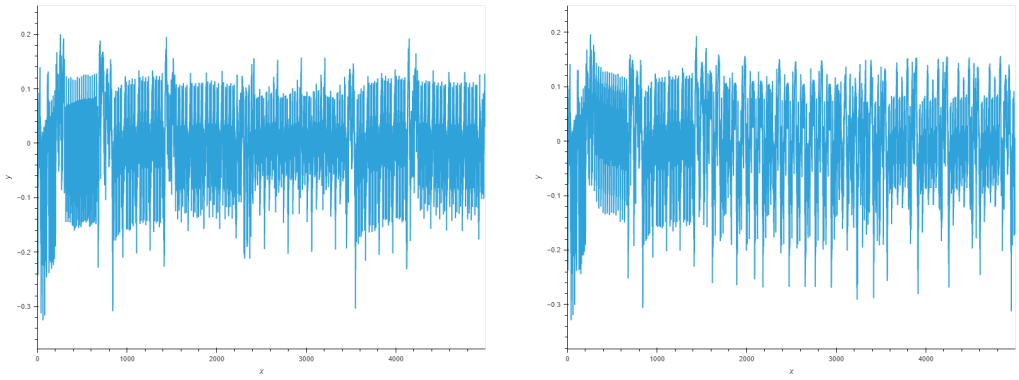


Figure 5.10: Example of two traces from two implementations of the mode ECB. On the left there is no noise introduced in the firmware of the target device. On the right, the target firmware was compiled with the addition of random delays

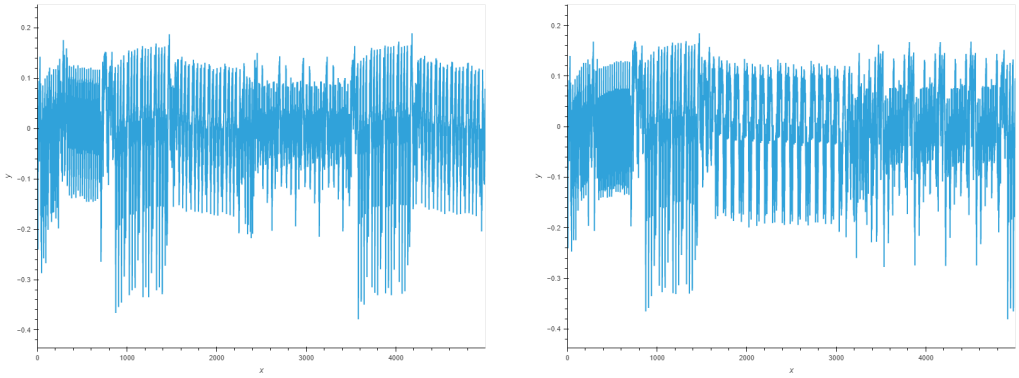


Figure 5.11: Example of two traces from two implementations of the mode OFB. On the left, there is no noise introduced in the firmware of the target device. On the right, the target firmware was compiled with the addition of random delays

Chapter 6

Conclusion

Considering the accomplished project, here are some reflections on the proposed objectives and events encountered. Apart from the aforementioned problems encountered in the testing phase, it's reasonable to confirm that the proposed goals and intentions of the project have been successfully achieved. Although some of the experiments were rather unfruitful, they shed light on some potential limitations and confirmed the project's strengths. These limitations are appertaining to the noise incoming from the hardware implementation for the capture of the traces. Unreliable and untrustworthy implementations produce more noise that is undesirably captured and inadvertently contaminates the power traces. Nevertheless, with the aid of more complex algorithms such as deep learning ones, these limitations might not be the limiting factor of a successful attack. At the same time, given a strong enough analysis, it should be possible to obtain a successful outcome with statistical significance, regardless of the amount of noise. Finally, previous to the development of this project, there was no other Application developed at CES Portugal with this purpose. As such, this should serve as an initial approach to the overall side-channel problem. The overall development of the GUI was successful, and through conversations and some testing with the Engineers at CES, some additions were implemented to aid in usability. Mainly the helper window, where the user can understand the meaning behind some unclear buttons/labels or even with links to more in-depth explanations.

6.1 Future development

As indicated previously, the major negative point was the testing with external hardware, which, as mentioned earlier could be solved by a more reliable and less noisy implementation. Regarding the case of the Raspberry Pi, perhaps the attack would have been more fruit full on the model Pi-zero underclocked to a more manageable speed. Also, since the ChipWhisperer used is not equipped for communication at higher speeds, perhaps a PicoScope [43] would meet the demand. Regarding the GUI, the methods of resynchronization provided in the ChipWhisperer API do not have callbacks, and as such, right now the only option to observe the progression of this process is through the terminal output. Also when applying the resynchronizations it would be reasonable to automate the SAD algorithm (even if it results in a more extensive process).

References

[1] B. Krebs, "Hacked cameras, dvrs powered today's massive internet outage." `https://krebsonsecurity.com/2016/10/hacked-cameras-dvrs-powered-todays-massive-internet-outage/`, 2016. [Cited on page 5]

[2] A.-R. Sadeghi, C. Wachsmann, and M. Waidner, "Security and privacy challenges in industrial internet of things," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015. [Cited on page 5]

[3] G. A. Fink, D. V. Zarzhitsky, T. E. Carroll, and E. D. Farquhar, "Security and privacy grand challenges for the internet of things," in *2015 International Conference on Collaboration Technologies and Systems (CTS)*, pp. 27–34, 2015. [Cited on page 5]

[4] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014. [Cited on page 5]

[5] E. Ronen, A. Shamir, A.-O. Weingarten, and C. O'Flynn, "Iot goes nuclear: Creating a zigbee chain reaction," in *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 195–212, 2017. [Cited on page 5]

[6] B. Nassi, Y. Mirsky, D. Nassi, R. Ben-Netanel, O. Drokin, and Y. Elovici, *Phantom of the ADAS: Securing Advanced Driver-Assistance Systems from Split-Second Phantom Attacks*, p. 293–308. New York, NY, USA: Association for Computing Machinery, 2020. [Cited on page 5]

[7] M. Riley and A. Vance, "Inside the chinese boom in corporate espionage," *Bloomberg*, 2012. [Cited on page 5]

[8] D. G. Ware, "Hacker took control of united flight and flew jet sideways, fbi affidavit says," *Ledger, Donjon, France*, 2015. [Cited on page 5]

[9] Y. Wang, R. Paccagnella, E. T. He, C. F. Hovav Shacham, and D. Kohlbremner, "Hertzbleed: Turning power side-channel attacks into remote timing attacks on x86," *31st USENIX Security Symposium*, 2022. [Cited on page 5]

[10] N. T. Inc., "Newae hardware product documentation." https://rtfm.newae.com/. [Cited on pages viii, 6, and 36]

[11] C. O'Flynn and Z. D. Chen, "Chipwhisperer: An opensource platform for hardware embedded security research," in *in: Constructive side-channel analysis and secure design - COSADE*, 2015. [Cited on pages 6 and 11]

[12] S. Moini, X. Li, P. Stanwicks, G. Provelengios, W. Burleson, R. Tessier, and D. Holcomb, "Understanding and comparing the capabilities of on-chip voltage sensors against remote power attacks on fpgas," in *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pp. 941–944, 2020. [Cited on page 6]

[13] L. Lathrop, "Differential power analysis attacks on different implementations of aes with the chipwhisperer nano," *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 1008, 2020. [Cited on page 6]

[14] J. Dofe, J. Frey, and Q. Yu, "Hardware security assurance in emerging iot applications," in *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2050–2053, 2016. [Cited on pages 6 and 11]

[15] D. E. Ryad Benadjila, Mathieu Renard and P. Trébuchet, "Leia: the lab embedded iso7816 analyzer a custom smartcard reader for the chipwhisperer," *SSTIC2019*, 2019. [Cited on page 6]

[16] J. Danial, D. Das, S. Ghosh, A. Raychowdhury, and S. Sen, "Scniffer: Low-cost, automated, efficient electromagnetic side-channel sniffing," *IEEE Access*, vol. 8, pp. 173414–173427, 2020. [Cited on pages 6 and 11]

[17] N. T. Inc. https://www.newae.com/products/nae-cw520. [Cited on page 6]

[18] K. M. Abdellatif and O. Heriveaux, "Silicontoaster: A cheap and programmable eminjector for extracting secrets," *Ledger, Donjon, France*, 2020. [Cited on page 6]

[19] Rambus. https://www.rambus.com/security/dpa-countermeasures/dpa-workstation-platform/. [Cited on page 7]

[20] U.S. National Security Agency, "Tempest: A signal problem," 1972. [Online; accessed January-2022]. [Cited on page 9]

[21] V. Chinta, "Subthreshold and gate leakage current analysis and reduction in vlsi circuits," 2007. [Cited on page 9]

[22] M. Stockinger, *Optimization of ultra-low-power CMOS transistors*. PhD thesis, Technical University of Vienna Faculty of Electrical Engineering, 2000. [Cited on page 9]

[23] F.-X. Standaert, *Introduction to Side-Channel Attacks*, pp. 27–42. Boston, MA: Springer US, 2010. [Cited on page 9]

[24] Physical_design, "Sources of power dissipation in cmos." `https://www.physicaldesign4u.com/2020/01/sources-of-power-dissipation-in-cmos.html`. [Cited on pages 9 and 10]

[25] A. C. Siva G. Narendra, *Leakage in Nanometer CMOS Technologies*. Springer, Boston, MA, 2006. [Cited on page 10]

[26] M. Hutle and M. Kammerstetter, "Chapter 4 - resilience against physical attacks," in *Smart Grid Security* (F. Skopik and P. Smith, eds.), pp. 79–112, Boston: Syngress, 2015. [Cited on page 11]

[27] D. J. Bernstein, "Cache-timing attacks on aes," 2005. [Cited on page 11]

[28] P. C. Kocher, "Timing attacks on implement at ions of diffie-hellman, rsa, dss, and other systems." https://link.springer.com/content/pdf/10.1007[Cited on page 11]

[29] "Pearson correlation." https://www.newae.com/products/nae-cw520. [Cited on pages 12 and 16]

[30] N. Projects, "Image encryption using aes algorithm," 2022. [Cited on pages vii and 13]

[31] M. Dworkin, "Recommendation for block cipher modes of operation: Methods and techniques," *SP 800-38A*, 2001. [Cited on page 13]

[32] Wikipedia Commons, "Block cipher mode of operation," 2022. [Online; accessed February-2022]. [Cited on pages vii, 14, 15, and 16]

[33] E. Prouff and M. Rivain, "Masking against side-channel attacks: A formal security proof," in *Advances in Cryptology – EUROCRYPT 2013* (T. Johansson and P. Q. Nguyen, eds.), (Berlin, Heidelberg), pp. 142–159, Springer Berlin Heidelberg, 2013. [Cited on page 17]

[34] A. Shamir, "Protecting smart cards from passive power analysis with detached power supplies," in *Cryptographic Hardware and Embedded Systems — CHES 2000* (Ç. K. Koç and C. Paar, eds.), (Berlin, Heidelberg), pp. 71–77, Springer Berlin Heidelberg, 2000. [Cited on page 17]

[35] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. Fernández del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy,

W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, p. 357–362, 2020. [Cited on page 19]

[36] N. T. Inc., "Chipwhisperer." `https://github.com/newaetech/chipwhisperer/`, 2022. [Cited on pages ix, 20, and 36]

[37] NewAE Technology Inc., "Cw-analyzer tool," 2016. [Online; accessed March-2022]. [Cited on page 23]

[38] S. Salvador and P. K. Chan, "Fastdtw: Toward accurate dynamic time warping in linear time and space," 2004. [Cited on page 23]

[39] NewAE Technology Inc., "Chipwhisperer," 2022. [Online; accessed February-2022]. [Cited on page 24]

[40] P. Colarusso, L. H. Kidder, I. W. Levin, and E. Neil Lewis, "Raman and infrared microspectroscopy," in *Encyclopedia of Spectroscopy and Spectrometry* (J. C. Lindon, ed.), pp. 1945–1954, Oxford: Elsevier, 1999. [Cited on page 24]

[41] T. Q. C. Riverbank Computing Limited, "Pyqt5 reference guide," 2021. [Cited on page 24]

[42] NewAE Technology Inc., "Simpleserial documentation," 2022. [Online; accessed March-2022]. [Cited on page 30]

[43] P. Technology, "Pc oscilloscope and data logger products," 2022. [Cited on page 40]

Appendix A

Python implementation on the Raspberry

```
1  import serial
2  from Crypto.Cipher import AES
3  import time
4
5  ser = serial.Serial(
6    port='/dev/serial0',
7    baudrate = 38400,
8    parity=serial.PARITY_NONE,
9    stopbits=serial.STOPBITS_ONE,
10   bytesize=serial.EIGHTBITS,
11   timeout=1000)
12
13 keyraw = b'2b7e151628aed2a6abf7158809cf4f3c'
14 key = [ int(keyraw[i:i+2],16) for i in  range(0,
       len(keyraw),2)]
15 cipher = AES.new(bytes(key), AES.MODE_ECB)
16
17 ack = b'z00\n'
18
```

```
19
20 while True:
21   try:
22     if ser.in_waiting > 0:
23       data = ser.readline()
24
25       if  len(data)==34:
26         command = data[0]
27         pt = [ int(data[i:i+2],16) for i in  range(1,
              len(data)-1,2)]
28
29         if  chr(command) == 'p':
30
31           ct = cipher.encrypt(bytes(pt))
32
33
34           rawmsg = ['r']
35           rawmsg.extend(['{:02x}'.
                format(a) for a in ct])
36           rawmsg.extend(['\n'])
37           msg = ''.join(rawmsg)
38
39           a=ser.write(msg.encode())
40           a=ser.write(ack)
41
42   except OSError:
43     print("error")
44     ser.close()
45     ser. open()
46 ser.close()
```

Listing A.1: Python implementation on the Raspberry

Appendix B

Implementation on the Arduino

```
1
2  void loop() {
3    char  buffer[34];
4    while (Serial.available()) {
5          Serial.readBytes( buffer, 34);
6
7          char currbyte[2];
8          unsigned char pt[16];
9          unsigned char ct[16];
10         for( int i=0; i<16; i++){
11             memcpy(currbyte,  buffer+1+i*2, 2);
12             currbyte[2]='\0';
13             pt[i] = ( int) strtol(currbyte ,0, 16);
14         }
15
16
17         aes_encrypt(ct, pt, key);
18
19         char ack[5] = "z00\n";
20         char msg[50] = "r";
```

```
21          char a[3];
22          for( int i=0;i<16;i++){
23              sprintf(a, "%02X", ct[i]);
24              strcat(msg, a);
25          }
26          strcat(msg, "\n");
27
28          Serial.write(msg, 34);
29          Serial.write(ack, 4);
30      }
31  }
```

Listing B.1: Implementation on the Arduino