A Multi-Threading Software Countermeasure to Mitigate Side Channel Analysis in the Time Domain

Submitted in fulfilment

of the requirements for the degree of

Doctor of Philosophy

of Rhodes University

Ibraheem Frieslaar

Grahamstown, South Africa

July. 2018

Abstract

This research is the first of its kind to investigate the utilisation of a multi-threading software-based countermeasure to mitigate Side Channel Analysis (SCA) attacks, with a particular focus on the AES-128 cryptographic algorithm. This investigation is novel, as there has not been a software-based countermeasure relying on multi-threading to our knowledge. The research has been tested on the Atmel microcontrollers, as well as a more fully featured system in the form of the popular Raspberry Pi that utilises the ARM7 processor.

The main contributions of this research is the introduction of a multi-threading software based countermeasure used to mitigate SCA attacks on both an embedded device and a Raspberry Pi. These threads are comprised of various mathematical operations which are utilised to generate electromagnetic (EM) noise resulting in the obfuscation of the execution of the AES-128 algorithm.

A novel EM noise generator known as the FRIES noise generator is implemented to obfuscate data captured in the EM field. FRIES comprises of hiding the execution of AES-128 algorithm within the EM noise generated by the 512 Secure Hash Algorithm (SHA) from the libcrypto++ and OpenSSL libraries.

In order to evaluate the proposed countermeasure, a novel attack methodology was developed where the entire secret AES-128 encryption key was recovered from a Raspberry Pi, which has not been achieved before. The FRIES noise generator was pitted against this new attack vector and other known noise generators. The results exhibited that the FRIES noise generator withstood this attack whilst other existing techniques still leaked out secret information. The visual location of the AES-128 encryption algorithm in the EM spectrum and key recovery was prevented. These results demonstrated that the proposed multi-threading software based countermeasure was able to be resistant to existing and new forms of attacks, thus verifying that a multi-threading software based countermeasure can serve to mitigate SCA attacks.

ACM Computing Classification System Classification

Thesis classification under the ACM Computing Classification System[1] (2012 version, revised in September 2016, valid through 2017):

Security and privacy – Side-channel analysis and countermeasures;
Cryptanalysis and other attacks;
Tamper–proof and tamper–resistant design.

General-Terms: Electromagnetic, Raspberry Pi, Compilers, CPA, C/C++

[1] http://www.acm.org/about/class/2012/

"In loving memory of my father Abduragman Frieslaar"

Acknowledgements

I would like to thank my parents for ensuring that I receive an education and providing me with the opportunity to seek knowledge. To my mother and relatives, thank you for the endless patience and support whilst I continued with my research, especially after the death of our beloved father. He will always be reminded.

My sincerest appreciation is given to Mr James Connan for having given me the opportunity to enter the research field and empowering me to strive for something greater, as opposed to living a mundane life. Words can not express my gratitude for introducing me to the Professors Barry Irwin and Karen Bradshaw.

My supervisor, Barry Irwin, thank you for undertaking this journey with me and having belief in my abilities. Thank you for pushing me beyond what is required and assisting me to achieve excellent research. It has been a pleasure to have you as a supervisor and receive your words of wisdom.

A personal thank you goes to Dane Brown for all the years of friendship. You are not just a friend, but I see you as a brother. Thank you for the endless advice and most importantly being the benchmark with regards to research ethics, intelligence and physical strength. Your presence has allowed me to better myself in all aspects of life

My dear Amira Damji, thank you for the constant encouragement and enabling me to grow. It has been an epic and wonderful journey by your side. Thank you for being my rock and giving your never ending support.

I would like to thank my lab mates from the "Red" room, especially Sean Pennefather, Deon Person, Brent Shaw and Dalitso Chindipu, for welcoming me into the lab and Grahamstown. From the first day of my arrival, you guys have made me feel at home, which I am deeply grateful for. With the addition of Motse Lehata and Antonio Peters, thank you all, for the trolls, the lulz and the epic moments we have shared together.

The work in this research was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from the Information Security Competency Area within Modelling and Digital Science (MDS) at the CSIR. Financial support from Telkom SA, Coriant, Easttel and Bright Ideas 39 is also acknowledged.

Contents

List of Figures

List of Tables

List of Listings

Seek knowledge from the cradle to the grave.

Prophet Muhammed

1

Introduction

1.1 Background and Motivation

MODERN digital systems rely on computer networks, embedded and specialized equipment to process, store, and transmit information, much of which can be deemed sensitive. As a result, these ecosystems are continuously targeted by adversaries with the intention to gather information for malicious use. Defending these devices is critical as a compromise in one device could lead to a comprise in the entire ecosystem within an organization. Cryptography plays an important role in safeguarding information, thus it is paramount that the cryptographic process is appropriately secured.

The growth of sensitive information produced daily is expanding exponentially, and the challenge arises to secure and protect this information. A major contributing factor to this rapid growth of data is the adaptation of the Internet of Things (IoT). The IoT is the internetworking of embedded devices outside of traditional computing devices with electronics, software, sensors, machinery, and network connectivity that enable these objects to collect and exchange data in real time (Chen, 2012). These devices generally consist of embedded microcontrollers and microprocessors such as the Raspberry Pi[1] that has a microprocessor capable of running a fully fledge Operating System.

[1]Raspberry Pi 2 – https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

These interconnected devices are situated in business, cities, and individuals homes. Examples of these interconnected systems are smart grids (Yun and Yuxin, 2010), where the distribution of electricity is controlled in a real time to ensure that it can be used effectively and efficiently; another example is the control and distribution of natural gas in pipelines (Bonomi *et al.*, 2012). Our homes have many sensors that are increasingly being utilised to monitor regular domestic conditions (Kelly *et al.*, 2013), such as informing the individual when to take out the trash, purchase groceries or digital thermostats that automatically regulates the living conditions. Recently, companies have introduced digital living assistant that has access to the individual's smartphones and all other interconnected devices to assist the individual. Although these devices contribute to the ease of daily activities and increase the financial reward, they pose a potential security threat.

Compromised devices of this type have already been utilised in a number of cyber attacks that rendered major services inaccessible to billions of people worldwide (Langner, 2011, Lee *et al.*, 2014, Button, 2016, Bertino and Islam, 2017). "Smart" devices such as TV's, media players, and internet cameras was demonstrated to be vulnerable to cyber attacks as the adversary was able to remotely control the functionality of these devices for illegal surveillance (Cimpanu, 2017).

The development of applications that serve to safeguard organizations infrastructure, financial and user information is a security concern as vulnerabilities will lead to critical information such as banking details, home security passwords, social security information and much more being at risk of stolen and used for fraudulent activities. Many of these applications are developed with embedded devices that assist the developer in securing sensitive information. Other devices such as Programmable Logic Controllers (PLC) are at risk as well. (Wei *et al.*, 2010, Amin *et al.*, 2013). Interconnected PLCs', allows the automation of electromechanical processes such as those used to control machinery, amusement rides, or centrifuges for separating nuclear material. These devices had been comprised in the 2011 Stuxnet attack (Langner, 2011) when the Iranian nuclear program had been sabotaged, followed in 2014 by the German steel-mill (Lee *et al.*, 2014). Having *open access* to these systems would allow attackers to take control of a nuclear silo base or a strategic military asset capable of mass destruction.

Cryptographic algorithms are one of the means used as a procedure to protect and ensure information is secured. These algorithms conceal information from eavesdroppers. If the adversary comprises the security of a system they can gain unrestricted access to the system. Although the adversary has no legitimate access, the system will regard it as a legal entry and there would not be any record of a breach. Therefore, the protection of secret keys used in the cryptographic algorithm is imperative. Stolen keys would lead to everyone having the ability to access the system. Additionally, this could lead to a major catastrophe to industrial control hardware and federal systems. Once keys are obtained, decrypting sensitive information becomes a trivial task.

Having open or authorised access to these systems would allow attackers to take control of power grids, traffic lights, hospitals, train stations, and much more. are at risk. A compromise in the security of a system would lead to financial and socio-economic disaster, with the possibilities of a compromised system being endless, such an example of such a vast comprise was the cyber attack in 2014 on a German steel-mill (Lee *et al.*, 2014) where critical infrastructure was destroyed. Once the adversaries gained access to the system, they proceeded by taking control of the production management software of the steel mill. Having this level of access, the adversaries had control of the plant's control systems where they methodically destroyed human-machine interaction. This resulted in preventing a blast furnace from initiating its security settings in time and caused serious damage to the infrastructure.

Although the cryptographic algorithms are mathematical secured, i.e: utilising todays technology it would take decades to brute force one password, devices executing these algorithm leaks out sensitive information during runtime operations (Kocher, 1996), typically these leaks are unintentional. It has been demonstrated by Kocher *et al.* (1999) that there is a correlation between the secret key-dependent intermediate variables of the cryptographic algorithm and the power consumption of a device. Upon acquiring this information the adversary is able to retrieve secret information such as the secret key which was utilised to encrypt data from the device, this process is known as Side Channel Analysis (SCA).

Power consumption is not the only form of measurement utilised to recover information, a well known alternative is Electromagnetic (EM) emissions (Gandolfi *et al.*, 2001), other measurements can include temperature, sound, and even photon emissions (Shamir and Tromer, 2004, Brouchier *et al.*, 2009, Schlösser *et al.*, 2012). EM emissions are captured from the device and are subsequently used to retrieve the secret information. EM measurement does not require direct contact with the device and can be performed at long distance without tampering or having physical contact with the victim's device. Therefore EM analysis is less intrusive than the conventional power analysis. Furthermore, SCA has been used against various cryptographic algorithms over the years, such as DES (Mangard *et al.*, 2008), RSA (Finke *et al.*, 2009), and Advance Encryption Standard (AES) (O'Flynn and Chen, 2015).

To mitigate or prevent SCA attacks, various countermeasures have been introduced, more specifically software based countermeasures such as boolean and arithmetic masking; hiding; random pre-charging and shuffling (Coron and Goubin, 2000, Messerges, 2001, Golic and Tymen, 2003, Tillich and Großschädl, 2007). The design of a software countermeasure is extremely challenging as the hardware leaks out information as opposed to the software application. The average software developer does not possess all the knowledge about an embedded device and this could lead to unintentional side channel leakage. Therefore, the dilemma presented to software security developers is to develop secure application while keeping in mind that their applications can be vulnerable to SCA attacks.

The research community has used SCA attacks against microcontrollers (Gandolfi *et al.*, 2001), smartcards (Quisquater and Samyde, 2001), FPGAs (Mulder *et al.*, 2005), and RFID tags (Plos, 2008) particular by analysing the EM emissions to recover secret information. However, the research has demonstrated that other devices such as laptops (Genkin *et al.*, 2014) and smartphones (Genkin *et al.*, 2016) are vulnerable to SCA attacks. It is projected that by 2018, there will be over 23 billion smart or IoT devices worldwide (Statista, 2017). Therefore, 23 billion devices are potentially vulnerable as the former director of the National Security Agency (NSA) has mentioned that IoT technology could be a "security nightmare" (McLaughlin, 2016).

Countless of embedded devices, smartphones and high frequency devices contain hardware capable of executing software in parallel. These devices make use of the process known as multi-threading. Multi-threading is mainly used to increase the performance and runtime execution of an algorithm or program. Therefore, this research investigates the use of multi-threading as a software-based countermeasure against SCA attacks. This investigation is the first of its kind, as there has not been any software-based countermeasure based on multi-threading.

1.2 Scope and Limits

This section discusses the scope of this research and elaborates on the limitations that will be applied to this research. Designing a software-based countermeasure can be complex as there are many variables to take into consideration. Therefore, this research will focus on developing and evaluating existing software-based countermeasure within the time domain.

This research considers the AES-128 block-cipher cryptographic implementation (Daemen and Rijmen, 2002) as the symmetric algorithm under test. The National Institute of Standards and Technology (NIST) has declared that the AES cryptographic implementation is the standard protocol to encrypt information (Heron, 2009). The AES-128 cryptographic algorithm is widely utilised today and is set to be used for many decades as the standard to encrypt information. Therefore, this cryptographic algorithm has been selected as it will be relevant for the foreseeable future and attacks extensible to future variants of the algorithm such as AES-256 and AES-512 (Mohammad *et al.*, 2011).

The target devices utilised in this research are the ATmega328p[2] and ATxmega128D4[3] Atmel microcontrollers and a Raspberry Pi 2 Model B. These devices have been selected as they are readily available, widely used in various scientific research (Kioumars and Tang, 2011, Aziz and Pham, 2013, Kunikowski *et al.*, 2015) and on occasion used as PLC's (Rao

[2] ATmega328p – http://www.microchip.com/wwwproducts/en/ATmega328P
[3] ATxmega128D4 – http://www.microchip.com/wwwproducts/en/ATxmega128D4

and Uma, 2015). By targeting these devices this research aims to enhance the security aspect and prevent known and future attacks. Furthermore, this research explicitly excludes laptops and high performance computers as it is outside the scope of this research as the focus is on devices such as embedded microcontrollers and microprocessors that can potentially be utilised together with critical infrastructure.

With regards to the recovery of secret information, this research will focus on the Correlation Power Analysis (CPA) and template attacks. The CPA approach has been selected as it utilises an enhance power model which allows the intendant attacker to recover all the secret bits via the leakage as opposed to that of the Differential Power Analysis (DPA) approach (Brier *et al.*, 2004) which targets a single bit. This enables the CPA approach to require minimal input data and less preprocessing. Furthermore, the template attack will be utilised as a secondary attack vector as it is a powerful probabilistic SCA attack (Chari *et al.*, 2003).

In order to gain extensive knowledge and develop a greater understanding with regards to the EM emissions leaked from the devices under test, it is more desirable to acquire the EM emissions leaked as close as possible from the device. Therefore, this research will not gather EM emissions via long range antennas but will utilise near-field EM probes to gather information. The devices under test will not be placed in a Faraday cage or be in a confined environment where the EM signals are limited. However, the devices will be placed in an environment where there are other devices such as laptops, desktop computers, music equipment to mention a few, that gives off EM radiation. A general image of the environment is provided in Appendix A.

The security of IoT devices is an imperative aspect today. However, this research primary focus is the development of a software-based countermeasure that utilises multi-threading. Software solutions can be easily modified and rolled out to various devices remotely without the interaction of the end user. However, with a hardware solution, it requires users to return their devices or purchase a more expensive device. Hardware countermeasures can easily become obsolete where as a software approach, can continuously be updated and can provide assurance to the end user that no additional cost would occur.

As mentioned before the primary focus of this research is to develop a software countermeasure. However, to develop and produce a novel software-based countermeasure, new attacks will be introduced by this research. Therefore, a by–product of this work would introduce new attack methodology to recover secret information. These new attack methods are vital in evaluating the novel countermeasure that this research will produce.

1.3 Research Questions

The following research questions are considered:

1. Can a software approach utilising multi-threads serve as a countermeasure against SCA attacks for embedded and a fully featured device?
2. Could the proposed software based countermeasure outperform existing software countermeasures?
3. Would the proposed countermeasure be able to work with other cryptographic libraries?
4. Can the proposed countermeasure be utilised on various devices?
5. Can the proposed software based countermeasure be resistant to a new state of the art attacks?

1.4 Research Objectives

This section describes the research objectives, as accomplishing these objectives would result in answering the proposed research questions mentioned in Section 1.3. The research objectives consist of two phases. The first phase will focus on embedded devices and the second phase will focus on the Raspberry Pi.

1.4.1 Embedded Devices

The objectives for this phase can be summarised as follows:

1. Implement the AES-128 cryptographic algorithm on an embedded device.
2. Perform SCA attacks against the embedded device while:
   (a) An unprotected implementation of the cryptographic algorithm is executing.
   (b) A protected implementation of the cryptographic algorithm with existing countermeasures is executing.
3. Obtain power and EM emissions from the embedded devices to confirm that this research is able utilise EM emissions in a SCA attack.
4. Implement a new software countermeasure based on multi–threads and perform a SCA attack on the proposed countermeasure.
5. Compare and evaluate the results from the unprotected, protected and proposed implementations against the SCA attack.

1.4.2 Raspberry Pi

The objectives for phase two can be summarised as follows:

1. Implement the AES-128 cryptographic algorithm on the Raspberry Pi and perform existing SCA attacks against the Raspberry Pi.
2. Evaluate if the existing techniques can be utilised to recover information from a Raspberry Pi.
3. Extended existing techniques and introduce new approaches to recover the full encryption key.

4. Implement the proposed software based countermeasure of multi-threading and perform an SCA attack against the proposed countermeasure.

5. Evaluate the results and improve the proposed software based countermeasure based on the information from the SCA attack.

6. Design a novel attack to potentially break the improved software-based countermeasure.

By following the above mentioned objectives, this research aims to develop a software-based countermeasure that can execute alongside existing cryptographic algorithms without the need to modify existing algorithms.

1.5 Document Conventions

The remainder of this document, as a general rule, URLs pertaining to websites, organisations, software or hardware mentioned, are provided as footnotes. The reasoning behind this is to minimise the break in the flow of the document and to allow readers with quick access to the information, rather than having to look up the relevant information in the References section of this document. URLs in footnotes that breaks to a new line due to the length of the URL, will be presented in a shortened form to ease readability. A full list of these shortened URLs can be found in Appendix B.

The electronic version of this document contains click-able hyperlinks for chapters, figures, listings, section references and tables. Citations in the text body are also hyperlinked to the appropriate entries in the References section. Bold italicised text within this document refers to specific lines of source code and italicised text accompanied by a brace, refers to functions within an algorithm, i.e: *MixColumns()*.

1.6 Document Structure

The remainder of this document is organised as follows:

Chapter 2 *Cryptography and Cryptanalysis:* This chapter describes basic cryptography concepts and background information regarding cryptography and cryptanalysis.

Chapter 3 *Side Channel Analysis:* This chapter provides an introduction to Side Channel Analysis (SCA) in Section 3.1, followed by the basic concepts of quantifying power, and discusses techniques to gain understanding into the device operations based on the power and electromagnetic emissions leaked from the device in Sections 3.3 – 3.7. In addition, Sections 3.8 reviews techniques utilised to implement a SCA attack, followed Section 3.9 that discusses existing countermeasures to mitigate these forms of SCA attacks.

Chapter 4 *Techniques and Approaches:* This chapter discusses the techniques and methods this research uses to extract information from electromagnetic signals and transform it into meaningful data.

Chapter 5 *Smartcards and Microcontrollers:* This chapter describes the experiments and results carried out on embedded devices.

Chapter 6 *Raspberry Pi:* This chapter describes the experiments and results performed with the Raspberry Pi.

Chapter 7 *Discussion:* This chapter consolidates and reflects on the significance and importance of the work carried out in this research and relates back to the research questions and objectives.

Chapter 8 *Conclusion:* This chapter highlights the novel contributions made towards the research area and provides directions for future work.

The document is concluded with a number of Appendices containing supplemental information. These are referred to within the main body of the text.

There are two kinds of cryptography in this world: cryptography that will stop your kid sister from reading your files, and cryptography that will stop major governments from reading your files.

Bruce Schneier

2

Cryptography and Cryptanalysis

CRYPTOGRAPHY is the art and science of encoding a message or information in such a way that only authorized personnel are able to access the information (Ferguson, 2003). The process to obfuscate information is known as encryption. The modern era has transformed cryptography into a vast discipline covering the domains of mathematics, computer science, and electrical engineering. This chapter introduces basic cryptography concepts and background information about cryptography and cryptanalysis.

The history of cryptography, including classical and mechanical cryptography are discussed in Section 2.1, this enables the reader to gain an understanding that cryptography has been around since ancient times and that it's importance is still viable today. Modern day digital cryptography is followed in Section 2.2 where the concepts of encryption, authentication, and public-key encryption are detailed. Block ciphers are discussed in Section 2.3, more specifically the Advance Encryption Standard (AES) block cipher. A brief description of the secure hash algorithms are mentioned in Section 2.4 and, finally the chapter is concluded with the cryptanalysis section and a summary of this chapter in Sections 2.5 and 2.6, respectively. Readers familiar with these concepts can proceed to Chapter 3.

2.1 History of Cryptography

The use of cryptography dated back thousands of years ago, as far back to 1500 BCE (Kahn, 1996a). Before the 20th century, the means to protect secret information was done by pen and paper, in some case by mechanical aid. However, as complex mechanical and electromechanical machines advanced more sophisticated ways to encrypt information arose such as the Enigma rotor machine. More elaborate and elegant algorithms or techniques were introduced to secure information as electronics and computing progressed.

2.1.1 Classic Cryptography

Transposition ciphers were the desired means to encrypt information at the start of the classic area. These ciphers, shuffles the order of letters in a message an example of is the reorganization of the message *"hello world"* to *"ehlol owrdl"*. Additionally, substitution ciphers which replace letters with other letters were used (Russell and Gangemi, 2006). An example of this method is to replace each letter with another letter that follows the original letter's order i.e: *"fly at once"* to *"gmz bu podf"*. A popular substitution technique used was Caesar's cipher (Churchhouse, 2002). This involves replacing the original letter with a new letter by a fixed number of positions further down the alphabet. Caesar used a shift of three.

The Greeks followed a different approach by concealing the fact that the information exists, this is known as steganography (Kahn, 1996b). An example of this approach was when a secret message got tattooed on a slave's shaved head. The slave would grow his hair and conceal the message under the regrown hair. However, steganography is the science of hiding information whereas the objective of cryptography is to make the information unreadable by an eavesdropper.

The Duke of Mantua introduced the homophonic substitution cipher (Oranchak, 2008). It integrates monoalphabetic and polyalphabetic features by replacing each letter with numerous symbols base on the letter frequency i.e: the letter *"E"* could be replaced by five different symbols, while the letter *"Q"* may only be substituted by one symbol.

Leon Battista Alberti was the first to document the use of polyalphabetic cipher. A metal cipher disc was used to switch between cipher alphabets. The switch to using a different alphabet only occurred after several words. Additionally, the location of the switch was documented as the letter of the corresponding alphabet in the ciphertexts. Johannes Trithemius, built on this work an introduced the tabula recta (Wrixon, 2005). The tabula recta was a square table of alphabets, each row was made by shifting the previous one to the left. The tabula recta was later used as the fundamental structure of Vigenère cipher (Bruen and Forcinito, 2005). The Vigenère cipher encrypts information by using a series of mixed Caesar ciphers based on the letters of a keyword. The Vigenère cipher was considered unbreakable for 300 years.

2.1.2 Mechanical Cryptography

Mechanical Ciphers are those that were developed around the second World War, which relies on sophisticated gearing mechanisms to encrypt messages. A well known historical example is the enigma encryption machines (Kozaczuk, 1984).

During World War II the Enigma cipher was used by the Germans. The enigma cipher refers to a range of similar cipher machines. Arthur Scherbius, invented the first enigma machine at the end of World War I and sold it commercially in the 1920's (Kruh and Deavours, 2002). Many nations adopted it in their military, most famously Nazi Germany. Although there are many models of the Enigma machine, the most frequently discussed model was the German military model, the Wehrmacht Enigma (Kozaczuk, 1984).

The Enigma machine is a combination of mechanical and electrical subsystems. The mechanical aspects consist of a keyboard, rotors, and stepping components. A message was scramble using rotors, scramblers, and plugboard. The rotors would output a different character of the alphabet based on the configuration as each rotor could output 26 different characters. The scramblers had six different combinations. All these settings were used in conjunction with plugboard. The number of different settings was approximately 158 million million million. Therefore, to decrypt the message the user had to have the exact configuration of the machine.

2.2 Modern Cryptography

Modern cryptography is a combination of various concepts of mathematics, computational-complexity theory, and probability theory. The secrecy is based on binary bit sequences as opposed to the historical schemes of using alpha-numeric characters. The entire scheme is not shared as a secret but only the secret key is required to decrypt a message.

Cryptography is often compared to a lock (Ferguson, 2003). In a larger system, the use of cryptography is extremely important as it has to allow and deny access to certain individuals. The challenging aspect of cryptography is to determine whether the individual gained access in a legitimate manner. Therefore, this section elaborates on the basic cryptographic concepts.

2.2.1 Encryption

The basic objective of cryptography is encryption (Menezes *et al.*, 1996). Figure 2.1 illustrates a basic setup of communication as *Alice* and *Bob* wants to send a message between each other. However, Eve is eavesdropping on the conversation and any messages m between *Alice* and *Bob* would be received by *Eve*.

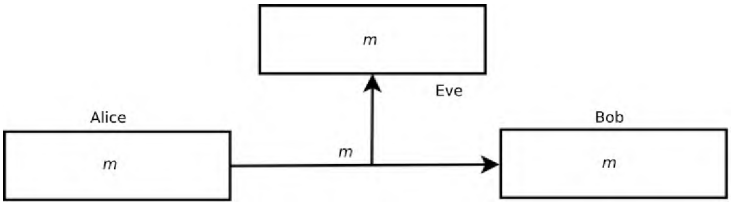


Figure 2.1: Communications between *Alice* and *Bob*, with *Eve* as the eavesdropper.

To prevent messages from being intercepted by Eve, *Alice* and *Bob* uses encryption as seen in Figure 2.1. Before encryption takes place *Alice* and *Bob* must agree on a secret key (K_e) that they will use to encrypt a message.

As *Alice* is about to send message m she uses the encryption function with the secret key to encrypt the message. The encryption function is written as $E(K_e, m)$ with an output

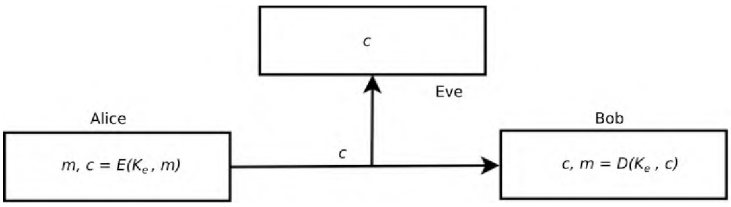


Figure 2.2: Generic technique for encryption.

of c. *Alice* would send *Bob* c, instead of sending m. As *Bob* receives the ciphertext $c = E(K_e, m)$, he decrypts the message using the decryption function $D(K_e, c)$ to uncover the original message m.

As *Eve* intercepts the communication between *Alice* and *Bob*, she would only receive c. However, she does not know the secret key K_e and can not use the decryption function to recover the original message m.

2.2.2 Kerckhoff's Principle

To decrypt the ciphertext Bob requires two elements. These two elements are the algorithm D and secret key K_e used to encrypt the message. Kerckhoff's Principle (Simmons, 1985) states that the security of the cryptosystem must only depend on the secrecy of the key and not the secrecy of the algorithm.

Kerckhoff's Principle is reliant as it is harder to change or keep secret an algorithm as opposed to a key. In the practical world, the same algorithms are used for decades where a key change can occur at any moment in time. A cryptosystem is not built for two users. However, it is built on a large scale and each individual of the crypto ecosystem make use of the same algorithm. Therefore, the algorithm should be public knowledge as it would be easy to steal the algorithm.

The algorithms should remain public knowledge as it is easy to make mistakes and a create an algorithm with flaws. Additionally, nobody would find fault until the point in which the algorithm is exploited and used in an attack, by then it would be too late thus, Secret algorithms are often insecure and flawed.

2.2.3 Authentication

Looking back at Figure 2.1 *Alice* and *Bob* need to address another challenge. The challenge *Bob* has, is to ensure that the message originated from *Alice*.

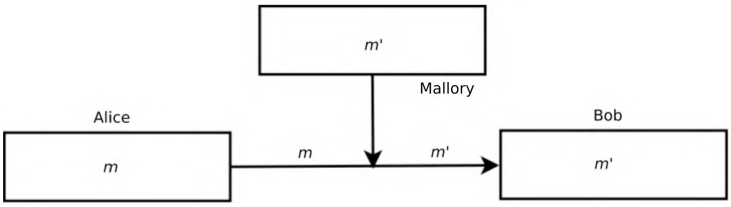


Figure 2.3: *Mallory* manipulates the message.

Figure 2.3 illustrates *Alice* sending *Bob* message m. However, *Mallory* intercepts the message and sends Bob m'. Additionally, *Mallory* could easily prevent *Alice* from sending a message to *Bob* or changing the order of the messages. *Bob* has no reason to believe that any messages originated from *Alice*, thus authentication is introduced as a means to ensure that the messages were from *Alice*.

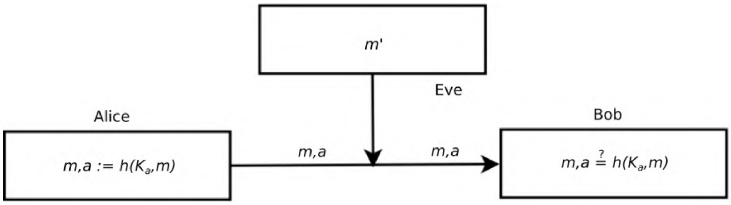


Figure 2.4: Generic technique for authentication.

Figure 2.4 depicts the default process for authenticating a message. Before message m is sent from *Alice*, she computes a Message Authentication Code (MAC). The MAC is defined as $a := h(K_a, m)$; where K_a is the authentication key and h is the MAC function. *Alice* sends *Bob* both a and m in order to recover the original message by determining what a was by using K_a.

In the event of Eve disrupting communications and replacing m with m', *Bob* would compare $h(K_a, m')$ with a, and determine that a is an incorrect message and the communications has been compromised. However, *Eve* is still capable of removing messages, changing the order, and repeating old messages. Therefore, a sequential numbering sys-

tem is combined with authentication. This allows *Bob* to recognize if any messages have been comprised.

It is important to note that the encrypted message can still be manipulated and the authenticated message is not secret. However, using these two techniques in combination will increase in the secrecy of the message.

2.2.4 Public-Key Encryption

As mentioned in Subsec 2.2.1. the secret key K_e is shared by *Alice* and *Bob*. However, *Alice* is unable to send the key to *Bob*, as *Eve* could have intercepted the key as well. Therefore, the distribution and managing of keys are critically important in the design of secure communications. A solution to this problem is public-key encryption.

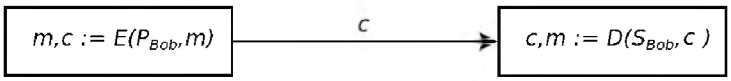


Figure 2.5: Generic technique for public-key encryption.

Figure 2.5 demonstrates an example of a default scenario as public-key encryption is used. The figure is similar to that of Figure 2.2. However, from this point, the attacker Eve is not depicted, as it assumed that *Eve* would constantly intercept information. A significant difference is shown in Figure 2.5 as *Alice* and *Bob* uses different keys.

Firstly, a pair of keys (S_{bob}, P_{bob}) is generated by *Bob*. S_{bob} is the secret key and P_{bob} is the public key. Bob reveals his public key to everyone. *Alice* uses the public key P_{bob} to encrypt the message m to obtain ciphertext c. She sends the ciphertext to *Bob*. Bob uses his secret key S_{bob} to decrypt the ciphertext. In order for this process to be a success, the key generation, encryption, and decryption algorithm have to yield the original message. Therefore, $D(S_{bob}, E(P_{bob}, m)) = m$ must hold for all possible messages. This is known as asymmetric-key encryption as opposed to symmetric encryption which was discussed earlier.

Key management is made simpler by using public-key cryptography. However, *Alice* needs to verify that the key came from *Bob*. A general solution is to use Public Key

Infrastructure (PKI). This is built on having a Certificate Authority (CA). Each user registers and links their specific public key with the CA.

Asymmetric-key encryption depends on maths and it is imperative that it should not be possible to compute the secret key based on a corresponding public key. Although this is a good approach, public-key encryption is less efficient by several orders of magnitude. In practical systems, symmetric and asymmetric encryption techniques are used in combination with each other. Therefore, the flexibility of public-key cryptography and the efficiency of symmetric-key cryptography is achieved.

2.2.5 Cryptographic Attacks

This section discusses potential attacks against the cryptographic system. The most basic method of attack is brute force attack i.e: attempting every possible combination of the secret key. Therefore, the length of the key determines the number of possible keys, and hence the feasibility of this approach.

The most challenging type of attack is a *cipher-text only* attack. In this scenario, the attacker only has the ciphertext as information to break into the system. Attempting to decrypt information with only the ciphertext is known as a *cipher-text only* attack.

A second type of attack is the *known plaintext* attack (Matsui, 1994). This type of attack involves having both the plaintext and ciphertext in order to recover the secret key. Moreover, the *known plaintext* attack is more powerful than *cipher-text only* attack as the attacker has more information to work with.

A more powerful attack is the *chosen plaintext* attack (Biham and Shamir, 1993). This allows the attacker to select predefined plaintext as input messages and receive its corresponding ciphertext. Using this approach grants the attacker a greater chance to recover the secret key. Using *chosen plaintext* on its own can be hard and tedious. However, combining this with statistical modelling and side channel analysis can provide the attacker with the secret key. This process will be further explained in Chapter 3.

2.3 Block Ciphers

A block cipher is an encryption function for fixed-size blocks (Knudsen, 1994). The most a block size can be is 128 bits and the key length can be 128, 192 and 256 bits. These block ciphers encrypt 128-bit of plaintext into 128-bit ciphertext. Additionally, the decryption process must reverse the function to output the original 128-bit plaintext.

This research will consider the AES-128 block-cipher cryptographic implementation as the symmetric algorithm under test. The AES-128 cryptographic algorithm is widely utilized today and is set to be used for many decades as the standard to encrypt information. Therefore, this cryptographic algorithm has been selected as it will be relevant for the foreseeable future.

Prior to the AES-128 block-cipher, there was the Data Encryption Standard (DES) algorithm (Standard, Data Encryption and others, 1977) which had a 56-bit block size. Although now considered insecure, it was highly influential in the advancement of modern cryptography. However, there are still systems today that continue to implement this legacy cryptographic algorithm. A slightly improved implementation known as Triple Data Encryption Standard (3DES) which consist of a 64-bit block size (Vaudenay, 2006) is currently utilised today within financial institutions.

2.3.1 AES

The Advanced Encryption Standard (AES) (Daemen and Rijmen, 2002), more formally known as Rijndael is a symmetric-key algorithm, thus the same key is used for both encryption and decryption. In 2001 the U.S Nation Institute of Standards and Technology (NIST) declared that AES would be the specification to encrypt electronic data, and replacing the DES algorithm (Standard, Data Encryption and others, 1977).

Figure 2.6 depicts one round of the AES-128 implementation. The process consists of four main steps. The *SubBytes()*. *ShiftRows()*, *MixColumns()*, and *AddRoundKey()* steps.

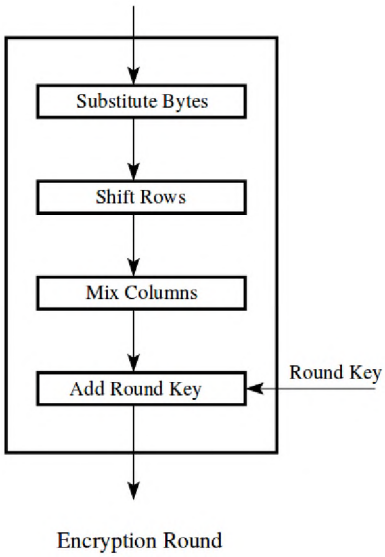


Figure 2.6: One round of the AES encryption algorithm.

The process begins with an initial round where each 16-byte plaintext is combined with a block of the round key using bitwise *XOR* ($\oplus$). This is followed by, the *SubBytes()* process as an 8-bit substitution box or *S-box* (lookup table) is used to replace each byte of the output from the initial round. The result of this process is in a matrix of four rows and four columns.

The *ShiftRows* procedure shifts the bytes in each row to the left. Any entries that "fall off" are reinserted on the right side of row. The *ShiftRows()* is carried out as follows:

1. Firstly, row one is not shifted.
2. Each byte in the second row is shifted one position to the left.
3. The third row is shifted two positions to the left.
4. Finally, the fourth row is shifted three positions to the left.

This results in each column of the output state of the *ShiftRows()* composing of bytes from each column of the input state as the new matrix.

In the *MixColumns* round, each column of the four bytes is transformed using linear transformation. This function takes the four bytes as the input of one column and outputs

four completely new bytes, which replace the original column. The result is a new matrix consisting of 16 new bytes.

The process is concluded with the *AddRoundKey* step. The 16 bytes of the matrix are XORed with the 128 bits of the round key. If this is the last round then the output is the *ciphertext*.

The focus of this process is extremely relevant and important as Side Channel Analysis (SCA) attacks against the AES-128 encryption algorithm targets these rounds (Kocher *et al.*, 1999). It has been demonstrated that there is a correlation between the power consumption of the device as each byte of the output from the initial round is replaced with the *S-box*. This correlation is utilised to recover 16 *subkeys* which relates to the full encryption key. Other attacks targets the *MixColumns* round (O'Flynn and Chen, 2015) to retrieve secret information. More details are provided in Section 2.5 and Chapter 3. The literature has demonstrated that mathematical attacks (Bernstein, 2005, Bonneau and Mironov, 2006) target these rounds as well.

The process of decrypting the AES ciphertext is similar to the encryption process in the reverse order. Each round consists of the same four processes mentioned above conducted in the reverse order. It is noted that the encryption and decryption algorithms need to be separately implemented, although they are very closely related.

2.4 Hash Functions

This research has selected hash functions as a candidate for the software based countermeasure. Therefore, this section will discuss the basic concepts of hash functions and the intended hash algorithms to be utilized within this research as a software-based countermeasure.

A hash function maps input data to a new data string that consists of a fixed size (Katz and Lindell, 2014). Hash functions are flexible as it can be used for encryption, authentication and even as a digital signature (Carter and Wegman, 1977). Hash functions can be

referred to the message digest. There are numerous applications for hash functions within cryptography as it can be utilized to generate pseudorandom values based on the secret key.

Hash functions are commonly utilized in digital signatures. A potential approach is to sign the message m. However, signing m itself is computationally expensive. Therefore, a hash function h can be applied to the message $h(m)$. The output of h can range between $128 - 512$ bits whereas the singing of m alone would resultant in thousands of bits, as the message could be hundreds or thousands of characters.

One of the characteristics of the hash functions is that it should be a one-way function i.e: the output cannot be reversed to recover the original message. The hash function should be resistant to collision attacks. A collision is when two different messages end up having the same hash (Katz and Lindell, 2014).

There are many hash functions available. However, this research will focus on the Secure Hashing Algorithms (SHA), more specifically the SHA–1 and SHA–2 algorithms. The SHA are a family of cryptographic functions published by the National Institute of Standards and Technology (NIST) (Eastlake and Jones, 2001). There is also the SHA–3 family of algorithms or more formally known as Keccak[1]. However, this research will not consider SHA–3 as SHA-3 differs internally from SHA–1 and SHA–2.

The basic concept of SHA is to transform plain text into secure data by utilizing a cryptographic hash function. A cryptographic hash function is a mathematical algorithm consisting of bitwise operations, modular additions, and compression functions that map the plain text to a bit string of a fixed size (Preneel, 1994).

Each SHA algorithm was designed to increase security and prevent attacks against the previous version. Table 2.1 displays the difference between the algorithms of interest. An example is the SHA–0 algorithm which is now obsolete due to the widely exposed vulnerabilities (Biham *et al.*, 2005). The algorithms of interest in this research are the SHA–1, SHA–224, SHA–256, and the SHA–512 hash algorithms. The SHA algorithms ranging from SHA–224 to SHA–512 are part of the SHA–2 family.

[1]Keccak – `https://keccak.team/index.html`

Table 2.1: Basic information regarding the hash functions.

Algorithm	Output Size (bits)	Block Size	Max Input Size
SHA-1	160	512	$2^{64} - 1$
SHA-224	224	512	$2^{64} - 1$
SHA-256	356		
SHA-384	384	1024	$2^{128} - 1$
SHA-512	512		

There has not been a software-based countermeasure that has applied the SHA to generate noise to obfuscate SCA attacks. Therefore, this research will be the first of its kind to embark on the process of determining if the SHA is a suitable candidate to serve as a noise generator. In addition, the SHA family can be implemented on various platforms and devices.

2.5 Cryptanalysis

Cryptanalysis is used to break ciphers or breach cryptographic security systems and reveal the contents of encrypted messages, even though the secret key is not known. As mentioned in Section 2.2.5 one of the first approaches is a brute force attack, this approach guesses the secret key by attempting all possible combination of the secret key. However, this approach is infeasible today as cryptographic algorithms have large key lengths, thus a brute force attack would require decades to guess the correct secret key.

This has paved the way for mathematical and statistical analysis. This approach requires the adversary to have extensive knowledge about the cryptographic algorithms and mathematical and statistical knowledge in order to the break or find a weakness in the cryptographic algorithm (Biham and Shamir, 1993). However, adversaries have found new ways to obtain secret information without having to break the cryptographic algorithm. This procedure is known as Side Channel Analysis (SCA) (Kocher, 1996) and is further discussed in detail in Chapter 3.

An SCA attack is an attack based on the information gained from the physical imple-

mentation of a cryptosystem, as opposed to a brute force or a mathematical attack. SCA utilizes, timing information (Kocher, 1996), power consumption (Kocher *et al.*, 1999), electromagnetic leaks (Gandolfi *et al.*, 2001) or even sound (Genkin *et al.*, 2017) as an added source of information which can be exploited to break the system without tampering with the cryptographic algorithm. A deeper discussion is presented in Sections 3.4 – 3.6 where the basics of a side channel analysis attack is presented, followed by statistical attacks that assist in the recovery of secret keys from cryptographic algorithms, more specifically from the AES-128 algorithm, while utilizing the power and electromagnetic emissions from the device.

The cryptanalysis of the classic encryption ciphers is not applicable to the cryptanalysis of modern ciphers. However, if the reader is interested, then the research by Kahn (1996a) makes for an interesting read.

Breaking a cipher does not mean discovering or implementing a new technique to recover the plaintext. In academic cryptography, breaking a cipher is described as finding a weakness in order to exploit the cipher such that the complexity is less than a brute-force attack (Schneier, 2000). Additionally, a majority of the breaks is against the reduced round version of the cipher, which is later extended to the full cipher. Therefore, a break on a reduced-round version is regarded as a published result.

2.6 Summary

This chapter presented a brief history of cryptography, starting from ancient times and leading up to the modern age of cryptography. The basic understanding of modern cryptography and cryptanalysis concepts were discussed.

The process of cryptography consists of two phases known as encryption and decryption. Encryption transforms *plaintext* (information) into *ciphertext* (meaningless information). Additionally, decryption is the process in reverse, thus converting ciphertext into plaintext. To perform encryption or decryption a cipher or cryptographic algorithm is used.

The cipher is used in tandem with a secret key. The secret key is only known by the sender and receiver. This is to ensure that a third party would be unable to decrypt the information.

The chapter continues by mentioning various types of attacks against cryptography, which is followed by the explanation of the AES-128 block cipher that will be a key aspect of this research. The topic of hash function was discussed as it will be utilized at a later stage in creating a software-based countermeasure.

Finally, the chapter is concluded with a discussion on the progression of cryptanalysis which started with brute force attacks, utilizing mathematical knowledge to break or find weakness in the cryptographic algorithm, whilst progressing to side channel analysis where the adversary uses information such as power consumption and electromagnetic emissions to retrieve secret information without breaching the cryptographic algorithm.

More details of the above process are provided in Chapter 3 as it will discus how the information in this chapter, more specifically, the design of the AES-128 cryptographic algorithm is used to locate points of attack and vulnerable areas in the algorithm with regards to the information leaked by the hardware which results in secret information recovered.

Success is not final, failure is not fatal: it is the courage to continue that counts.

Winston Churchill

3

Side Channel Analysis

SIDE Channel Analysis (SCA) is based on the "Side Channel Information" gathered by the adversary. This information can be retrieved from a device as it executes a cryptographic algorithm. The information typically consists of the device power consumption and/or the Electromagnetic (EM) emissions emitted from the device during the encryption process, which is unintentionally leaked. The main concept of SCA is to utilise side channel information to recover secret information leaked during normal operations, more specifically, in the context of this research, the AES-128 secret encryption key without breaking the cryptographic algorithm.

An introduction to SCA is discussed in Section 3.1, this includes a brief history of SCA and Electromagnetic (EM) attacks against embedded and high frequency devices. This is followed by Section 3.2 that elaborates on the basics of quantifying device consumption, a brief overview of microcontrollers and how the architecture is linked to the device consumption. Sections 3.3 – 3.7 discuss specific techniques utilised to capture and measure power and EM consumption that can be used to recover secret information. These techniques range from basic visual characteristic to more advanced statistical methods.

The process to implement a SCA attack is discussed in Section 3.8. This includes data collection and preparation, good measurement setup, additional signal processing techniques to enhance the signal and finally, a method to evaluate the success of the SCA attack.

Section 3.9 elaborates on SCA countermeasures, ranging from hardware software, and EM based countermeasures. Finally, the chapter is concluded with Sections 3.11 and 3.12 that details potential software that is not commonly utilised as a software based countermeasure but can potentially serve as a countermeasure.

3.1 Introduction to Side Channel Analysis

As discussed in Chapter 2, an encryption device is perceived as a unit that receives plaintext input and produces a ciphertext output. Therefore, attacks against cryptographic schemes are based on either knowing the ciphertext or plaintext, alternatively, the adversary could utilise both (Matsui, 1994, Biham and Shamir, 1993).

Encryption devices produce information about the time it takes to execute the cryptographic algorithm and even the subroutines within the algorithm (Kocher, 1996). Device leakage such as power (Kocher *et al.*, 1999), EM (Agrawal *et al.*, 2003), sound (Shamir and Tromer, 2004), temperature (Brouchier *et al.*, 2009), and even photon emissions (Schlösser *et al.*, 2012) contain critical information about the device and the inner workings of the cryptographic algorithm. SCA attacks utilise this information, along with other cryptanalytic techniques, which will be discussed in Sections 3.3 – 3.7 to recover the secret key the device used to encrypt information. As mentioned in Section 1.2 the focus of this research is symmetric cryptography, thus the theme of this section and the subsequent sections in this chapter will focus on SCA attacks against symmetric cryptography.

SCA based attacks are effective and can be mounted quickly (Gandolfi *et al.*, 2001, Brier *et al.*, 2004) as opposed to the more traditional mathematically based attacks (Biham and Shamir, 1993) which require computational power and many years to break a cryptographic algorithm. SCA attacks can be implemented using readily available hardware costing from only a few hundred US dollars. The time required for the attack and analysis depends on the type of SCA attack. The basic techniques to recover information can consist of a few seconds to more advanced techniques consisting of hours.

The first known SCA attack was successfully performed by Kocher (1996) as he demonstrated that by monitoring the timings of the RSA cryptographic algorithm, he was able to retrieve secret information. The work was extended by Kocher *et al.* (1999) as they introduced power analysis attacks, more specifically Differential Power Analysis (DPA). It was demonstrated that the power consumption of the device has a correlation to the functions of the intermediate process of the cryptographic algorithm. This information was further utilised to uncover the secret key used in the encryption algorithm to encrypt information. A more visual representation of the above is presented in Sections 3.3 – 3.5.

As the research into SCA attacks expanded, the utilisation of EM emissions was shown to be a useful source of data while performing SCA attacks. EM data is an extremely attractive target because of its simplicity, low-cost, and efficiency compared to other existing input data such as power and temperature (Gandolfi *et al.*, 2001, Quisquater and Samyde, 2001, Mulder *et al.*, 2005). Although power and EM consumption are generally utilised as input, other sources such as sound, optical emission and using the temperature of the device have been utilised for SCA attacks.

Shamir and Tromer (2004) introduced the first SCA utilising acoustics. It was observed that heat causes mechanical stress and thus in turns generated acoustic noise[1]. This noise was analysed and it was discovered that secret information was leaked out by the CPU. The research by Shamir and Tromer (2004) has set the foundation for there recent work, where they demonstrated that acoustics (Genkin *et al.*, 2017) could be utilised against high frequency devices and recovered secret information from various cryptographic algorithms.

Brouchier *et al.* (2009) demonstrated that capturing temperature from a device, revealed secret information. The cooling fan of the CPU was monitored as the cryptographic algorithm entered various states, the temperature would have different values depending on the subroutines of the algorithm. This information was used to extract bits of the secret password. Hutter and Schmidt (2014) intentionally operated a target device beyond its maximum temperature ratings and introduced exploitable faults due to extensive heating. These faults allowed the researchers to extract private RSA keys from the device.

[1] i.e: in humanly audible spectrum

Other SCA attacks such as those utilising optical emissions were utilised to gain secret keys from the AES-128 cryptographic algorithm (Ferrigno and Hlavac, 2008, Schlösser *et al.*, 2012). Furthermore Skorobogatov (2009) examined the photons emitted from SRAM, EEPROM, and Flash memories to recover information.

The focus of this research lies within the EM spectrum, thus the remainder of this section will discuss the research carried out while utilising EM emissions in SCA attacks. Furthermore, the AES-128 cryptographic algorithm has been extensively researched across multiple devices (Carlier *et al.*, 2004, Oswald *et al.*, 2005, Bogdanov, 2008, Aboulkassimi *et al.*, 2011, Goller and Sigl, 2015, Genkin *et al.*, 2017). and this research focuses is on the AES-128 implementation as it relates back to current and future research within the scope of side channel analysis and cyber security. The experiments and results with the AES-128 cryptographic algorithm in this research can be found in Chapters 5 and 6.

3.1.1 Electromagnetic Attacks

Since the declassification of The National Security Agency (NSA) document by Friedman (1972) the potential of exploiting electromagnetic emanations has increased tremendously. Recent research has demonstrated that SCA based attacks have shifted from microcontrollers to smartphones (Aboulkassimi *et al.*, 2011) and other high frequency embedded devices (Belgarric *et al.*, 2016, Frieslaar and Irwin, 2017d, Genkin *et al.*, 2017).

Quisquater and Samyde (2001) investigated the EM leakage from a microcontroller. The EM field around the microcontroller was monitored and it was demonstrated that information leaks in the EM field. The work carried out in this research merely shows it was possible to utilise EM emissions. However, no cryptographic algorithm was attacked. Furthermore, they introduced the term Differential Electromagnetic Analysis (DEMA).

Gandolfi *et al.* (2001) carried out the first practical attacks against microcontrollers utilising EM emissions. Their research targeted three different microcontrollers from different manufacturers. Both a DPA and DEMA attack was performed and the results were compared to determine whether EM analysis can outperform power analysis.

The first experiment conducted by Gandolfi *et al.* (2001) was on a microcontroller executing the Alleged Comp128[2] encryption algorithm. A total of 256 power and EM traces were captured as the algorithm encrypted the message. These traces were used in the DPA and DEMA attack. Their results indicate that the DPA leaked information when a correct guess occurs and the DEMA displayed the same results as seen in Figure 3.1 by the power and EM spikes for a correct prediction.

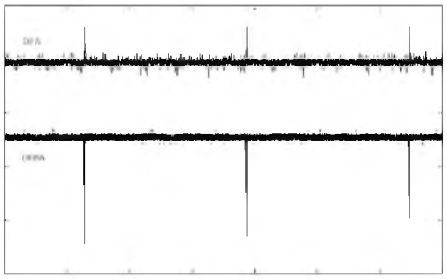


Figure 3.1: Results for the DPA and DEMA attack (Gandolfi *et al.*, 2001).

The next experiment focused on the execution of the DES cryptographic algorithm. The same experimental procedure was followed as before. However, in this scenario, 500 traces were utilised as input. The results demonstrated that the DPA predicted the incorrect value whereas the DEMA attack predicted the correct value as seen in Figure 3.2.

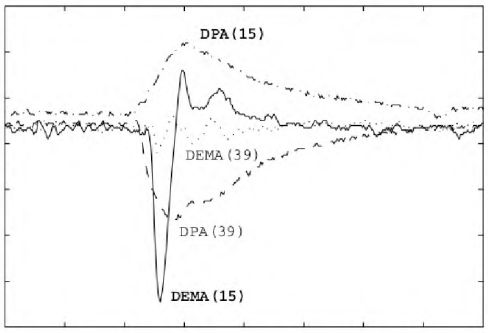


Figure 3.2: DPA and DEMA results against DES (Gandolfi *et al.*, 2001).

Figure 3.2 indicates that the DPA predicted 39 as the correct result. However, the correct value was 15, which was predicted by the DEMA attack. Additionally, it was demon-

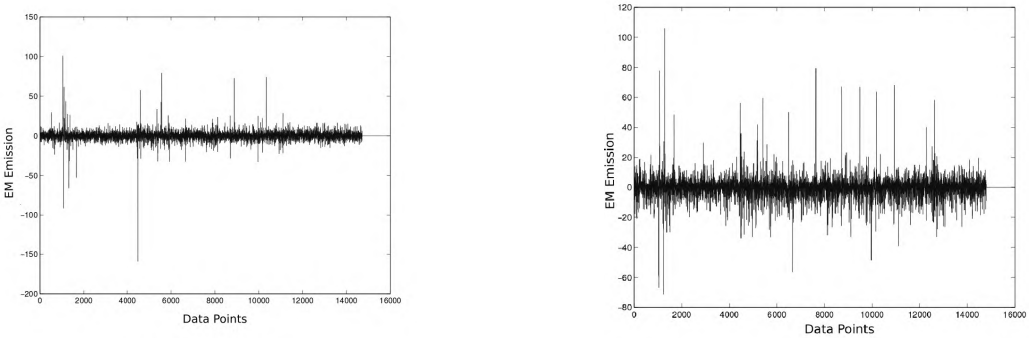
[2]Alleged Comp128 – http://www.scard.org/gsm/

strated that the error rate for DEMA is less than DPA attack as the results for the DEMA reduced the dispersion of peaks.

The final experiment by Gandolfi *et al.* (2001) captured the power and EM traces while the microcontroller executed the RSA cryptographic algorithm. It was observed that the signals produced by the power consumption had more fluctuations as opposed to the EM emissions. They concluded that EM emissions are a great alternative as opposed to power consumption. It was further demonstrated in certain scenarios more information was gathered via the EM field.

Agrawal *et al.* (2003) categorized the EM emissions into two different classes. These emissions are known as the intentional and unintentional EM emissions, respectively. The intentional is caused by the current flow of the device and the unintentional is due to the electric and electromagnetic coupling between different components within a circuit which will be detailed in Section 3.2.

One experiment of note by Agrawal *et al.* (2003) targeted the intentional emission leaked from the device and the results demonstrated that there was no usable information in this area. The next experiment focused on the unintentional emissions. It is revealed that demodulation was applied to the captured signal and the results revealed usable information.



(a) 262 MHz.

(b) 188 MHz.

Figure 3.3: EM emissions at different carrier frequencies Agrawal *et al.* (2003).

An additional experiment was carried out by Agrawal *et al.* (2003) as the EM emissions were captured at different carrier frequencies of 188MHz, 224.5MHz, and 262MHz with an

H-field probe while the DES cryptographic algorithm executed on a smartcard. Figure 3.3 depicts the results of this experiment. It can be seen that at different frequencies, the EM emissions varied in shape and the amount of data points produced.

Carlier *et al.* (2005) successfully demonstrated that they were able to target a Field-programmable gate array (FPGA) and recover secret information via EM emissions. They firstly utilised a simple analysis to locate the occurrence of the AES-128 algorithm by observing the EM emissions visually. Once the exact location was known, the EM emissions at that location were captured and used in a DEMA attack. The results obtained from the DEMA attack was used in an additional attack that targets the *addRoundKey()* and *MixColumns()* operation of the AES-128 algorithm. The results of this experiment demonstrated that they were able to recover the secret information.

Hutter *et al.* (2007) presented work that utilises EM emissions to attack a Radio-frequency identification (RFID) device. They focused on both the software and hardware implementation of the AES-128 algorithm. The attack structure involved capturing 1000 traces and utilising them in the DPA and DEMA attack. Firstly, their results indicated that a DEMA attack was successful and resulted in the recovery of secret information. Secondly, the DEMA had a higher correlation than that of the DPA attack, thus a lower error rate.

Unfortunately, an exhaustive analysis of existing research is not feasible. However, this section is intended to convey the message that electromagnetic emissions are a viable source of data to perform SCA attacks and successfully recover secret information. Various research has stated DEMA can be more effective than a DPA attack. Moreover, this section has discussed that various cryptographic implementations leak out information in the EM spectrum.

3.1.2 Side Channel Analysis Attacks on High Frequency Devices

EM based attacks are not limited to an embedded device which ranges at a high radio frequency of 3 – 30 MHz. However, it can be used against devices running at frequencies ranging in the very high radio spectrum (30 – 300 MHz) or even in ultra high radio

spectrum (300 MHz – 3 GHz) (Aboulkassimi *et al.*, 2011, Nakano *et al.*, 2014, Genkin *et al.*, 2015). Tables C.1 and C.2 illustrates the various radio frequencies and the abbreviations of frequencies, respectively. This is critical as smartphones and laptops of today starts at a base frequency of 300 MHz and can go up to 2.4 GHz or higher. At these higher frequencies, basic antennas or near-field probes are used to capture the EM signature and uncover secret information. Therefore, this section discusses the SCA attacks involving EM attacks against high frequency devices such as smartphones and systems utilising ARM processors. Only the research that focused on cryptographic attacks will be discussed in this section.

To date, the research into utilising EM information as input for SCA attacks has generally targeted microcontrollers (Gandolfi *et al.*, 2001), smartcards (Quisquater and Samyde, 2001), FPGAs (Mulder *et al.*, 2005), and RFID tags (Plos, 2008). In more recent years, SCA attacks, more particularly using EM emissions has focused on the side channel vulnerabilities of high frequency devices such as laptops (Genkin *et al.*, 2014), smartphones (Belgarric *et al.*, 2016, Genkin *et al.*, 2016).

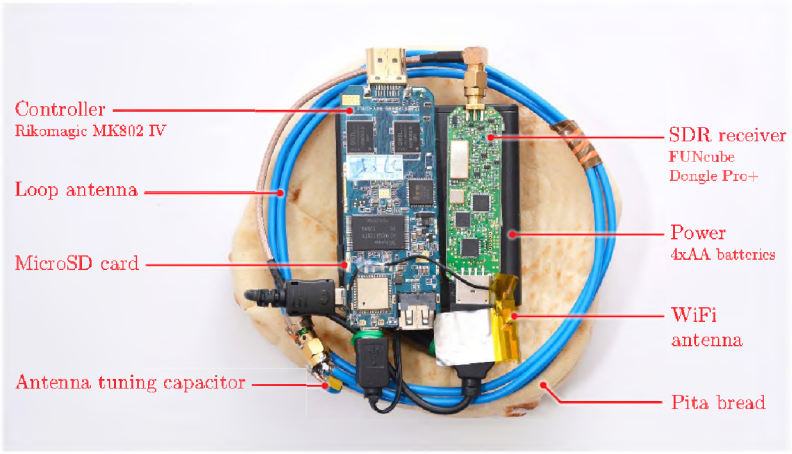
Aboulkassimi *et al.* (2011) performed an EM attack on symmetric ciphers by capturing signals at the device clock rate on a Java based cellphone. However, they placed a MicroSD extension cable to the MicroSD card to extract EM information. Goller and Sigl (2015) implemented attacks on a public key cryptography algorithm against an Android smartphone executing RSA where the smartphone's shielding plate was removed.

A Differential Power Analysis (DPA) attack against the bitsliced AES encryption algorithm executing on a BeagleBone Black Cortex-A8 processor running at 1 GHz on a development board was successfully performed by Balasch *et al.* (2015). Additionally Longo *et al.* (2015) carried out a similar attack against the same development board. Vulnerabilities were demonstrated in both cases against the symmetric key encryption. However, both studies developed and proposed a hardware countermeasure.

Genkin *et al.* (2015) carried out successful SCA attacks on the RSA and ElGamal cryptographic algorithms. They primarily demonstrated that they were able to utilise a Software Defined Radio (SDR) to capture EM emissions from a laptop executing the cryptographic

algorithms. The captured EM data was utilised to recover the secret signing keys from the RSA and ElGamal cryptographic algorithms.

The area of interest is that they created an untethered approach to capture the EM signals remotely by combining the commercially available Funcube Dongle SDR[3] with a TV-tuner. Figures 3.4a and 3.4b illustrates the SDR connected with the TV-tuner to form the Portable Instrument for Trace Acquisition (PITA) and the untethered measurement device setup, respectively.



(a) The PITA

(b) Untethered measurement device setup

Figure 3.4: Untethered setup while utilising the PITA (Genkin *et al.*, 2015).

The PITA (handheld) measures the target computer (left) at a specific frequency band and streams the digitized signal over WiFi, in real time, to the attackers computer (right). The attackers computer can be located anywhere in the world as long as the attack device is in range of the target device. This is extremely important as this work demonstrates

[3]This will be further discussed in Section 4.4.3

that the adversary does not require thousand dollar oscilloscope. Therefore, this research will utilise an SDR to captured EM emissions from a Raspberry Pi.

Successful attacks against the Elliptic Curve Digital Signature Algorithm (ECDSA) implementation in Android's BouncyCastle library[4] was demonstrated by Belgarric *et al.* (2016) and Genkin *et al.* (2016) concurrently. However, Belgarric *et al.* followed an intrusive approach by placing the magnetic probe within the smartphone, whereas Genkin *et al.* situated the magnetic probe in close proximity of the device, hence a less intrusive approach. Genkin *et al.* demonstrated the vulnerabilities of both iOS and Android devices where the ECDSA signing keys from OpenSSL were recovered. Furthermore, secret keys from the CoreBitCoin[5] application were recovered.

Sections 3.3 – 3.7 discusses the techniques utilised in an SCA attack to uncover secret information. It is noted that these techniques can be utilised in any form of SCA attacks such as power (Kocher *et al.*, 1999), EM (Gandolfi *et al.*, 2001), temperature (Brouchier *et al.*, 2009) and sound (Genkin *et al.*, 2014). The difference between the leakage emitted from the device is the manner the emissions are collected. However, the same post processing techniques can be applied across all modalities, it is merely a matter of naming convention i.e for power it would be Simple Power Analysis (SPA) and for EM it would be Simple Electromagnetic Analysis (SEA). Therefore, moving forward this thesis will keep the naming convention as close as possible to electromagnetic.

3.2 Power and Electromagnetic Consumption

Measuring power consumption from a microcontroller can be achieved in numerous methods. One approach is to measure the voltage across a resistor which is connected in series with the microcontroller and the power supply or alternatively, inserting a current probe to the device. Although both approaches would yield similar results, the insertion of the resistor is a cheap and effective solution as the cost of a current probe, at the time of

[4]BouncyCaste Library – https://www.bouncycastle.org/
[5]CoreBitCoin – https://github.com/oleganza/CoreBitcoin

writing, can range from \$700 – \$5000[6], and generally used as lab grade equipment. However, in many cases, the insertion of a resister would alert the victim that his device has been tampered with. Therefore, measuring the EM field around the microcontroller is a good alternative.

3.2.1 Basic Concepts of The Electrical Interaction of a Device

CMOS technology (Allen and Holberg, 2011) has been around for many decades and are used in a number of products, such as microprocessors, microcontrollers, and other digital logic circuits. CMOS devices are extremely popular as they are immune to high noise and utilises minimal static power (Weste and Eshraghian, 1985). In order to fully understand the power dissipation of a CMOS circuit, this section will discuss the dynamic power consumption, the static power consumption, and the power dissipation caused by short circuits (Rabaey *et al.*, 2003). These concepts are important to the understand as the lay the foundation for Side Channel analysis.

The basic design of a CMOS circuitry is that all PMOS transistors must either have an input from the voltage source or from another PMOS transistor. Figure 3.5 illustrates a basic CMOS electrical diagram as the load capacitance on the inverter's output is defined at C_{load}.

(a) CMOS inverter. (b) CMOS charging of load.

Figure 3.5: A CMOS inverter(a) and the CMOS charging load (b).

[6]Current Probe Prices – https://www.tek.com/current-probe

As V_{in} is in a high state, the PMOS transistor is effectively switched off, resulting in shorting V_{out} to the ground which causes C to discharge. The opposite occurs as V_{in} changes from a low state as the load capacitance is switched from zero to V_{dd} which is depicted in Figure 3.5b. The energy (E) consumed during the charging of C_{load}, is derived as follows:

$$E = \int_0^{\infty} V_{dd} i(t) dt = \int_0^{\infty} V_{dd}\, C_{load} \frac{dv_{out}}{dt} dt = \int_0^{V_{dd}} C_{load}\, V_{dd}\, dv_{out} = C_{load} V_{dd}^2 \tag{3.1}$$

The calculation for the dynamic power (P_d) consumption is dependent on the number of instances the CMOS changes from zero to one (Tsui *et al.*, 1993). Therefore, the dynamic power consumption is dependent on the input of the circuit and the following equation is derived:

$$P_d = \alpha f E \tag{3.2}$$

Where α is the switch activity and f is the circuit's clock frequency. Since the power consumption is dependent on the switch activity of the circuit, the data variations of a program executing can be directly correlated to variations in the power consumption. This serves as the basis of power analysis.

An alternative to dynamic power is static and short circuit power consumption. Static power is power consumed while there is no circuit activity. Unfortunately, static power consumption is independent of the input data and generally very small compared to the dynamic power consumption (Kim *et al.*, 2003). Therefore, it is of little interest in the context of power analysis and of this thesis.

Short circuit power dissipation is caused by the effect that both transistors can conduct at the same time, thus resulting in a direct path between the power supply and the ground (Rabe and Nebel, 1996). This is generally utilised in glitch attacks (Anderson and Kuhn, 1998) where the adversary interrupts the power cycle to bypass procedure in the algorithm. This form of attack is direct and intrusive. This approach does not involve EM analysis and falls outside the scope of this research.

3.2.2 Electrical Interactions

This section elaborates on the electrical interactions between internal components of a microcontroller. It is important to have a general understanding of these interactions in order to evaluate and quantify the leaked side channel information that will be utilised for a SCA attack. More details are provided in Sections 3.2.3 – 3.2.5 with regards to capturing and quantifying these electrical interactions.

A microcontroller is a small computer which is generally integrated onto a circuit (Kumar *et al.*, 2011). The microcontroller contains one or more processors, memory, and programmable input/output peripherals amongst others. In comparison with a microprocessor, the microcontroller has far less processing power and emits less heat energy. It is also cheap and easily available, thus making it a perfect test device for this research. The Central Processing Unit (CPU) is the most vital component in the microcontroller. The CPU retrieves machine instructions from memory, decodes it and executes the input instructions (Hennessy and Patterson, 2011). In order to perform these tasks, the CPU consists of the Control Unit (CU), Arithmetic Logic Unit (ALU), memory and registers. The CPU and its components are interconnected through a service known as the bus which transports data to and from the CPU, memory and input/output (I/O) peripherals which are depicted in Figure 3.6.

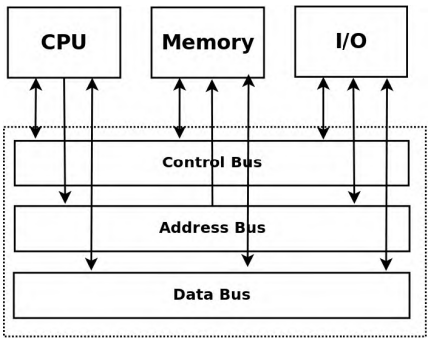


Figure 3.6: Block layout of a microcontroller as data is transported.

When reading from or writing to memory the bus lanes is set to a charged or discharge state, i.e: the lanes are set to one in a charge state and zero in a discharge state. The bus lanes physical characteristics are long and have high capacitive loads which result

in high power consumed as data is transferred. Although a brief overview of the target microcontroller was discussed in this section, it is unnecessary to divulge further into detail as the intrinsics of the targeted microcontroller would prove helpful in other power analysis attacks which falls outside the scope of this work. The details of the microprocessor embedded into the Raspberry Pi 2 B+ (Raspberry Pi, 2015) model is not discussed as the similar logical structure applies as that of a microcontroller with regards to the CPU and transportation of data through bus lines.

3.2.3 Capturing Power and Electromagnetic Emissions

The adversary's main interest is the power/EM consumption that reveals a correlation to the cryptographic algorithm and not the overall consumption. The remainder of this section will discuss utilising a shunt resistor and probing the electromagnetic field to obtain the usable power and EM consumption of the device.

The power consumption of a microcontroller can be calculated by inserting a resistor between the V_{dd} pin and the power supply and measuring the voltage drop. This is referred to as a shunt resistor (Ott, 1988).

Figure 3.7 illustrates a schematic to measure the microcontroller's power consumption with a shunt resistor, where the microcontroller is represented by a generic load with a resistance R_{load} and the power consumption of the device is calculated by measuring V_{load}.

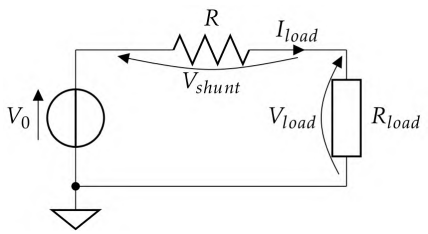


Figure 3.7: Measuring the microcontroller's power consumption with a shunt.

The microcontroller's instantaneous power (P) consumption is calculated as follows:

$$P = R_{load} I_{load}^2 \tag{3.3}$$

By applying Ohm's law ($I = V/R$) the current I_{load} can be calculated as follows:

$$I_{load} = \frac{V_{shunt}}{R} = \frac{V_0 - V_{load}}{R} \tag{3.4}$$

With regards to V_{load}, P is a decreasing function as V_{load} is always less than V_0, while P is proportional to the square of the current. That is, the higher the power consumption, the lower the voltage is at V_{load}.

With regards to power analysis attacks, V_{load} is the value of interest while P and I_{load} are not required. Therefore, R_{load} does not require to be known.

One of the issues of this approach is that additional demand is put onto the measurement circuitry as V_{load} is extremely close to V_0. Therefore, the voltage range must have the ability to include the entire power consumption signal while still maintaining a high resolution which provides acceptable results.

Finally, there are numerous parameters to take into consideration when selecting a shunt resistor. One of these aspects is to know that the shunt resistors have a maximum current rating. The voltage drop at the maximum current rating is used to determine resistance value. i.e: a shunt resistor rated with 100A and 50mV has a resistance of 50/100 = 0.5 Ohm[7]. The voltage drop at maximum current is typically rated 50, 75 or 100 mV. Additionally, the temperature coefficient of resistance and the power rating is a further important aspect to take into consideration, as the amount of electric power that the resistor can dissipate at a given ambient temperature without damaging or changing the resistor parameters is vital.

Capturing the EM emissions emanated from a microcontroller is another approach to find a correlation between the cryptographic algorithm and the device in order to reveal

[7]Ohm is unit of electrical resistance

secret information. As electrical current passes through the device, a magnetic field is generated which varies depending on the structure of the algorithm (Power, 1965). In order to capture this information, an H-field probe can be utilised. An H-field probe is a conducting wire with a coil at the endpoint and it can be seen in Figure 3.8a.

(a) An H-field probe (NewAe, 2017).

(b) An E-field probe (Batronix, 2017).

Figure 3.8: Types of EM probes.

In addition to the H-field probe, there is an E-field probe that is depicted in Figure 3.8b. The E-field probe varies in many aspects compared to the H-field probe and the differences are summarized in Table 3.1.

Table 3.1: Comparison between H-field and E-field probe.

	H-field	**E-field**
Response	Response to the electrical field of the device.	Response to the magnetic field of the device
Field	The electric field is generated by the voltage change.	The magnetic field is generated by the current change.
Shape	It has a loop like structure and has magnetic shielding.	Has the characteristics of a stub shape.

Table 3.1 displays that the E-field probe targets voltage change. However, for this research, the interest lies in the current change. This research is interested in the electrical field, thus it will capture EM information via an H-field probe.

The H-field probe is generally placed in close vicinity of the target device so that the magnetic field induces a voltage through it. This voltage can be measured by an oscilloscope and other devices such as Software Defined Radio (SDR) and other custom device

utilised to capture EM emissions such as the ChipWhisperer which will be discussed in Section 4.4.3.

The Biot–Savart law (Raymond and Jewett, 2013) is applied to determine the EM field generated by an electric current. The equation is as follows:

$$\boldsymbol{B} = \frac{\mu_0 I}{4\pi} \int_c \frac{dl \times \boldsymbol{r}}{|\boldsymbol{r}|^3} \tag{3.5}$$

Where, the magnetic field $\boldsymbol{B}$ at position $\boldsymbol{r}$ is produced by a static electric current I in path C. Additionally, the magnetic constant is μ_0 and dl is a vector whose magnitude is the length $\boldsymbol{r}$. According to Faradays law of induction (Sadiku, 2014), an electromotive force emf is generated within the H-field as variations in the magnetic field occurs. Faraday's law of induction is as follows:

$$emf = -\frac{d}{dt} \int_S \boldsymbol{B} \cdot d\boldsymbol{s} \tag{3.6}$$

Where, S is the surface area of the loop at the end of the H-field probe. As mentioned in Section 3.2.3 the actual power consumption is not required and for the purpose of EM analysis only the voltage measured in the H-field probe is required.

3.2.4 Modeling the Device's Electrical Consumption

In order to perform an SCA attack, it is required to model the power consumption captured from the target device. It has been well established that a device leaks the Hamming weight of the data it manipulates, especially microcontrollers (Kocher, 1996). Although circuit simulations are possible, it is, however, complex and time consuming. Two common models are generally utilised, the Hamming weight and distance models.

The basis of binary models is such that if a condition is true the device will consume more power than when the condition is false. Although, this is a crude approach, as long as the difference can be detected the model serves its purpose. This approach is not limited to

8-bit microcontrollers, it can be applied to a full scale systems with 32/64-bit processors. Experiments with this approach is discussed in Section 5.2.2.

The Hamming weight model consists of summing the number of 1's in a bit string i.e: "111010000", thus the string has a hamming weight of four. The equation for the Hamming weight model (Hamming, 1950) is as follows:

$$HW(B) = \sum_{i=0}^{n-1} b_i \tag{3.7}$$

Where the binary string $B = b_{n-1} \ldots b_1, b_0$. The model is built on the knowledge that power is consumed as a data change occurs as explained in Section 3.2.

The majority of the power consumed by the microcontroller is due to the transportation of data between the memory and registers of the device. Prior to the transportation of data, the bus lines is set to a precharge state. This state allows the bus to be in a phase between one and zero. If the bus is set to zero before the transportation of data, a correlation between the number of ones within the data is achieved. i.e: the number of transitions will be exactly the same as the number of ones. However, the mode would fail to correctly predict the correct power consumption if the precharge state was unknown constant.

The Hamming distance (Hamming, 1950) measures the number of bits as two binary numbers differ, effectively utilising binary substitution. i.e: string $a = 0100$, string $b = 1001$, thus the distance is three.

$$\begin{array}{r} 0\,1\,0\,0 \\ -\,1\,0\,0\,1 \\ \hline 1\,1\,0\,1 \end{array}$$

Figure 3.9: Hamming Distance calculation.

After, the substitution process there are three ones as depicted in Figure 3.9, hence the distance is three. The equation for the Hamming distance (HD) of two binary strings A

and B is as follows:

$$HD(A, B) = \sum_{i-0}^{n-1} |a_i - b_i| = HW(A \oplus B) \tag{3.8}$$

The Hamming distance method is used in scenarios when the precharge state of the device is an unknown variable to the adversary.

The hamming weight and distance concepts are extremely useful as they are utilised for SCA attacks. The process of utilising these concepts will be discussed in Sections 3.3 – 3.6. The SCA experiments in Chapters 5 and 6 are based on these concepts.

3.2.5 Device Consumption Components

As mentioned earlier in Section 3.2, the instantaneous power consumption can be used to recover information. However, the power consumption itself consists of two factors: The leakage and noise signal. The leakage signal is the segment of the power consumption that has usable information and the noise signal is the remainder or the consumption that has no usable information. Therefore, the total power consumption (P) can be defined as:

$$P = P_s + P_n \tag{3.9}$$

Where, P_s is the leakage signal and P_n is the noise signal. The value of the leakage signal (PS) is case dependent on the target exploit. This makes P_s dependent on the type of data produced by the target device.

As mentioned above the noise component contains useless information with regards to a power analysis attack. Electronic and quantization noise are two types of noise that are completely independent of either power models.

Electronic noise can be caused by all components of the device ranging from the transistors in the microcontroller, wiring on the Printed Circuit Board (PCB), and even the measurement equipment. Quantization noise is when an analog signal is converted to its digital format.

The Signal-to-Noise Ratio (SNR) (Collins, 2016) compares the level of a desired signal to the level of background noise, thus if the SNR has a high value, the stronger the signal is. The SNR can be computed by utilising the standard deviation of P_s and P_n:

$$SNR = \frac{\sigma^2_{P_s}}{\sigma^2_{P_n}} \tag{3.10}$$

These noise concepts are taken into consideration to develop a noise generator in this research and will be detailed in Sections 6.4 and 6.4.2.

3.3 Simple Analysis

Simple Power Analysis (SPA) is the technique to interpret the captured power trace of the cryptographic process from the target device. SPA is utilised to gain a more comprehensive understanding of the target device's operations. SPA allows the adversary to examine features that are clearly visible to the naked eye in a single power trace. This section further discusses the concepts of single and trace pair analysis to recover secret information.

3.3.1 Trace Analysis

The first procedure in SPA is to observer the captured trace and draw a conclusion with regard to the operations within the captured trace. These conclusions can correspond to a range of factors such as timing, device attributes, and algorithm structure, amongst others, and more importantly it can be visibly seen.

An example is taken from Kocher *et al.* (2011) as the power consumption was captured from a smart card executing a pseudorandom number generation operation while utilising the Triple Data Encryption Standard (3DES) cryptographic algorithm. Figure 3.10 indicates that variations in the power consumption can convey critical information about the devices operations such as the input phase, the cryptographic procedure and writes

to EEPROM. Each of these procedures can be seen to have a different pattern in terms of power usage.

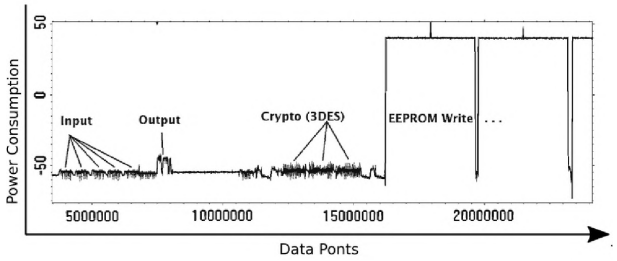


Figure 3.10: Variations in the power consumption Kocher *et al.* (2011).

By locating repeated patterns and counting iterations within loops vital information can be obtained by the adversary, an example of this is depicted in Figure 3.11 as it is observed that there is repeatable pattern looping on occasions. This relates to the eight S-boxes within the first round of the DES encryption scheme (Coppersmith, 1994).

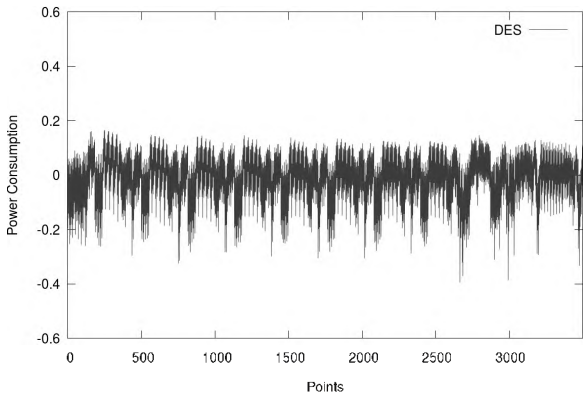


Figure 3.11: Zoomed in view of the DES cryptographic process.

An additional example is depicted in Figure 3.12. Although this is an RSA example, this was the foundation and first example of a successfully SPA attack.

Figure 3.12 illustrates a segment of the power trace from an RSA cryptographic implementation. The trace represents a sequence of squares and multiplications while the target device performed modular exponentiation as the binary left-to-right algorithm was used (Koc, 1995). From the trace, it can be deduced that more power was consumed at the multiplication stage as appose to the square procedure which is indicated by the higher spikes within the trace.

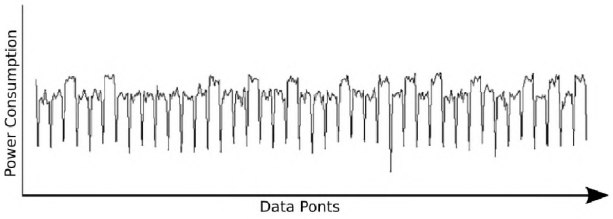


Figure 3.12: Power leakage from an RSA implementation (Kocher, 1996).

Since the binary left-to-right algorithm is publicly available, it is known that for every exponentiation loop one square is performed and multiplication will only be carried out when a bit of the exponent is 1. Having this knowledge, a further conclusion can be reached as each 1 bit in the secret exponent appears as a tall bump, while a 0 bit appears as a shorter bump without a subsequent taller one. Therefore, the bits of the exponent can be recovered and is depicted in Figure 3.13.

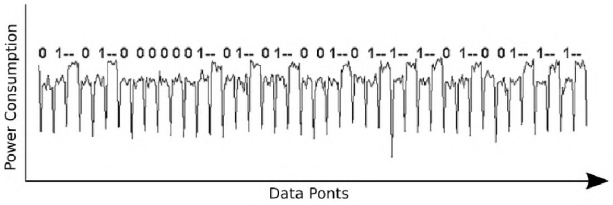


Figure 3.13: SPA revealing the secret bits of the exponent in the RSA algorithm.

The basic concept of trace pair analysis is to compare the captured traces with each other, in order to reveal any similarities or differences between the traces. An example is depicted in Figure 3.14 as the power consumption of a permutation function was calculated as two different inputs were utilised.

(a) Trace 1. (b) Trace 2.

Figure 3.14: Power consumption of a permutation function for two different inputs.

The Figure 3.14 depicts the total time to complete the operation was the same and a similar pattern has been produced. However, by subtracting the two traces from each

other more information can be revealed and can be seen in Figure 3.15.

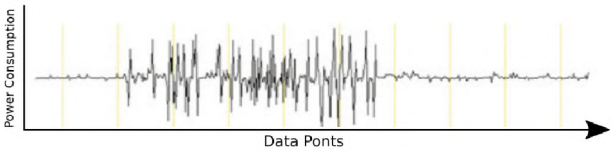


Figure 3.15: Difference between the traces as different input messages was used.

Figure 3.15 display that although the timings were the same, a difference in amplitude and timing can be located by a simple subtraction of the two traces. The adversary can utilise this information and carry out more sophisticated attacks which will be discussed later in this chapter.

3.3.2 Leakage and Attacks

The CPU instruction set can be an additional element in the leakage and aspect of a SCA attack (Mane, 2012), e.g: the multiplication process of an ARM7 where $\pm(32 \times 32)$ would result in a 32 bit output whereas $\pm(64 \times 64)$ results in a 64 bit output[8]. Although it is the same operation more power would be consumed as the outputs are larger. Instructions with only variations in machine code can have visible data-dependent variations in power consumptions. This type of analysis will be further demonstrated in Section 6.3.

Other information can be acquired via an SPA, such as timer interrupts and context switches on multi-threaded CPUs. However, in terms of the cryptographic algorithms, they do not contribute any useful information as they are generally random and not constant. Predefined inputs as discussed in Section 2.2.5 can be utilised to further find a correlation between input and intermediate states of the cryptographic algorithm.

Upon identifying the SPA leakage, a simple analysis can be performed as demonstrated in Section 3.3.1 while utilising one trace, and in other scenarios, a more in depth analysis is required, which will be discussed in the subsequent section.

[8]ARM Multiply and divide instructions – https://goo.gl/Z5cjs4

Typically in a black box assessment, it is extremely difficult to find a correlation between the input and the algorithm. However, to overcome this, collision attacks can be used to exploit the SPA leaks (Schramm *et al.*, 2003, Bogdanov, 2007). A collision attack is aimed at finding two input strings that produce the same result without having a greater understanding of the target device.

An additional approach is to incorporate algebraic attacks with SPA. The algebraic attack involves three stages, firstly expressing the operations of the cipher as an equation, secondly substitute known data into the variables, additionally these known variables can be populated with the use of SPA, and finally solving the equation, which results in the secret key (Renauld *et al.*, 2009, Renauld and Standaert, 2010)

Once the SPA vulnerability has been identified and characterized it is trivial to recover the secret key as the adversary has information about computational intermediates of the algorithm. However, SPA is only really practical when there is a significant correlation between the leakage and cryptographic algorithm. In a well designed system, data-dependent power consumption is well hidden within the noise. SPA attacks may visibly reveal information. However, their interpretation is so tedious that an automated attack is more attractive. Therefore, statistical based attacks are advantageous and will be discussed in the upcoming sections.

3.4 Differential Analysis

Differential Power Analysis (DPA) is a statistical approach that analyses sets of power traces to identify data-dependent correlations. The basic concept is to segment a group of traces into smaller subsets, compute the average of each subset, followed by the calculation of the average of differences between the subsets. Given enough captured traces, extremely tiny correlations can be isolated and thus the adversary would be able to determine secret information. However, this requires thousands, if not millions of traces to determine a correlation.

The most important aspect to DPA is the selection function. This function assigns the captured traces to the subsets and is typically based on a prediction as to a potential value an intermediate value can be within a cryptographic operation. After the average difference are computed the output trace would either reveal large spikes or none in the computed trace. The large spike represents a correlation between the device and value the adversary used and if there are hardly any spikes in the trace, there was no correlation found. Furthermore, a selection function can be the predicted value of a single bit, such as an output bit from an S-box round in the AES-128 encryption algorithm.

In order to gather a better understanding of the process, the DPA attack on the AES-128 algorithm will be discussed as the AES-128 encryption algorithm is a focus of this research. All other statistical approaches in the forthcoming sections will target the AES-128 cryptographic algorithm.

As mentioned in Section 2.3.1 we already know the following information regarding the first round of the AES-128 algorithm (Daemen and Rijmen, 2002):

1. A key schedule is generated from a 16 byte key.
2. The AddRoundKey function XOR's The 16 byte secret with the 16 bytes of the plaintext state.
3. An S-box is utilised to replace each byte of the state in the SubBytes round.
4. The ShiftRows round shuffles each row of the state.
5. Finally, each column in the state is mixed using a linear operation in the Mix-Columns operation.

The target of the DPA attack is the output of AddRoundKey and SubBytes in the AES - 128 algorithm. Figure 3.16 illustrates the a target area of the DPA attack in the AES-128 algorithm. Based on the structure of the AES-128 algorithm, the values can be replaced in an equation format. Per captured trace, the process after the SubBytes round can be defined as I_i.

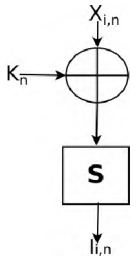


Figure 3.16: Target area of the DPA attack in the AES-128 algorithm.

It is known that the n^{th} byte's value can range between 0 – 15 , thus denoted as $I_{i,n}$. K_n is denoted by the first round n^{th} byte's key and $X_{i,n}$ is denoted by n^{th} byte of plaintext X_i. Based on the algorithm, it is known that $I_{i,n}$ is dependent on one byte of $X_{i,n}$ and K_n. Therefore, the following equation can be derived:

$$I_{i,n} = S[X_{i,n} \oplus K_n] \tag{3.11}$$

Based on the above equation, $X_{i,n}$ is a known variable and K_n is the secret constant with S being the lookup table. The lookup table has been predefined as a standard. Therefore, $I_{i,n}$ is the unknown variable which requires being solved. Moreover, it is known that K_n is an 8-bit value, thus K_n has 256 possible outcomes. It is also known that the AES-128 algorithm has 16 subkeys. Therefore, 16 K_n values are required to be solved in order to retrieve the correct secret key. By having this information, each subkey can be solved separately. It is noted that although each subkey can be solved individually, a full key is required to determine the correct outcome (Martin *et al.*, 2015).

The final step is to determine which solved value is, in fact, the correct subkey. As mentioned in Section 3.3.1 a selection function is required to determine the correctness of a value. Therefore, based on Equation 3.11 to test if a candidate value of K_n is correct, each point j in the differential trace Δ_D for the guess K_n is computed as follows:

$$\Delta_D[j] = \frac{\sum_{i=1}^{m} D(C_i, K_n)T_i[j]}{\sum_{i=1}^{m} D(C_i, K_n)} - \frac{\sum_{i=1}^{m}(1 - D(C_i, K_n))T_i[j]}{\sum_{i=1}^{m}(1 - D(C_i, K_n))} \tag{3.12}$$

Where, $T_i[j]$ is the captured trace at j time and C_i is the known input, with guess K_n.

The differential trace Δ_D that has the largest spike for guessing $K - n$ is considered to be the correct value.

It is noted that C_i can also be the output ciphertext as Kocher *et al.* (2011) demonstrated that utilising C_i as the output ciphertext yielded in the recover bytes of the last round key. There are variants of the to the SPA and DPA attacks, that will be discussed in the subsequent sections.

3.5 Correlation Analysis

Brier *et al.* (2004) demonstrated that Correlation Power Analysis (CPA) can be used as a practical solution to retrieve secret information as it evaluates the degree of correlation between variations within the set of measurements. CPA is utilised with a power model to locate a correlation between the power and one of the intermediate values of the cryptographic algorithm. Depending on the scenario either the Hamming weight or distance model can be used.

It is has been established that the CPA attack outperforms the basic DPA attack (Brier *et al.*, 2004) with regards to the following:

1. The CPA is much faster in processing data since it only requires a few power traces whereas the DPA is slower and needs thousands of power traces.
2. The CPA attack is more accurate at predicting the correct subkeys as it finds a correlation between all the key guesses.

The CPA approach has been selected by this research to predict the secret while utilising power and EM emissions from the microcontroller and Raspberry Pi, the results can be found in Chapters 5 and 6.

The section continues by discussing the mathematical approach of the CPA attack which is followed by the procedure to retrieve the secret key 16 subkeys from the AES-128 algorithm.

It is assumed that the adversary has captured a trace $t_{d,j}$ with d as the total number of traces, and t the time index per power trace. Simply, D measurements with each measurement being T points long are captured by the adversary. If the adversary knows the exact point of the encryption process then only a single point would be measured such that $T = 1$ for each d trace and the plaintext P_d is also known by the adversary.

It is assumed that the microcontroller's power consumption is dependent on the intermediate value's hamming weight, $h_{d,i} = l((P_d, i))$, where a given intermediate value's leakage model is $l(x)$, and $w(p, i)$ produces an intermediate value based on the given input plaintext with a guess number i.

In terms of the AES-128 algorithm, the intermediate value would be selected as follows: each byte of the plaintext is XORed with each subkey byte of the secret key, hence:

$$l(x) = HammingWeight(x)$$

$$w(p, i) = p \oplus i \tag{3.13}$$

This implies that a single byte of the plaintext p is being attacked at a time which relates to the AES-128 key is being attacked a single byte at a time. It has been mentioned in Section 2.3.1 there are 16 subkeys. Therefore, there are only 16×2^8 possibilities, instead of 2^{128} possibilities to predict the entire secret key (Le *et al.*, 2008, Kocher *et al.*, 2011).

The procedure is continued by establishing a linear relationship between the captured power traces $t_{d,j}$ and the predicted power consumption model $l(x)$ using the correlation coefficient. The aim is to ensure that there is a non-linear relationship between $w(p, i)$ and either p or i. Therefore, it is possible to attack the AES-128 algorithm at the point of the non-linear S-Boxes. Finally, the correlation coefficient is calculated for each of the possible subkey values I over all traces D for each point of j, thus the equation is:

$$r_{i,j} = \frac{\sum_{d=1}^{D}[(h_{d,i} - \bar{h_i})(t_{d,j} - \bar{t_j})]}{\sqrt{\sum_{d=1}^{D}(h_{d,i} - \bar{h_i})^2 \sum_{d=1}^{D}(t_{d,j} - \bar{t_j})^2}} \tag{3.14}$$

Where h is the hypothetical values produced by the power consumption model.

Based on the above understanding the secret key of the AES-128 algorithm can be recovered by following four steps (Appendix D.1 – D.4 displays the Python code for these steps.):

1. While the AES-128 algorithm is executing, power or EM traces, along with the input text will be captured.

2. Implement a power leakage model, where the known input text is used with a guess of the key byte.

3. The known values will be substituted into the correlation equation that loops through all the captured power traces.

4. Create a ranking procedure that determines the most likely key based on the correlation equation.

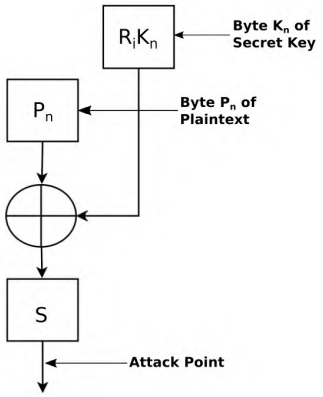


Figure 3.17: Point of attack against the AES-128 algorithm.

It is depicted in Figure 3.17 that each input byte of text is XORed with the secret key which passes through an S-Box. The output of the S-Box is the target area where the attack will take place, this can be seen by the red arrow in the figure. The AES round has been explained in Section 2.3.1.

Upon acquiring the data, a power leakage model. A single subkey is attacked per instance, thus the attack system loops through a single subkey and guesses every possibility for that subkey. The guessed values range from 0 – 255. Once the guess has been calculated, the intermediate value corresponding to the guess needs to be calculated. Therefore, a single byte of input and a single byte of the guessed key is used to return the output of the S-Box. For each guess, the number is converted to binary and the Hamming weight is calculated.

The correlation is calculated by substituting all the acquired data into Equation 3.14. It is noted that a negative correlation is possible. However, the absolute value is taken. To achieve a ranking system the correlation of each guess is stored and the guess with the maximum correlation is predicted to be the subkey. The ranking system this research will utilise to predict a correct subkey is further explained in Section 3.8.4.

3.6 Template Attack

The template attack first introduced by Schindler *et al.* (2005) constructs a model of the device's power consumption. In comparison to the CPA approach, the actual power measurements from the device are utilised to create the model (Köpf and Basin, 2007). The significance of this attack is that the result obtain are extremely accurate and consistent as the attack is based on the actual power draw from the target device (Oswald and Paar, 2011). Statistical parameters are utilised to create a correlation between the measurements and power states of the target device. In order to fully explain the template attack, this section will be segmented into three smaller sections consisting of multivariant statistics, template creation, and applying the attack.

3.6.1 Multivariant Statistics

As mentioned in Section 3.2.5 electrical signals are noisy and the measuring of voltage does not equate to perfect readings. However, it is possible to model the voltage source

by utilising Equation 3.9 with the Probability Density Function (PDF) of a Gaussian distribution (Parzen, 1962). The PDF equation is as follows:

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{P-\mu}{\sigma})^2} \tag{3.15}$$

where, μ and σ is represented by the mean and standard deviation respectively.

The PDF works well for one variable. However, the adversary requires using multiple variables in order to recover the 16 subkeys. To overcome this issue, multivariate distributions can be used (Balakrishnan, 2000). Multivariate distributions allow for the modelling of multiple variables that may or may not be correlated. Instead of determining a signal σ, the system would determine a matrix of covariances. In order to model three variables X_1, X_2, X_3, the matrix would be as follows:

$$\Sigma = \begin{bmatrix} Var(X_1) & Cov(X_1, X_2) & Cov(X_1, X_3) \\ Cov(X_2, X_1) & Var(X_2) & Cov(X_2, X_3) \\ Cov(X_3, X_1) & Cov(X_3, X_2) & Var(X_3) \end{bmatrix} \tag{3.16}$$

Each value mean has to be calculated, therefore:

$$\mu = \begin{bmatrix} \mu X_1 \\ \mu X_2 \\ \mu X_3 \end{bmatrix} \tag{3.17}$$

To calculate the PDF of a multi-variance a vector (v) is required, where $v = [X_1, X_2, X_3 \ldots X_n]^T$ Thus the equation for N variables is:

$$f(x) = \frac{1}{\sqrt{(2\pi)^N |\Sigma|}} e^{-\frac{(v-\mu)^T \sum^{-1} (v-\mu)}{2}} \tag{3.18}$$

Therefore if N points are taken from the power trace and substituted into v from Equation 3.18. with $f(x)$ returning as a high value. The system would interpret this as a good guess for the potential subkey value.

3.6.2 Creating The Template

A template is effectively a set of probability distributions that visually portrays the power consumption as various random keys are used as input. Thus, if key (k) is used as input, the power trace for that key would be the distribution $f_k(x)$. As this research is focused on the AES-128 algorithm, the adversary can model a sensitive section within algorithm. Using the s-box as a reference point, the adversary can create a model of every possible hamming weight, thus only nine models are required.

A template is effectively a set of probability distributions that portrays what the power consumption would likely be for various random keys. Therefore, if the key (k) is used, the power trace of that key would be the distribution $f_k(x)$. In order to create a good distribution to model the power trace for every key, thousands or even millions of traces are required as the attacker would require creating 256 models. However, there is an alternative approach. The adversary can model a sensitive section in the AES-128 algorithm. By focusing on the s-box in the algorithm the adversary can create a model of every possible hamming weight, thus only nine models are required. This reduces the order of magnitude drastically.

The adversary assumes that the selection of the key does not affect the entire power trace. It is considered that the subkeys only influence the power consumption at certain locations. If the attacker is able to determine these points of interest he would only require three points to create a Three-dimensional (3D) distribution model.

The adversary would be able to determine these points of interest by utilising the sum of differences method. The algorithm is as follows: For every sample i with every operation k, determine the average power consumption $M_{k,i}$. Hence, if there are T_k traces where the attacker performed K operations, the average would be calculated as follows:

$$M_{k,i} = \frac{1}{T_k \sum_{j=1}^{T_k} t_{j,i}} \tag{3.19}$$

Upon acquiring the set of averages, the absolute pairwise differences is calculated, followed

by the summation of these values. This will generate a power trace which consists of various peaks as depicted in Figure 3.18.

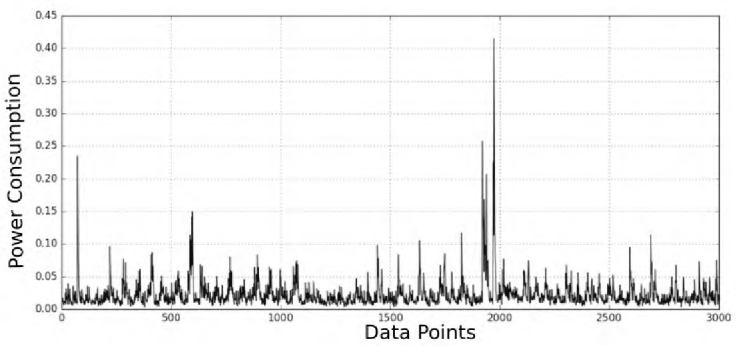


Figure 3.18: Power trace with peaks.

The next procedure would be to select the highest peak, remove minimal peaks close to the highest peak and traverse through the power trace until enough points of interest have been selected.

The adversary's aim would be to determine the mean and covariance matrix for every operation. This is achieved as follows:

1. Locate every power trace that falls under the category of certain operation. Example: locate every power trace where the subkey was 0xFF.
2. Determine the mean at every point of interest.
3. calculate the variance at each point of interest.
4. Determine the covariance of every pair of points of interest.
5. Repeat for all operations.

Following the above procedure the adversary will have K mean and covariance matrices, modelling each of the K different operations that the target can do. In Appendix D the discussed procedures Python code for these snippets is located in Listings D.5 – D.8.

3.6.3 Applying the Template

In order to recover the secret key of the AES-128 algorithm, a small number of traces from the victim's device is required, usually between 1 – 20 traces are required. Upon acquiring these traces the PDF as discussed in Section 3.6.1 is calculated for every key guess based on the location of the points of interest from the template. It is assumed that the adversary has access to a copy of the device he intends to attack. Once the device is obtained, four steps are required to perform the attack.

1. Capture a large data set from the copy device using random plaintext and cryptographic keys as input.
2. Generate a template based on the points of interest which consist of the multivariate distribution of the traces.
3. Capture the traces from the victim's device.
4. Apply the template on the captured trace from the victim's device.

In some cases, the template may be constructed using the actual target device. For example utilising the same smartphone manufactured. Another scenario is to build the template based on the device architecture family, i.e: target the same microcontroller in different PCB's.

3.7 Extended the Side Channel Analysis to AES-256

Although, the focus of this research is the AES-128 cryptographic algorithm. It has been demonstrated by O'Flynn and Chen (2015) that the SCA attack can be extended and target the AES-256 cryptographic algorithm. Since the AES-256 implementation is based on AES-128 the process to generate the ciphertext is similar. Table 3.2 illustrates the key differences between the two algorithm.

Table 3.2: Key differences between the AES-128 and AES-256 cryptographic algorithm.

	AES-128	**AES-256**
Key Length	16 Bytes	32 Bytes
Key Scheduler Length	176 bytes	240 bytes
Round Transformations	10 rounds	14 rounds

As discussed in Section 3.5 it is known that the vulnerability of the AES-128 to SCA attacks is at the *S-box* round. Each *S-box* has one byte. Thus, 16 bytes can be recovered from the *SubBytes* operation. However, for the AES-256 implementation that equates to only half the key recovered and requires an additional attack against the next round.

In this new round, there is an additional operation known as the *MixColumns* round. As explained in Section 2.3.1 *MixColumns* converts four input bytes to four outputs bytes, thus four bytes requires to be predicted instead of one byte. Guessing 4 bytes would become a complicated task. However, a solution is to formulate an equation from the last known state. The state at the end of the 13^{th} round is as follows:

$$X_{13} = SubBytes^{-1}(MixColumns^{-1}(ShiftRows^{-1}(X_{14} \oplus K_{13}))) \tag{3.20}$$

where X_{14} is the output of round 14, with K_{13} the 16 byte round key of round 13. Finally, X_{13} is the output of the 13^{th} round, which will be the point of attack. It is known that *MixColumns* is linear, thus it equates to $\text{MixColumns}(A + B) = \text{MixColumns}(A) + \text{MixColumns}(B)$. Applying this property the hypothetical key can be formulated for the 13^{th} round, which is: $K'_{13} = \text{MixColumns}^{-1}(\text{ShiftRows}^{-1}(k_{13}))$. Finally, the hypothetical key can be used to calculate the output as:

$$X_{13} = SubBytes^{-1}(MixColumns^{-1}(ShiftRows^{-1}(X_{14}) \oplus K'_{13})) \tag{3.21}$$

The hypothetical key can be used to perform a CPA attack by recovering each subkey one byte at a time and the round key can be recovered by calculating:
$K_{13} = MixColumns(ShiftRows(K'_{13}))$. By applying these equations a full attack on the AES-256 implementation can be performed as follows:

1. Perform a CPA attack against the first *S-box* output to recover 16 bytes of the 14^{th} round key.

2. Calculate the second *S-box* by using the 14^{th} round key. Apply this attack point to determine 16 bytes of 13^{th} round hypothetical key by inverting the *MixColumns*.

3. Apply *MixColumns* and *ShiftRows* to the hypothetical round key to give the actual value of the 13^{th}round key.

4. Apply the AES-256 key schedule to reverse the 13^{th} and 14^{th} round key to compute the 256 bit secret encryption key.

Once the above the procedure is carried out, a successful attack on the AES-256 implementation would be completed by attacking two separate AES-256 S-Boxes.

3.8 Implementing a Side Channel Analysis Attack

The various approaches and techniques mentioned previously in this chapter do not necessarily guarantee full key recovery and may completely fail due to countermeasure or noise, which will be discussed in Section 3.9. However, the adversary is able to use other techniques such as trace alignment and digital filtering to compensate for SCA countermeasures and noise generated from the device. The adversary can optimize the data collection to reduce computational time for obtaining secret information. Therefore, this section will discuss techniques that enable the adversary to optimize data collection, gather good measurements, reduce noise within the signal, and construct an evaluation process to determine if the information is usable.

3.8.1 Data Collection and Preparation

Time is always a major factor in breaching a system as the system could be configured to utilise different secret keys each week. Therefore, the adversary is constantly attempting

to reduce the time taken to perform a successful attack. Improving signal quality is only the first step in the process as a higher SNR would result in fewer traces required to obtain the secret information.

The signal quality of a measurement is influenced by various factors such as capturing measurements closer to the Integrated Circuit (IC) that performs the cryptographic operations. Devices with multiple power and ground connections usually have a separate chip that focuses on the cryptographic implementation such as IBM's CryptoCards (IBM, 2017), thus capturing measurements in this area increases the desired signal. Furthermore, the removal of decoupling capacitors (O'Flynn and Chen, 2013) can increase the signal quality.

The signal quality can be further affected by the target devices operating parameters such as the input voltage and the clock rate frequency, to mention a few. Pushing the device to its limit (Bar-El *et al.*, 2006) can result in information leakage and reduce the effectiveness of certain countermeasures. It has been demonstrated that setting the device to a constant frequency (Belgarric *et al.*, 2016, Genkin *et al.*, 2016), secret information can be recovered.

While utilising the various attacks previously mentioned in Sections 3.3 – 3.5, the input messages are usually random. However, it has been shown that chosen input text (Kelsey *et al.*, 2000) – a certain part of the input is kept constant – can reveal additional leakage and simplify the analysis. An alternative is, adaptively chosen input technique (Novak, 2002) can be used. This technique involves carrying out multiple attacks and changing the input text based on the results from the attack.

3.8.2 Measurement

Data collection is generally carried out by utilising an oscilloscope. A digital storage oscilloscopes consist of high speed Analog-to-Digital (ADC)'s, high memory and trigger flexibility to assist in locating the exact moment of an execution. The data collection is regarded as the most time consuming aspect of carrying out an attack. Alternatively,

other devices that have high speed ADC can be utilised to capture data. This research will be using the ChipWhisperer kit to capture power and EM consumption from a microcontroller and a Software Defined Radio (SDR) to capture EM data from a Raspberry Pi, which will be discussed in Section 4.4. It will be demonstrated in Chapters 6 and 5 how these device are utilised by this research to capture power and EM emissions. Moreover, these equipment consists of high speed ADC's and thus, this research does not require, an expensive oscilloscope.

In the case of high frequency devices, the EM analyst can apply additional preprocessing which assists in isolating the signal of interest. This procedure is known as demodulation and is further explained in Section 4.1.1. Similarly, bandwidth limiting or digital filtering can be utilised to remove unwanted signals. The digital filtering technique utilised in this research is discussed in Sections 4.1.3 – 4.2.3.

To limit unnecessary data collected the adversary performs a initial SPA attack to locate the precise execution of the cryptographic functions as discussed in Section 3.3. Alternatively, a DPA or a CPA attack can be performed with known input and output to find the locations of the cryptographic algorithm within the traces. This usually involves utilising the entire captured trace and systematically attacking segments within that trace to locate the cryptographic process.

3.8.3 Signal Processing

Once the traces have been captured and stored in a digital format. Additional signal processing can be applied to improve the effectiveness of a side channel analysis attack (Barenghi *et al.*, 2010).

The most common procedure in signal processing is to align all captured traces. A basic alignment technique is to determine the time shift (Cohen, 1995) between a reference and a captured trace. Based on the reference trace, the other captured traces would be aligned while utilising the time shift factor. More often than not, more complex methods are required to align traces such as Elastic Alignment (van Woudenberg *et al.*, 2011), see

Section 4.2.3. Good alignment reduces the number of traces and computational power required to extract a key.

Traces can often carry unwanted artefacts. However, these effects are easily detected and subtracted from the traces. Additional noise and unwanted data can be removed by capturing multiple traces of the same operation whilst using identical input and then averaging the captured traces.

Finally, a noise reduction algorithm can be utilised to reduce the noise in the traces and increase the SNR factor. The signal processing techniques utilised in this research will be discussed in Chapter 4. Once the traces have been through the additional signal processing procedure, it would be ready to be utilised by statistical attacks mentioned in Sections 3.3 – 3.6.

3.8.4 Evaluation

The results obtained by the various attacks can be evaluated by visual inspection or by an automated process. In terms of DPA attacks, the trace with the highest peak is the correct key guess which can be seen visibly in the data. As mentioned in Sections 3.5 and 3.6 this research has chosen to utilise the CPA and template attack, Python software is used to indicate which subkey has been predicted as the correct key.

SubKeys

PGE Rank	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PGE	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
0	2B 0.8393	7E 0.8829	15 0.8342	16 0.8500	28 0.7452	AE 0.9136	D2 0.8315	A6 0.8711	AA 0.7969	F7 0.8656	15 0.8477	88 0.8983	09 0.8068	CF 0.9082	4F 0.8995	3C 0.8504
1	2A 0.7565	A5 0.7137	FF 0.6737	B8 0.6980	C0 0.7024	9C 0.7077	79 0.7164	08 0.6625	AB 0.7439	B8 0.7497	A2 0.7051	A3 0.6817	68 0.7017	90 0.6941	8B 0.6805	6F 0.7161
2	5D 0.7422	18 0.7017	2F 0.6726	4D 0.6696	3C 0.6811	3D 0.7051	5A 0.7113	18 0.6578	48 0.7024	F2 0.7158	5A 0.6993	3A 0.6813	00 0.7008	12 0.6714	23 0.6793	58 0.7093

Figure 3.19: Results table of the CPA attack.

Figure 3.19 illustrates the output of the results table. The table indicates the correlation accuracy per predicted subkey. The higher the correlation, the higher the subkey would be ranked. A Partial Guessing Entropy (PGE) is displayed in the table if the PGE value is zero, the correct subkey has been predicted.

The PGE is a useful metric of where the correct subkey is ranked. However, this requires the knowledge of the actual encryption key used during encryption. As mentioned in Section 3.5 the CPA attack stores the results in an array which has the predicted subkey. The location of the actual subkey is located within this array, once the location is returned the PGE shall receive its value (O'Flynn and Chen, 2014).

In other scenarios such as performing multiple iterations of the attack (O'Flynn and Chen, 2015), two separate rounds in the AES-258 cryptographic implementation has to be attacked, before the entire secret key can be predicted.

3.9 Side Channel Analysis Countermeasures

Based on the related research it has widely been accepted that a single and efficient countermeasure cannot provide complete protection against a large variety of SCA attacks. This section will briefly discuss hardware countermeasures, software countermeasures, an approach to quantify the software countermeasure, electromagnetic countermeasures and finally, a look at the various software that could potentially serve as countermeasures against SCA attacks.

3.9.1 Generic Hardware Countermeasures

This section will discuss the generic hardware countermeasures. Although there are many hardware solutions, they are mostly based on common practised solutions which will be discussed.

One of the most common hardware countermeasures against SCA is Random Process Interrupts (RPIs) (Clavier *et al.*, 2000, van Woudenberg *et al.*, 2011). Firstly, the CPU does not execute code execution sequentially, it interleaves the codes execution with dummy instructions such that corresponding operation cycles do not match because of time shifts.

A second approach is to generate white noise. This approaches commonly utilises the device to generate additional electrical current as the algorithm executes. There are various solutions to generate a hardware noise generator (Hörmann and Leydold, 2003, Chen *et al.*, 2004, Lee *et al.*, 2004, 2005).

There are additional hardware countermeasures such as internally integrated filters (Ratanpal *et al.*, 2004), voltage regulators (Telandro *et al.*, 2006, 2009) or a current mask generator (Mesquita *et al.*, 2005, 2007). These are merely the most common hardware countermeasures. Although, the hardware countermeasure is outside the scope of this research, for completeness a brief overview was presented.

3.9.2 Software Countermeasures

As previously introduced in Section 3.8, the first step in executing an SCA attack is to perform a SPA attack. The SPA assists the adversary to accurately determine the location of specific operations within the cryptographic algorithm. Therefore, the first step in a creating a countermeasure is to mitigate the SPA attack. This can be achieved by reducing the SNR or by introducing noise that confuses the adversary.

Since this research focus is a software based solution, this section will discus the most prominent existing techniques used as a software based countermeasures against Side Channel Analysis (SCA) attacks. These techniques are masking (Oswald *et al.*, 2005), random precharging (Tillich and Großschädl, 2007), hiding (Tillich *et al.*, 2007) and shuffling (Veyrat-Charvillon *et al.*, 2012).

Signal Masking and Scrambling (Golic and Tymen, 2003, Oswald *et al.*, 2005, Prouff and Rivain, 2013) are techniques to remove the attackable segment of the signal from the adversary. Although masking is effective, these form of schemes are usually very specific, requiring expert knowledge about specific cryptographic algorithms and it involves significant modification to the algorithm. The performance is reduced and the device resources increased (Goubin and Patarin, 1999). Therefore, this research will not focus on masking techniques as the aim is to create a software based countermeasure that does

not modify the existing cryptographic algorithm but to have a countermeasure that can execute alongside the cryptographic algorithm as this would increase flexibility and allow the countermeasure to be executed with various other cryptographic implementations. However, if the reader is interested in masking schemes, good knowledge can be found in (Golic and Tymen, 2003)

The hiding countermeasure aims to reduce the correlation between the intermediate variables and its power consumption from a cryptographic device. Hiding can possibly occur in the amplitude or time domains. The hiding of information in the amplitude domain attempts to diminish the power consumption of various executions of the algorithm to reduce the overall power consumption whereas, the time domain requires the use of other techniques to generate more noise and decreases the SNR.

It is easier to implement the hiding technique in the time domain as a software countermeasure, as opposed to the amplitude domain where this form of hiding is more commonly performed at a hardware level (Popp and Mangard, 2005). Utilizing this approach reduces the correlation between the power usage and its secret key-dependent intermediate variables. Two methods can be utilised to achieve randomness. These two methods are inserting dummy operations and shuffling the operations of an algorithm. It is important that the dummy code can not be recognized from the normal operation, as the adversary would visibly see the difference in the SPA attack and could ignore the dummy code.

Variations in the amplitude domain can be purposely generated by changing the clock rate of the device on each execution of a program. Phase Shifting as described by Yamaguchi (2006) can lead to a lower SNR which increases the difficulty of an SCA attack. Alternately embedded devices often integrate hardware noise generators (Kamoun *et al.*, 2009) to add to the difficulty of performing a successful attack.

Since this research is investigating multi-threading as a software countermeasure. It is possible to generate a software based countermeasure that can affect not only the time domain, but the amplitude domain. Results of this approach are discussed in Chapter 6.

Hiding in the time domain can occur when traces are misaligned intentionally or unintentionally. Unintentional alignment arises due to lack of a satisfactory trigger signal such

as trigger jitter and phase shift. Intentional hiding in the time domain is attributed to hiding through randomization. This can be achieved by inserting dummy operations at critical sections in the algorithm or having random delays or wait for timers.

A second approach is to shuffle the sequence of operations. In terms of the AES cryptographic algorithm operations such as *AddRoundKey()*, *SubBytes()*, *ShiftRows()*, or *MixColumns()* can be shuffled. Figure 3.20 illustrates a randomization of the *SubByte()* round and a code snippet of this process is located in Listing D.9.

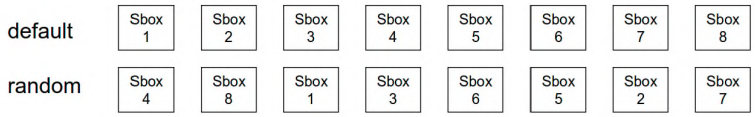


Figure 3.20: Randomization of the *SubByte()* round.

This research will incorporate the implementation of the countermeasures discussed in this section as a comparison between existing and the proposed countermeasure which will be discussed in Section 5.2.4.

3.10 Quantifying a Software Countermeasure

As countermeasures are designed a vital question is put forward. How can the system be evaluated in terms of SCA resilience? as there is not one standard metric of evaluation to follow. However, the most generally and strongest indication of a countermeasures overall strength is the success rate of a SCA attack.

The success rate can be further broken down into categories such as the number of traces required to perform a successful attack, the overall time that is taken to perform a successful attack – this includes data collection and digital processing – and finally, which attack models are successful.

In some cases, the countermeasure is not fully resistant to SCA attacks. However, the resistance of the countermeasure can be so high that to perform a successful attack would

be financially and computationally expensive. For example, a good SCA resistance can be achieved by using both the shuffling and insertion of dummy operations in the AES algorithm (Tillich *et al.*, 2007).

The most effective location in the AES algorithm for this approach is at the S-box round of encryption. As mentioned in Section 2.3.1 the AES-128 round consists of 16 S-boxes thus the probability to locate a specific value at a specific point in time is $p = \frac{1}{16}$. Dummy operations (T) can be inserted at the S-box round, this leads to T operations added to 16 true states. On each execution of the algorithm, the randomizer would integrate the true states with the dummy operations randomly. Therefore, the probability of finding a specific value at a specific point in time becomes.

$$p = \frac{1}{(T+1) \cdot 16} \tag{3.22}$$

The protection against SCA attacks is determined by the value of p. Therefore, the greater the probability the more resistant the algorithm becomes against SCA attacks (Tillich *et al.*, 2007).

There are other researchers that focus on evaluating the entire cryptographic system at an early stage by utilising statical models and simulations to evaluate the source code to find a correlation between them and the power leakage.

This thesis will evaluate the countermeasures based on the success rate of the attack. The success rate S_r will be calculated by taking the number of subkeys recovered N_r and divided it by the number of actual subkeys (16) i.e:

$$S_r = \frac{N_r}{16} \tag{3.23}$$

The above equation will be utilised throughout this thesis when referring to the calculation of the success ratio of an attack. The correlation accuracy discussed in Section 3.5 will be utilised and the overall correlation accuracy will be calculated by averaging the correlation of the 16 subkeys.

A potential means of evaluating the software based countermeasures can be by utilising

a statistical approach, known as the Chi-square (Satorra and Bentler, 2001) test. This approach is utilised to evaluate hardware countermeasures, especially with regards to hardware noise generators (Gornik *et al.*, 2015). The purpose of this test is to evaluate how likely the EM emissions generated as the AES–128 algorithms executed can be detected within the noise.

3.10.1 Electromagnetic Countermeasures

EM analysis countermeasures include circuit redesign as discussed in Section 3.9.1 to reduce the SNR observed by the adversary. One example is to have EM shielding or introduce additional noise. Another option is to set up physically secure zones where entry is restricted, with these zones surrounded by Faraday cages to prevent the adversary from capturing a strong EM signal. Although all of the above mentioned can secure a device, it is, however, extremely expensive with regards to a number of resources and the money spent to secure a device. Therefore, this research will mainly focus on a software based solution.

3.11 Multi–Threading as a Software Countermeasure

This section will discuss techniques and approaches that are commonly utilised to increase the run-time execution of a program as a potential solution for a software based countermeasure. These techniques are multi-threading and compiler optimizations. Therefore, a brief overview of multi-threading is provided in this section and Section 3.12 discusses compilers.

Multi-threading has been widely utilised to speed up run time operations of an executable. However, this research will investigate the use of multi-threading as a possible software based countermeasure. The AVR thread library (Ferreyra, 2008) for multi-threading on microcontrollers will be used and four different multi-threading API's will be used on the Raspberry Pi. These four implementations are Portable Operating System Interface

(POSIX) Threads or more formally referred to as *pThreads*[9]; C++11 multi–threads[10] ; Threading Building Blocks (TBB)[11]; and Open Multi-Processing (OpenMP)[12].

Multi-threading is the process of executing multiple threads or instructions concurrently. These threads are managed independently by a task scheduler (Hennessy and Patterson, 2011). Normally, one process can consist of multiple threads, as a thread could be a component of a process. The multiple threads execute their instructions in sequential order and share hardware resources such as memory, caches and registers (Patterson and Hennessy, 1998). Although the threads share resources, they are able to execute independently. Therefore, the threaded approach provides many developers with a good platform to create concurrent execution.

Two procedures are generally used in multi-threading to increase the performance of the cores' utilisation of resources to decrease the runtime execution of a program (Rau and Fisher, 2001). These procedures are known as thread-level and instruction-level parallelism. They are normally used in combination with each other in hardware architecture that consists of multiple CPUs.

Based on the above procedures, multi-threading can be further broken down into three techniques. These techniques are blocked, interleaved, and simultaneous multi-threading (Hennessy and Patterson, 2011). The basic multi-threading approach is the block multi-threading technique. While one thread is executing, an event would trigger to block the thread from completing, this is known as a stall or a pause. To prevent this pause, the task scheduler would switch to another thread that is ready to execute instructions. Once the required data is received to unblock the previous thread from completing, the task scheduler would switch back to the blocked thread and complete the remaining instructions.

The second technique, interleaved multi-threading is designed to increase the efficiency of the first approach. This is achieved by eliminating all data dependency stalls from the

[9]pThreads – https://computing.llnl.gov/tutorials/pthreads/
[10]C++11 – http://en.cppreference.com/w/cpp/thread
[11]TBB – https://www.threadingbuildingblocks.org/
[12]OpenMP – https://www.openmp.org

execution pipeline. As there is a high probability of one thread being independent of the rest of the threads, there is a minor chance that one thread would need output from an older thread. This technique is also known as pre-emptive multi-tasking (Levinson *et al.*, 2000).

The third technique involves using a specialised processor built for parallelisation. This processor is known as the superscalar. This processor uses the instruction-level parallelism. A superscalar processor is able to execute instructions from multiple threads every CPU cycle by simultaneously dispatching multiple instructions to different execution units on the process (Hennessy and Patterson, 2011).

The main concern with implementing multi-threading on a single core processor is that the instructions are executed concurrently instead of simultaneously. However, it is still possible to achieve a performance gain using the multi-thread approach on a single core.

In terms of user acceptance, many applications use multi-threads to improve and enhance user experience and responsiveness or an application (Akhter and Roberts, 2006). Instead of blocking the User Interface (UI) on time consuming events, the application would create new threads as the user makes new requests to the application. This approach allows developers to improve application performance, responsiveness, execution time, and lower resource consumption.

Developers utilise various libraries to develop their code in a multi-threaded framework. The most common framework in the UNIX environment is the POSIX Threads (pThreads) (Kunikowski *et al.*, 2015). Multi-threading libraries provide developers access to creating applications with multi-threaded support. These libraries provide developers with the option to create a synchronized environment between the threads by using mutexes, condition variables, semaphores, monitors and other synchronization primitives (Tanenbaum *et al.*, 1987).

With regards to the experiments performed on the microcontrollers in this research, this research has selected the AVR Threads Library which was designed by Ferreyra (2008). This framework was designed to work with the Atmel AVR family of microcontrollers

and has multi-threading support. A basic preemptive multitasking and a simple round-robin style task switcher are implemented by the library. Originally, the library had been designed to work with the previous generation of Atmel microcontroller. Therefore, the code has been modified and recompiled to work for the ATmega328p and ATxmega128D4 microcontrollers.

POSIX Threads is a C API multi-thread library that has standardized functions above the Operating System (OS) infrastructure. It allows the user to spawn a new concurrent process flow and it is extremely effective on multi-processor or multi-core systems where the process flow can be scheduled to run on another processor. This interface has been specified by the IEEE POSIX 1003.1c standard (Barney, 2017).

The C++11 thread library is more of a memory model approach that supports multi-threading (Ellis and Stroustrup, 1990). C++11 threads are fully incorporated into C++ as a language, such that there is no need to allocate arguments in a form of a *struct*. C++11 multi-threaded library allows for numerous amounts of arguments to be passed to a function that has a thread, and further performs *typedef* checks automatically.

The Intel TBB (Reinders, 2007) is a portable and open-source C++ template library for parallel data processing on shared memory architectures. It implements task flows in conjunction with the new C++ lambdas. The TBB is a high level implementation, thus it is implemented as a library and not a language extension. TBB allows the developers to create portable code that can execute on different OS architectures.

OpenMP is a directive based extension to C/C++ and Fortran as it supports data and task parallelism on shared memory architectures (Dagum and Menon, 1998). As opposed to TBB, OpenMP is a language extension that requires an OpenMP-enabled compiler.

A common aspect of these multi-threading techniques is that they are all built upon pThreads. However, each threading approach has its own advantages and disadvantages. Depending on the task at hand, the correct multi-threading technique can be selected.

If the developer desires total control, then C++11 or pThreads might be the best choices. If implicit resource management is required then TBB might be the better option. How-

ever, if the developer requires a multi-threading API for cross-platform capabilities with less overhead and fewer integrated possibilities, OpenMP is the better candidate.

In terms of EM emissions and data leakage, it is unknown what effects each multi-threading approach could potentially have. Therefore, this research feels that it is required to investigate the effects each multi-threading approach would potentially have on the EM field. The experiments and results can be found in Section 6.3.

3.12 Compilers as a Potential Countermeasure

This section discusses the basic concept of a compiler and elaborates on inner workings of the GNU Compiler Collection (GCC)[13] and clang[14] compilers. These compilers have been selected as they are the most prominent C compilers.

A compiler translates source code from a high-level programming language to machine code which the CPU interprets. Compilers form a link between high-level languages and the underlying hardware. A compiler verifies code syntax, performs run-time organization, and process the output according to assembler and linker conventions (Fraser and Hanson, 1995).

A compiler consists of three stages (Johnson, 1981). These stages are the front, middle, and back end. The syntax and semantics are verified in the front end. Lexical and syntax analysis is performed to inform the user of any errors or warnings in the source code. The front end generates an Intermediate Representation (IR) of the source code which is processed by the middle end.

Optimizations are carried out in the middle end such as the removal or useless code, repositioning of loops that require fewer resources, and more. This results in an optimized IR which is sent to the back end.

[13]GCC – https://gcc.gnu.org/
[14]clang – https://clang.llvm.org/

The optimized IR is further analysed and additional optimizations are performed in the back end. These may include optimizing the code for a particular target hardware and the out of this stage produces machine code specialized for a particular processor and operating system.

3.12.1 GNU Compiler Collection

The GNU Compiler Collection (GCC) is a compiler system produced by the GNU Project. The standard compiler for most Unix Operating Systems is the GCC compiler (Von Hagen, 2006). Two options are available to compile C/C++ code, gcc for C source code g++ for C++ source code. The gcc and g++ commands invoke the compiler which converts the source code into machine code and produces a complete executable binary.

These compilers utilise the three stage process. A parser is used to produced syntax trees from the source code in the front end. LALR parsers generated with Bison (Levine, 2009) was first used by GCC. However, hand-written recursive-descent parsers (Mössenböck, 1991) are used today to support C++ source code. The trees are converted to the middle end's input representation of language-independent trees which is known as GENERIC or GIMPLE form (Novillo, 2006). GENERIC is more complex as apposed to GIMPLE, which is a simplified GENERIC, where various constructs are lowered to multiple GIMPLE instructions.

Compiler optimizations and static code analysis techniques are applied to the trees. These work on multiple representations, mostly the architecture-independent GIMPLE representation and the architecture dependent RTL representation (Gough and Stallman, 2004). Finally, machine code is produced using architecture-specific pattern matching.

3.12.2 The clang Compiler

The clang compiler was designed to replace the GCC compiler (Lattner, 2008). It works in conjunction with the Low Level Virtual Machine (LLVM) (Lattner and Adve, 2004).

This combination allows replacing the full stack of the GCC compiler. It is based on a library-based design which facilitates in the incorporation of other applications.

Since clang is a library-based architecture it aids the compiler to be integrated with tools that interact with source code, such as an Integrated Development Environment (IDE), Graphical User Interface (GUI). In compassion with GCC, which utilises compile-link-debug cycle, it becomes difficult to integrate with IDEs.

More information is retained during the compiling process as opposed to GCC. The original code is preserved, which make it easier to map errors back into the original source. The error reports are more detailed and specific in such a way that IDEs can index the output of the compiler during compilation as the parse tree supports automated code refactoring, as it directly represents the original source code.

3.12.3 Compiler Optimizations

As mentioned before the compilers have various optimization (GCC, 2013, LLVM, 2013) which can be used to increase the execution time and secure the binary executable. Table 3.3 displays the optimizations for the the GNU and clang compilers.

To enable an optimization the *-O* is used followed by the optimization flag (Stallman, 2000). As seen in the table the flags used can be from flags ranges from *O1* – *O3*, *Os*, *Ofast*, *Og* and *Oz*. Each flag has it's own optimization parameters. An example of invoking the parameters would be "*gcc -O3*" or "*clang -O1*".

Optimizations are mainly used to increase the performance of a program. However, this research aims to investigate the possibility of utilising the optimizations as a security feature in terms of protecting against side channel analysis attacks. This research will focus on the optimizations ranging from *O1* – *O3* and the results of this investigation is located in Section 6.3.

Table 3.3: A comparison between the optimizations of the GNU and clang compilers.

	Compiler	
Flag	**gcc/g++**	**clang**
0	No optimization is performed and the source code is compiled in the most straightforward way possible.	No optimization is performed at this level, the compiler generates the fastest and most debuggable code.
1	This level enables the most common forms of optimization which do not require any speed–space tradeoffs. With this option, the resulting executables should be smaller and faster than with *-O0*.	A minimal optimization level which lays between *O0* and *O2*.
2	This option turns on further optimizations which include instruction scheduling. Only optimizations that do not require any speed-space tradeoffs are used.	This is a moderate level of optimization which enables most optimizations and reduces the code size.
3	This option turns on more expensive optimizations, such as function inlining. The speed of the resulting executable is increased. However, the size is increased as well. Under some circumstances where these optimizations are not favourable, this option might actually make a program slower.	This is built on the previous optimization, except that it enables optimizations that take longer to perform or that may generate larger code.
s	*Os* enables all *O2* optimizations and performs further optimizations reducing source code size.	The same as *O2* with extra optimizations to reduce code size
fast	*Ofast* enables all *O3* optimizations with added optimizations that are Fortran-specific.	—
g	Enables optimizations for debugging	The same as *O1* with debugging optimizations
z	—	*Oz* is based on *Os* but reduces code size further.

3.13 Summary

The chapter commences with an introduction to SCA, including a brief history of SCA and Electromagnetic (EM) attacks against embedded and high frequency devices. This is followed by the basics of quantifying device consumption, a brief overview of microcontrollers and how the architecture is linked to the device consumption. and techniques utilised to capture and measure power and EM consumption of the device was mentioned. The chapter continues by discussing techniques that can be utilised to recover secret information based on the device consumption such as simple, differential and correlation analysis.

The procedure to implement an SCA attack was discussed, this included data collection and preparation, good measurement setup, additional signal processing techniques to enhance the signal and finally, a method to evaluate the success of the SCA attack. SCA

countermeasures ranging from hardware, software, and EM only countermeasures, more specifically the hiding and shuffling countermeasure in the time domain was focused on. Finally, the chapter was concluded by detailing potential software that is not commonly utilised as a software based countermeasure but can potentially serve as a countermeasure.

Based on the literature in this chapter, this research aims to contribute to the body of knowledge by being the first to develop a multi-threading software based countermeasure for microcontrollers and a Raspberry Pi by utilising multi-threads and the secure hash algorithm. The effects of the GNU and clang compilers with different optimization levels have on the cryptographic binary at runtime will be investigated in the hopes of improving the software based countermeasure against SCA attacks.

The information in this chapter will be utlised in Chapter 4 as a design an implementation for an attack methodology to recover sensitive information by reducing the time complexity of a Side Channel Analysis (SCA) attack.

If you think you are too small to make a difference,
try sleeping with a mosquito.

Dalai Lama XIV

4

Digital Filtering and Trace Alignment

THIS chapter discusses the digital filtering, and trace alignment techniques used in this research that assist in the recovery of sensitive information by reducing the time complexity of a Side Channel Analysis (SCA) attack.

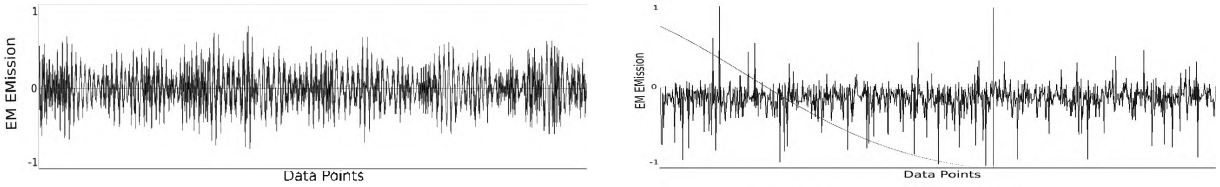
Section 4.1 focuses on the digital filtering techniques such as the Fast Fourier transform, low and high pass filters, and the Savitzky-Golay filter. Section 4.2 elaborates on trace alignment techniques such as the Elastic Alignment, peak detection and the sum of difference techniques. This is followed by a discussion in Section 4.3 as to how this research will incorporate the digital filtering and trace alignment techniques to increase the success ratio of a SCA attack. The chapter concludes with Section 4.4 as the equipment, hardware and software confrontational setup for this research is discussed.

4.1 Digital Filtering

As introduced in Section 3.8 digital filtering can be utilised to perform mathematical operations on a sampled, discrete-time signal to reduce or enhance certain aspects of a signal. This section discusses the principals of transforming the signal and inserting digital information within the carrier signal. Additionally, filtering techniques which allow for certain frequencies to be omitted and the reduction of noise within a signal is discussed.

4.1.1 Modulation and Demodulation

In order for information to be passed over the carrier signal, an additional signal is required and operates at a given frequency. The process of combining the information with the carrier signal is known as modulation. This can be seen as information hidden and the process to retrieve information from a modulated signal is known as demodulation. Figure 4.1 depicts the modulated signal and the effect on the signal after demodulation.



(a) Modulated signal. (b) Demodulated Signal.

Figure 4.1: Modulated (a) and demodulated (b) signals.

There are three types of modulation: amplitude, frequency and phase modulation (Antoniou, 2016). However, this research is focused on phase modulation as it is an integral part of many digital transmission coding schemes. In addition, this research will be monitoring the radio spectrum, thus it is required to apply phase modulation to uncover information.

Phase Modulation (PM) also known as quadrature modulation, is the basis for many digital modulation formats, in which a modulated signal is divided into in-phase (0°) and quadrature (90°) signal components. Digitized information does not require to be continuous. However, it can send a burst of information as required and can be demodulated by the receiver at any time. This research will use quadratic demodulation to recover the information within the signal with regards to raw electromagnetic emissions produced from the Raspberry Pi into useful information. For more details on this process, the reader is referred to Section 6.1.1 as it forms part of the experiments and results of this research.

4.1.2 Fast Fourier Transform

The Fast Fourier Transform (FFT) algorithm (Cooley and Tukey, 1965) computes the Discrete Fourier Transform (DFT) of a sequence. Simply, the signal is converted from

the time domain into the frequency domain. The FFT reduces the time complexity of a DFT from $O(n^2)$ to $O(n \log n)$ time. The FFT algorithm is faster than the DFT implementation as time complexity is generally estimated by counting the number of elementary operations performed by the given algorithm (Welch, 1967).

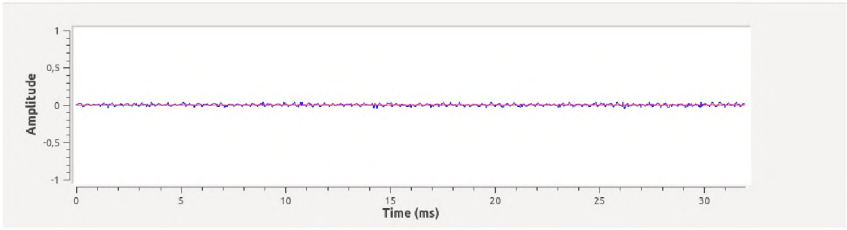
This research demonstrates in Section 6.1.1 how to utilise the FFT algorithm to determine at which frequency the device under test is leaking EM information. The resultant time sequence from the FFT will be used to determine at which point in time an event occurs in the EM field, i.e: the occurrence of the cryptographic algorithm. The FFT consists of two categories, the decimation in time, and frequency. The FFT algorithm of Cooley-Tukey rearranges the input elements in bit-reversed order, then constructs the output in the time frequency.

The DFT of length N can be written as the sum of two DFTs, each with a length of $N/2$. The first DFT is formed by the even numbered points and the second DFT by the odd number of points. The DFT's n^{th} point is denoted by F_n:

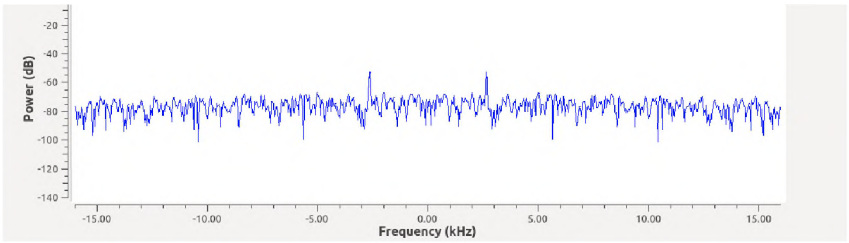
$$\begin{aligned} F_n &= \sum_{k=0}^{N-1} fke^{-2\pi ink/2} \\ &= \sum_{k=0}^{N/2-1} e^{-2\pi ikn/(N/2)} f_{2k} + \boldsymbol{W}^n \sum_{k=0}^{N/2-1} e^{-2\pi ikn/(N/2)} f_{2k+1} \\ &= F_n^e + W^n F_n^o \end{aligned} \tag{4.1}$$

Where, $\boldsymbol{W} = e^{-2\pi i/N}$ and $N = 0, \ldots, n$. Recursion can be utilised to segment the $N/2$ even-odd points to $N/4$ even-odd points, if N is a power of 2. Following this procedure the original transform is converted to $\log n$ transform, as each individual transform point has $F_n^{e\,e\,o\cdots} = f_k$ for some k. Setting $e = 0$, $o = 1$, and reversing the order of even and odd points, the binary value of k is produced. This is the basis for the FFT.

Figure 4.2 illustrates an example of a signal being converted from the time domain to frequency domain. This process is utilised in this research to locate the EM leakage at specific radio frequency. More information can be found in Section 6.5.



(a) Time domain.



(b) Frequency domain.

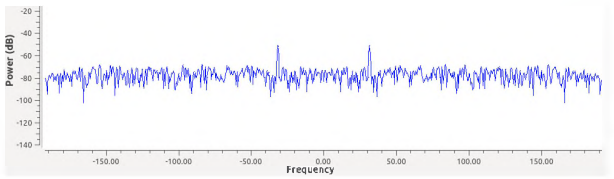
Figure 4.2: Signal converted from the (a) time to the frequency domain (b).

4.1.3 Low and High Pass Filters

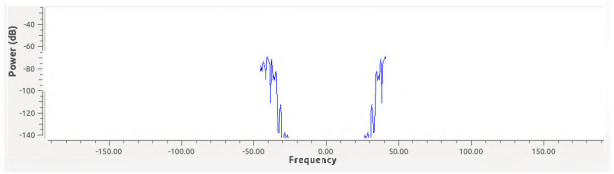
A low-pass filter is a filter that takes the input signal and removes any data above a certain cutoff frequency, with the remainder being the output signal i.e: cutoff frequency is 35kHz, only allow the output signal to consist of a frequency below 35kHz. In addition, a high pass filter is the inverse of a low pass filter as it allows any frequency above the cutoff frequency and impedes anything below it.

In terms of analog circuitry low and high pass filters are constructed by utilising resistors with either capacitors or inductors. If the filter comprises a resistor and a capacitor, it is known as a low/high pass Resistor Capacitor (RC) filter, and if the filter has a resistor and an inductor it is called a low/high pass Resistor Inductor (RL) filter. However, since this research is using a digital filter, the low and high pass filters are designed within the software. The software utilised to create an implement these filters is GNURadio and will be discussed in Section 4.4.4. Figure 4.3 depicts an example of high and low pass filters being applied to the signal at 600MHz, while the rest of the signal around the 600 MHz baseband is discarded.

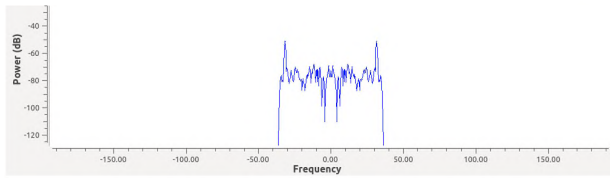
GNURadio allows for the modification of the filters in real time. Therefore, the filters do not require to be redesigned like an analog filter where the circuitry itself has to



(a) Original Signal.



(b) IIigh pass filter applied.



(c) Low pass filter applied.

Figure 4.3: Raw signal (a) passed through the high (b) and low (c) pass filter.

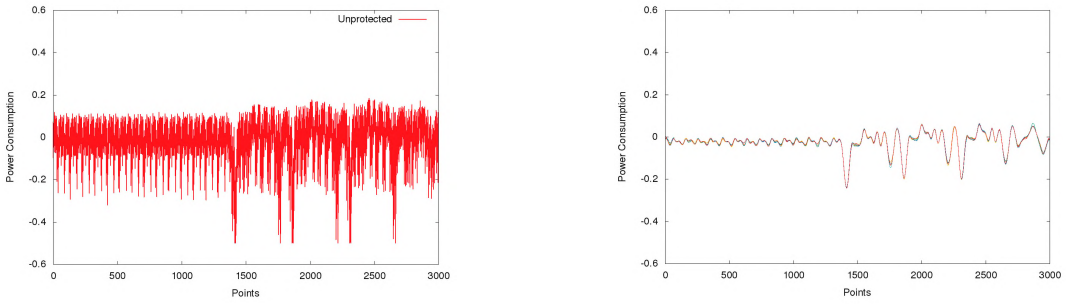
be modified (Rolf, 2009). This allows the research to interactively determine the exact frequency the device is leaking out information, which are demonstrated in Section 6.5.

4.1.4 Savitzky–Golay Filter

The Savitzky and Golay (1964) (SG) is a process which can be applied to a digital signal for the purpose of "smoothing" the signal. The approach aims to increase the signal-to-noise ratio without greatly distorting the signal. The data smoothing technique is based on local least-squares polynomial approximation (Savitzky and Golay, 1964). Research demonstrated that the smoothing reduces noise while preserving the peaks in the waveform (Schafer, 2011).

In order to achieve smoothing the process of convolution is applied (Smith, 1997). This is achieved by taking successive sub-sets of adjacent data points with a low-degree polynomial by utilising linear least squares. An analytical solution to the least-squares equations is determined when the data points are equally spaced. The solution generates a set of convolution coefficients which is applied to all subsets to form the smooth signal. Fig-

ure 4.4 presents the original EM trace of an unprotected implementation and execution of the AES-128 algorithm compared to the trace being transformed by the SG filter.



(a) Original signal.

(b) Post SG filter.

Figure 4.4: EM trace compared to the trace being manipulated by the SG filter.

In the work by Frieslaar and Irwin (2016a) it was demonstrated that while utilising the SG filter on the traces data, the SNR was increased and full key recovery of the AES-128 algorithm was still possible. This research will use the SG filter to increase the SNR, reduce the input data and improve the success ratio of an SCA attack. The experiments and results while utilising the SG filter can be found in Section 5.2.4.

4.2 Trace Alignment

This section discusses the techniques used in this research to align traces. As mentioned in Section 3.8.3 trace alignment is utilised to increase the success rate and reduce the time and computation complexity of the attack and their application to recover secret information faster. The techniques discussed in this section are Sum of Absolute Differences (SAD), peak detection and Elastic Alignment. Experiments and results of these various techniques within this research can be found in Chapters 5 and 6.

4.2.1 Sum of Absolute Differences

The SAD (Sun, 1998) computes a similarity between image blocks. This is determined by taking the absolute difference between each pixel in the input block and the corresponding

pixel in the template block. A simple metric of block similarity is generated by summing up the difference. The process is further elaborated in the following example:

A template image of 3×3 pixels is used, while the search image is 3×5 pixels. Each pixel is represented by a single integer from 0–9 as seen in Figure 4.5.

2	5	5
4	0	7
7	5	9

2	7	5	8	6
1	7	4	2	7
8	4	6	8	5

Figure 4.5: Template image (left) and the search image (right).

Firstly, the search image is overlaid in three sections within the template image as illustrated in Figure 4.6. Followed by the calculation of the SAD values going by column, thus the difference between 2 and 2 is 0, 4 and 1 is 3, 7 and 8 is 1, and so forth. The resultant calculation is depicted in Figure 4.7.

A

2	7	5	8	6
1	7	4	2	7
8	4	6	8	5

B

2	7	5	8	6
1	7	4	2	7
8	4	6	8	5

C

2	7	5	8	6
1	7	4	2	7
8	4	6	8	5

Figure 4.6: Application of SAD.

A

0	2	0
3	7	3
1	1	3

B

5	0	3
3	4	5
3	1	1

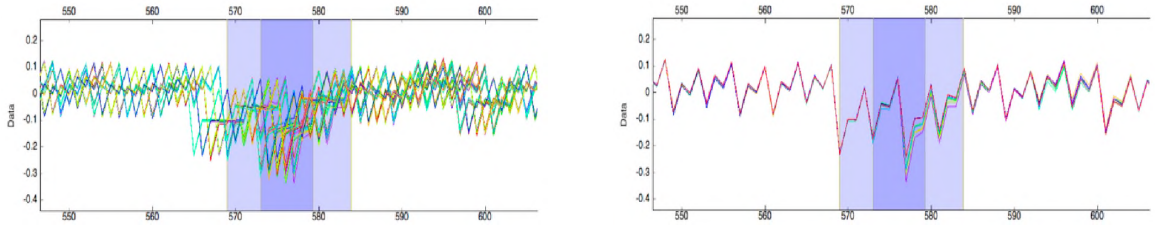
C

3	3	1
0	2	0
1	3	4

Figure 4.7: Resultant SAD calculation.

The absolute differences in each section are tallied, resulting in the values of 20, 25, and 17. The value with the lowest absolute number is the is the most similar to the template image.

The SAD is used to align traces and can be seen in Figure 4.8. The shaded areas within the figure represent the SAD search window. Once the data is aligned, it is used as input for the SCA attack. Chapters 5 and 6 will demonstrate that utilising SAD produces good alignment and assists in the recovery of more secret information.



(a) Misaligned traces. (b) Aligned traces.

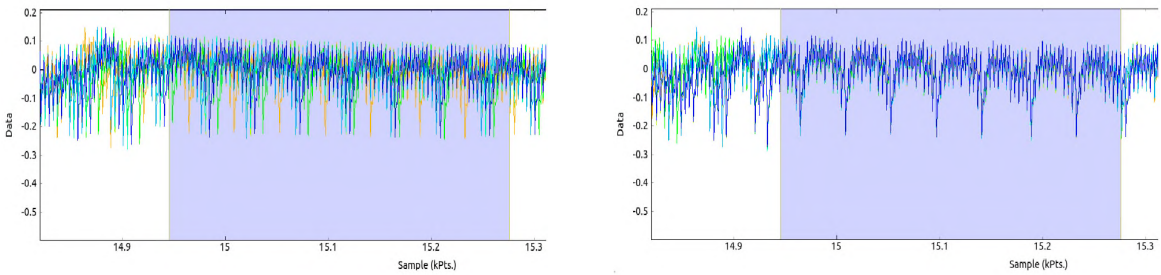
Figure 4.8: Applying SAD on misaligned traces.

4.2.2 Peak Detection

Peaks can consist of a maximum and minimum peak. These are the highest or lowest point in a specific region within the signal (Pan and Tompkins, 1985). In order to align traces based on peak detection by taking a certain region as a reference, the following must hold or the trace is rejected:

$$\frac{(1 - \text{region}) < (\text{peak value from sample trace})}{(\text{peak value from template}) < (1 + \text{region})} \tag{4.2}$$

If the region is 0 then this is ignored, and all traces are kept. Applying this equation the traces can be aligned, the alignment of traces utilising peak detection can be seen in Figure 4.9. The shaded areas within the figure visually represent the area in which peak detection is applied.



(a) Misaligned traces. (b) Aligned traces.

Figure 4.9: Applying Peak detection alignment on misaligned traces.

4.2.3 Elastic Alignment

The Elastic Alignment algorithm (van Woudenberg *et al.*, 2011) aligns captured traces that have been misaligned due to resyncing issues or due to SCA countermeasures. Elastic Alignment is based on the Fast Dynamic Time Warping (FastDTW) algorithm which in turn was based on the Dynamic Time Warping (DTW) algorithm (Sakoe and Chiba, 1978). Originally, DTW was used in speech recognition to align record speech patterns. However, it was replaced with FastDTW (Salvador and Chan, 2007). DTW applies elastic warping to measure the distance between the two utterances over time. The DTW algorithm produces a warping path, which is used to warp the signal even though the utterance had different timing lengths.

In practice, the time complex of DTW is very expensive as approximately $O(T^2)$ time is required for the alignment process. However, Salvador and Chan (2007) proposed an improved DTW algorithm known as FastDTW. The algorithm bounds elements to utilise a stricter warp path. The warp path is determined by utilising various resolutions of the traces which results in the time complexity of $O(T)$.

The warp path is an array of indexes that represents which samples correspond to each other from traces X and Y. Having X and Y traces, the warp path is defined by F

$$F = (c(1), c(2) \dots c(k)) \tag{4.3}$$

Where $c(k) = x(k), y(k)$ indexes in X and Y. An example of the time warp seen in Figure 4.10. However, there are several constraints on the warp path (van Woudenberg *et al.*, 2011):

- Monotonicity: $x(k-1) \leq x(k)$ and $y(k-1) \leq y(k)$.
- Continuity: $x(k) - x(k-1) \leq 1$ and $y(k) - y(k-1) \leq 1$.
- Boundary: $x(1) = y(1) = 1, x(K) = T$ and $y(K) = T$, with T the number of samples in X and Y .

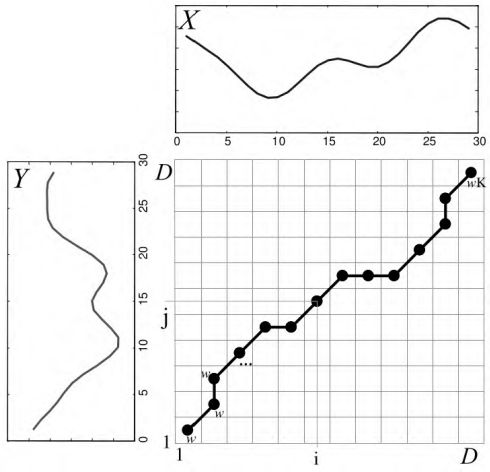


Figure 4.10: Example warp path for traces X and Y. The warp path shows the optimal matching of the two traces by index pairs i and j.

In order to align traces, matching samples need to be projected onto a new set of traces. This implies that Equation 4.4 gives rise to two projections onto aligned traces $\dot{X}$ and $\dot{Y}$. The following asymmetric projections are produced with the assumption that the length of the traces would not increase:

$$\begin{aligned} \dot{X}[i] &= X[i] \\ \dot{Y} &= \frac{1}{|k|x(k) = j|} \sum_{x(k)=j} Y|y(k)| \end{aligned} \tag{4.4}$$

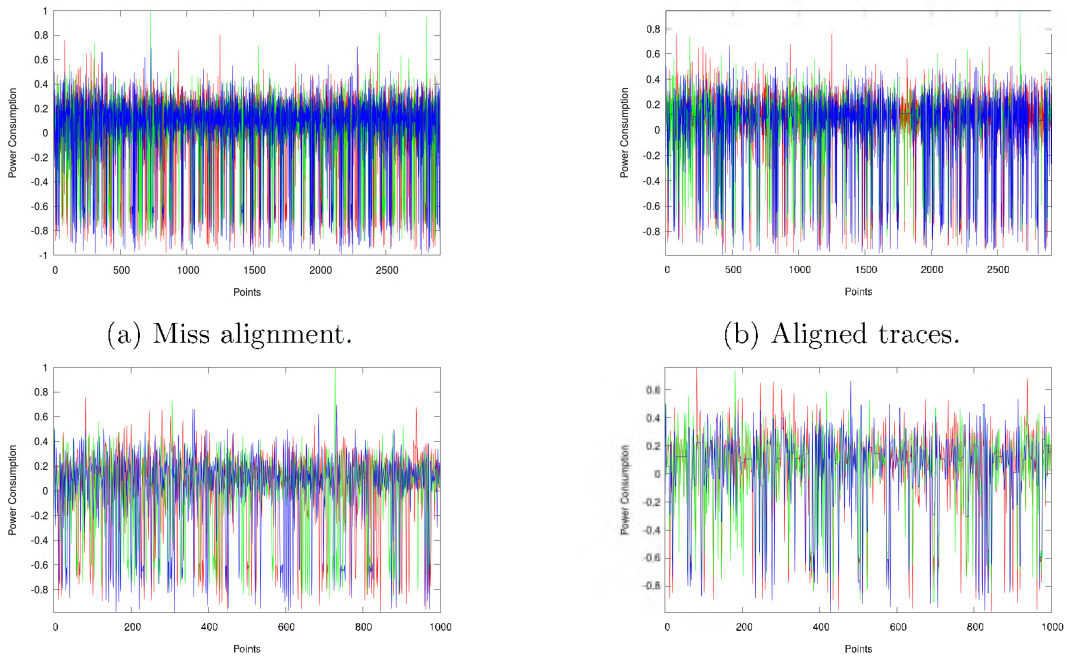
The above equation relates to elastically aligning traces Y to X by duplicating and averaging samples of Y based on the minimal length warping path.

The elastic alignment technique is thus calculated as follows:

1. Capture trace X as a reference trace.
2. Capture a set of traces y; all traces of length T
3. For each $Y \in y$:
   (a) Utilize FastDTW to calculate the warp path.
   (b) Calculate $\dot{Y}|j|$ for $1 \leq j \leq T$.

(c) Output $\dot{Y}$.

Following the above procedure elastic alignment utilises the warp path to resynchronize the traces. Figure 4.11 depicts two misaligned traces, followed by the aligned trace after the Elastic Alignment has been applied. Figure 4.11a displays a few power misaligned power traces and Figure 4.11b demonstrates the traces after the Elastic Alignment technique was applied. Figures 4.11c and 4.11d illustrates a zoomed in view of the traces (from data points 0–1000) for better visualization.



(a) Miss alignment.

(b) Aligned traces.

(c) Miss alignment.

(d) Aligned Traces.

Figure 4.11: Difference between EM traces as Elastic Alignment has been applied.

4.3 Cohesion

The digital filtering and trace alignment techniques introduced in Sections 4.1 and 4.2 will be utilised in this research to reduce the computational time and input data required to predict the correct secret key. Utilising one of these techniques alone will increase the success ratio. However, this research will demonstrate in Chapters 5 and 6 that a

combination of these techniques will assist in the recovery of the secret key from the AES-128 encryption algorithm.

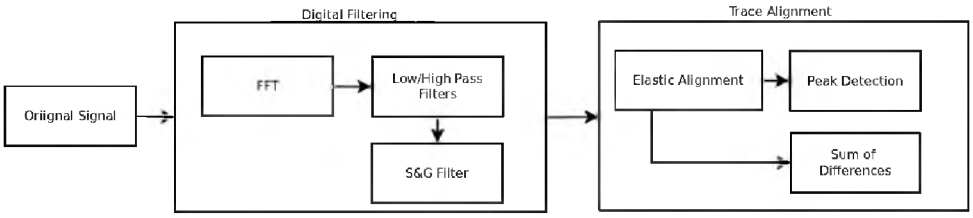


Figure 4.12: Flow diagram linking techniques.

The basic flow diagram of utilising these technique is depicted in Figure 4.12. The process commences by passing the raw signal through the FFT where a low and high pass filter will be applied to remove any unwanted frequencies. This is followed by the SG filter that increases the SNR ratio. The resultant signal is sent to the trace alignment procedure, where firstly, the Elastic Alignment technique is applied. If the trace alignment is still required after the Elastic Alignment technique, the peak detection and sum of difference techniques will be utilised.

Although this is the basic procedure, during empirical testing, this research developed a procedure to follow in order to gain the secret key from the AES-128 algorithm executing on the Raspberry Pi while using EM emissions as input data. The final procedure can be found in Section 6.2.

4.4 Equipment

This section discusses the equipment used in this research. The equipment used to interface and recover information from the microcontrollers were the ChipWhisperer Capture Rev 2 (NewAE Technology, 2017a) and the ChipWhisperer Lite (NewAE Technology, 2017b).

The ChipWhisperer bundle consist of hardware and software which are open source and creates an opportunity for students and researchers to perform side channel analysis at low

cost. The bundle consists of a target device, measurement equipment, capture software, and attack software (O'Flynn and Chen, 2014).

The hardware makes use of a field-programmable gate array (FPGA) and consists of various FPGA code to interact and work with other FPGA's based on the Spartan 6 FPGA (XILINX, 2011). The ChipWhisperer uses a ZTEX FPGA Module with a Spartan 6 LX25 FPGA[1]. The board consists of various features such as mounting points for Analog–to–Digital Converter (ADC) or Digital-to-Analog (DAC) boards, an AVR programmer for interact with microcontrollers, voltage-level translators for the target device, clock inputs, additional power ports of a Low Noise Amplifier (LNA), external Phase Locked Loop (PLL) for clock recovery (Talbot, 2012), and extension connectors for future improvements. This board is known as the ChipWhisperer Capture Rev2 and is depicted in Figure 4.13.

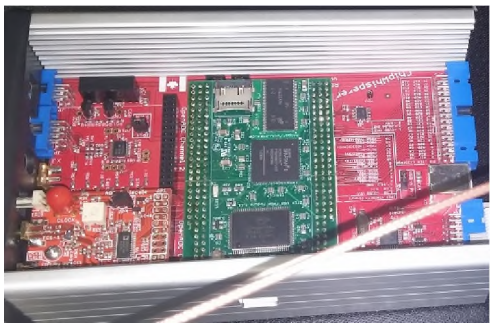


Figure 4.13: ChipWhisperer Capture Rev2.

An LNA is an electronic amplifier which is utilised to amplify possibly weak signals such as those received by an antenna and/or via an EM probe (Bruccoleri *et al.*, 2005).

Figure 4.14 depicts the design of the FPGA with a central register control module. This allows the user to add additional modules to the FPGA. The additional module would be independent of the rest of the system, thus preventing damaging and adding to the flexibility of the ChipWhisperer.

The FPGA captures samples synchronously. It has been demonstrated that capturing synchronously with a low clock of 96 MS/s (Mega-Samples/Second) would yield the same

[1]Spartan 6 FPGA – `https://www.ztex.de/usb-fpga-1/usb-fpga-1.11.e.html`

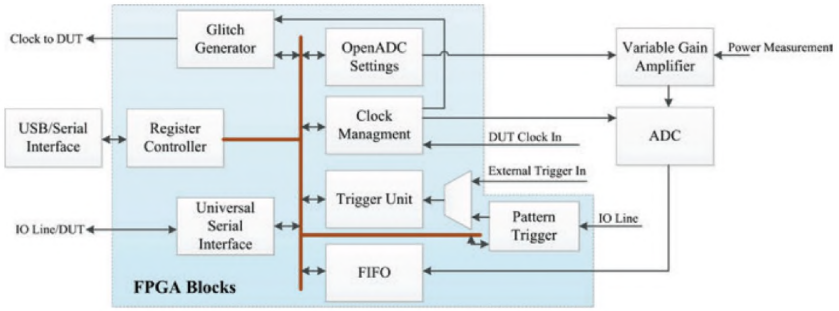
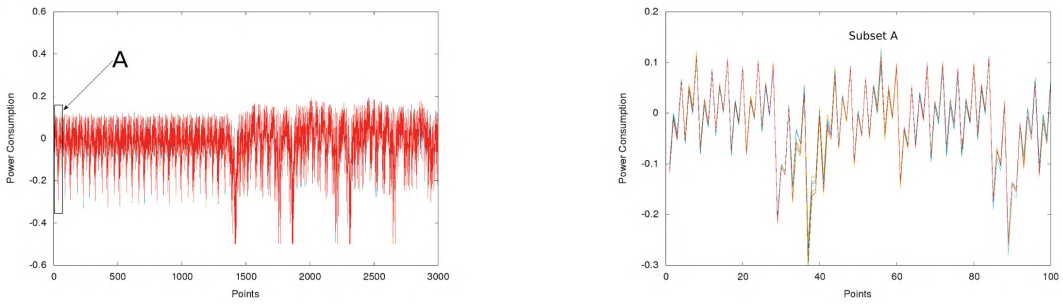


Figure 4.14: Block representation of the FPGA design (O'Flynn and Chen, 2014).

results as captured with a 2 GS/s (Giga-Samples/Second) asynchronously clock (O'Flynn and Chen, 2013). Therefore, the adversary does not require to spend thousands of dollars to acquire an oscilloscope with high sampling rate. The use of synchronization allows the advisory to capture the power signature in a repeatable manner as depicted in Figure 4.15. This reduces the amount of data and time required to perform a successful attack.



(a) 10 traces captured synchronously.

(b) Zoomed in representation.

Figure 4.15: Synchronized sampling.

Figure 4.15a exhibits the power consumption of 10 traces captured synchronously and Figure 4.15b depicts a zoomed-in representation of the 10 traces captured synchronously from 0 – 100 data points for a better graphical representation of synchronous sampling.

In order to apply synchronous sampling, the system clock of the FPGA must be able to lock onto the clock of the device under test (DUT). This can be achieved via the ADC (O'Flynn and Chen, 2013) and if the clock of the DUT is available as a digital signal, it can be sent to the FPGA where the clock frequency can be manipulated for better results. Clock manipulation is discussed in Section 4.4.2.

The FPGA has basic input/output (IO) functionality that allows communicating between other devices such as Universal Asynchronous Receiver/Transmitter (UART), smartcards, and Universal Serial IO device, which can be controlled from a computer. Additionally, the FPGA has several triggering settings. The triggering option used in this research is the rising edge trigger. This is suitable when analysing devices which the researcher has programmed to determine specific events at various trigger points.

A generic target board was provided with the ChipWhisperer and it is known as the multi-target board[2]. The multi-target board provides various configurations to attack different types of hardware such as a 28-pin AVR socket, an XMEGA device, and a Smart Card socket. The board's configuration is changed by jumpers, as various configurations of the jumpers would attack different devices as can be seen in Figure 4.16.

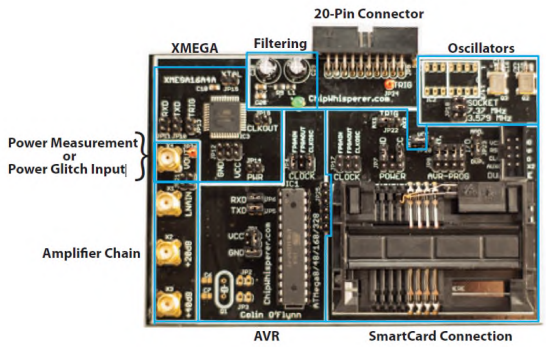
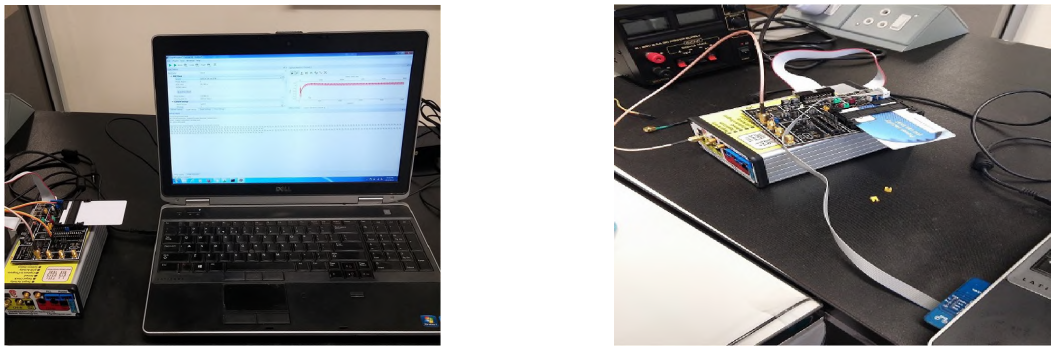


Figure 4.16: Multi-target board (O'Flynn and Chen, 2014).

The multi-target board allows for the interaction with a smartcard in two different configurations. These two approaches will be known as the direct and indirect approach. The direct approach is to insert the smartcard directly into the multi-target board whereas the indirect is to insert an additional feed-through smartcard Printed Circuit Board (PCB) which interns connects directly to the host reader.

Figure 4.17 illustrates the comparison between the direct and indirect approaches to communicate with a smartcard. The direct approach sends commands via the connection between the laptop and FPGA, where as the indirect approach sends commands to the

[2]The multi-target board – https://wiki.newae.com/CW301_Multi-Target



(a) Direct approach. (b) Indirect approach.

Figure 4.17: Direct (a) and (b) indirect acquisition.

smartcard reader, these commands are transferred from the feed-through cable to interact with the smartcard placed in the multi-target board. Experiments were conducted to demonstrate the difference in power leakage between the different inputs and can be found in Section 5.1.2. Code segments to issue commands can be found at Appendix D.10 and D.11 as the different approaches differ in code.

During the experimental stages of this research, an upgraded version of the ChipWhisperer was released. This version is known as the ChipWhisperer Lite and is depicted in Figure 4.18.

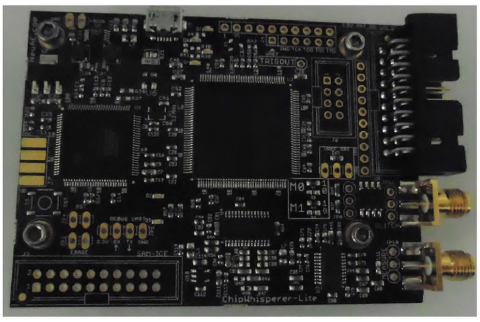


Figure 4.18: ChipWhisperer Lite.

The ChipWhisperer Lite is a small form factor of the ChipWhisperer Capture Rev 2 and is made up of one PCB. It consists of the same Spartan 6 FPGA[3], with high quality $30\mu m$ plated gold connectors, a 20 pin connector to connect external devices and two SMAs (SubMiniature version A) connectors. One connector to attach a coaxial connector

[3]Spartan 6 FPGA – https://www.xilinx.com/products/silicon-devices/fpga/spartan-6.html

which monitors power consumption or attach EM probes, the second connector to attach a coaxial to perform glitch attacks.

4.4.1 Microcontrollers Under Test

This section elaborates on the microcontrollers used as testing targets by this research. The two microcontrollers used for testing were the ATmega328p and the ATxmega128D4. Both microcontrollers have cost-effective in-circuit upgrades making it is easy to provide software updates that will be able to mitigate new and unknown potential attacks. Therefore, these microcontrollers have been selected as this feature allows for a software based defence strategy.

The ATmega328p is part of the megaAVR microcontroller device family from Atmel which offers substantial program and data memories with reduced instruction sets at a minimal power cost. All megaAVR devices offer self-programmability for fast, secure, cost-effective in-circuit upgrades. It has 32 registers which are directly connected to the Arithmetic Logic Unit (ALU), allowing two independent registers to be accessed in a single instruction executed in one clock cycle. For more details on the ATmega328p, the reader is referred to the vendor datasheet (Atmel, 2015).

The ATxmega128D4 which is embedded into the target PCB depicted in Figure 4.19 is part of a microcontroller family that consists of low-power and high-performance, with peripheral rich CMOS 8/16-bit microcontrollers based on the AVR enhanced Reduced Instruction Set Computer (RISC) architecture. The ATxmega128D4 targets low powered applications as it is an entry–level 8–bit microcontroller. More information can be found in the vendor's datasheet (Atmel, 2014).

The Atmel AVR architecture consists of two memory sources, the data and programmable memory. The data memory stores the data whilst the programmable memory stores the executable code as well as the data. The programmable memory which includes flash memory is categorized into three sections. The data memory is made up of SRAM, IO

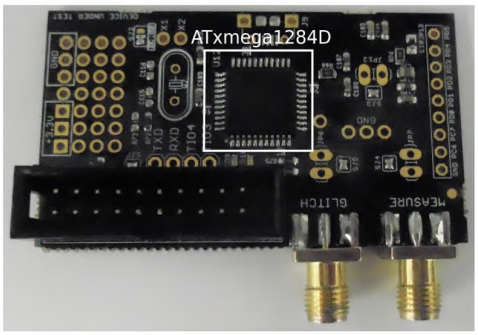


Figure 4.19: ATxmega128D4 embedded into the target PCB.

memory and EEPROM or nonvolatile data storage. Based on this knowledge, this research uses the EEPROM to store random seeds which will be discussed in Section 5.2.1.

4.4.2 ChipWhisperer Software Interface

Python is used as the software to interface with the FPGA. Python has cross-platform capabilities which allow it to easily interact with other languages such as C/C++ and MATLAB. It has a large collection of modules which provide functionality such as cryptographic functions, plotting, numeric computations, low-level IO, and smart card interfaces.

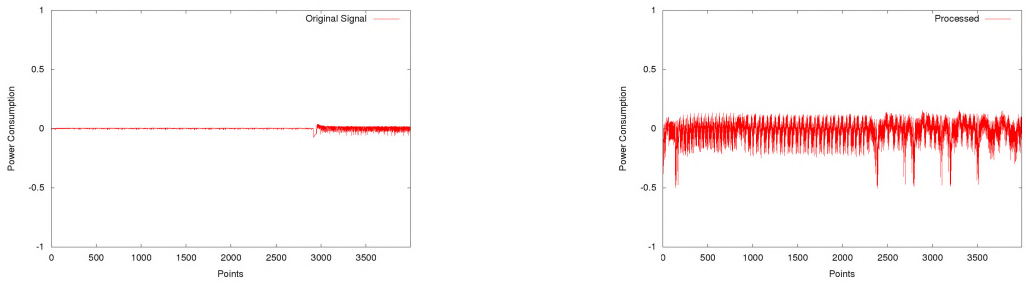
The package consists of two main python programs, the capture[4] and analysis[5] programs. The capture program is used to capture the power or EM traces and the analysis program uses the captured data to perform side channel analysis. Pre-existing data from other sources such as MATLAB can be imported into the analysis program without using the capture program.

The ChipWhisperer capturing software allows for the preprocessing of data. Provision is made for sampling the device at four times its original clock speed, thus improving on the interpretation of captured data. Examples of the signal manipulation are provided below.

Figure 4.20 illustrates a comparison between capturing the power consumption with no processing and with digital post processing. Figure 4.20a is less defined than that of

[4]CWCapture – https://wiki.newae.com/CW-Capture_Tool
[5]CWAnalyse – https://wiki.newae.com/CW-Analyzer_Tool

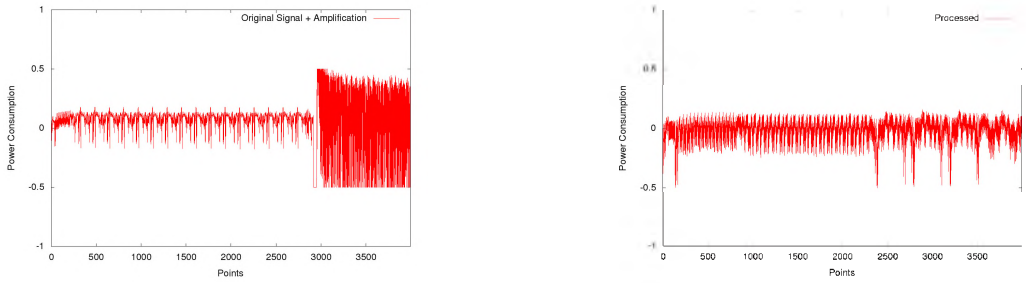
Figure 4.20b, as this power trace, has been amplified and had its clock multiplied by four, by multiplying the clock more information can be sampled.



(a) Original signal. (b) After processing.

Figure 4.20: Example of digital post processing.

Additionally, Figure 4.21 demonstrates a comparison between capturing at 7.38 MHz and 29.5 MHz. Capturing at 29.5 MHz is more beneficial than at 7.38 MHz as more data can be acquired. This is further evident when observing from data points 3000.



(a) Power Trace at 7.38MHz. (b) Power Trace at 29.5MHz.

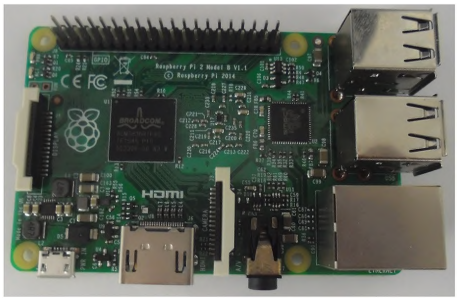
Figure 4.21: Power traces at different frequencies.

The ChipWhisperer suite comes with an analyser program. This program allows the adversary to analyse the captured traces and apply trace alignment techniques such as SAD and peak detection. It has a built in ranking system which determines the likelihood the correct subkey has been obtained. The ranking system has been discussed in Section 3.8.4.

4.4.3 Additional Equipment

The research makes use of the Raspberry Pi 2 Model B – depicted in Figure 4.22a – which is the second generation Raspberry Pi. It includes A 900MHz quad-core ARM

Cortex-A7 CPU with 1 GB RAM. The ARMv7 processor enables the device to install a full range of ARM GNU/Linux distributions. The ARM A7 architecture is found in many smartphones.



(a) The Raspberry Pi.

(b) H-field Probe and SDR dongle

Figure 4.22: Raspberry Pi 2 Model B (a) and the CW505 Planar H-Field Probe (b,Top), with the FUNcube dongle Pro+ (b,Bottom).

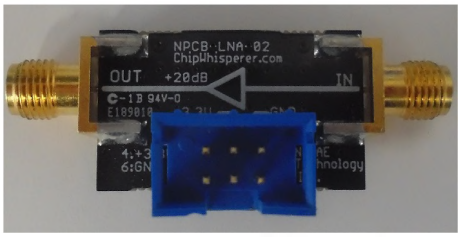
Two Raspberry Pi 2's, the FUNcube Dongle Pro+ Software Defined Radio (SDR) and a CW505 Planar H-Field Probe was used in this research to recover EM data off the Raspberry Pi. Figure 4.22b depicts the FUNcube Dongle Pro+ and the associated H-Field probe. The FUNcube dongle Pro+ dongle receiver contains a software-adjustable mixer and a 192kHz ADC, accessed via USB as a soundcard audio interface to receive radio frequencies. It has a range of presents for the input bandwidth, ranging from 44.1kHz – 384kHz.

The CW304 Notduino PCB Target board[6] depicted in Figure 4.23a was used as a target for the ChipWhisperer. Both versions of the ChipWhisperer can interface with the target board. The board allows for SCA as well as glitch attacks to be performed against the microcontrollers. The research makes use of the CW502 Low Noise Amplifier (LNA) depicted which is displayed in Figure 4.23b to amplifier the signal. The amplifier is based on a BGA2801 device from NXP which has a 20 dB gain up to 2GHz. Details can be found in the BGA 2801 Datasheet (NXP, 2015).

The hardware in this section will be utilised to capture EM emissions from a Raspberry Pi. The hardware setup, experiments and results of utilising the hardware mentioned here are

[6]CW304 Notdunio https://wiki.newae.com/CW304_Notduino_Target

(a) ATmega328p microcontroller in target board.



(b) CW502 low noise amplifier.

Figure 4.23: CW304 Notduino board (a) and the low noise amplifier (b).

discussed in Chapter 6. capturing traces using an SDR simplifies the collection process but requires additional post processing. Therefore, the techniques mentioned in Sections 4.1 and 4.2 are utilised. Since the SDR can collect data continuously, there is no need to modify the cryptographic device to add a trigger. A near-field probe is placed just above the device and the SDR can immediately begin recording the emissions from encryption operations being performed.

4.4.4 Additional Software

As mentioned in the previous section this research will utilise an SDR to capture EM emissions from a Raspberry. In order to interface with the SDR and perform digital signal processing, GNURadio 3.7.9[7] and Baudline 1.0.8[8] was utilised.

GNU Radio is a free software development toolkit that provides signal processing blocks to implement software defined radios and signal processing systems. Baudline is a time-frequency browser designed for scientific visualization of the spectral domain. Baudline can be utilised to the extract signals at specific points in time.

The utilisation of these software packages will be discussed in Section 6.1.1.

[7]GNURadio https://www.gnuradio.org/
[8]Baudline – http://www.baudline.com/

4.5 Summary

This chapter discusses the digital filtering, and trace alignment techniques that assist in the recovery of sensitive information by reducing the time complexity of a Side Channel Analysis (SCA) attack. Techniques to increase the signal-to-noise ration such as the Savitzky–Golay and trace alignment techniques such as the Elastic Alignment techniques were discussed. A detailed procedure on how these techniques would be integrated to increase the success ratio of a SCA attack against both microcontrollers and the Raspberry Pi was presented.

The chapter was concluded by detailing the hardware and software confrontation setup utilised to captured power and electromagnetic (EM) consumption from both microcontrollers and a Raspberry Pi.

The next chapter, Chapter 5 will implement the attack and key recovery procedure, discussed in this chapter. The devices under test will consist of smartcards and microcontrollers.

Impossible is a word to be found only in the dictionary of fools.

Napolèon Bonaparte

5

Smartcards and Microcontrollers

THIS chapter discusses the experiments performed and results obtained while utilising smartcards and microcontrollers. This chapter demonstrates that this research is capable of recovering secret information, more specifically the AES-128 encryption key from various hardware such as smartcards and microcontrollers. Although the focus of this research is to develop a countermeasure it is essential to be able to recover the secret information in order to evaluate a countermeasure.

Section 5.1 discusses the experiments and results against a smartcard, starting with a basic power analysis and progressing towards a more advanced statistical analysis attack to recover the AES-128 encryption key. Section 5.2 discuss the experiments carried out against the ATmega microcontrollers as power and Electromagnetic (EM) measurements are utilised to recover secret information. The chapter concludes with Section 5.3 as additional experiments are conducted against other algorithms such as the AES-256 and the DES cryptographic algorithm.

5.1 Smartcards

This section elaborates on the experiments and results obtained while utilising a smartcard. The smartcard had been selected as it is used in various applications such as making

payments and storing personal information. The smartcard is related to this research since the microcontroller embedded in a smartcard utilises the 8-bit microcontroller architecture as discussed in Section 3.2.2. The initial work in this section has been previously published in Frieslaar and Irwin (2015)

5.1.1 Simple Power Analysis

This section discusses the simple power analysis performed against a smartcard which utilised the Java Card framework (Chen, 2000). For all experiments, the same smartcard was used and each program was compiled with the Java Card 2.2.1 framework[1]. This framework was targeted as it is widely used in SIM and ATM cards.

In order to recover information from the smartcard, the direct approach as discussed in Section 4.4 was followed. The smartcard was inserted into the multi–target board, as seen in Figure 4.17a from Section 4.4. This board was connected to the FPGA which intern was connected to a laptop. Furthermore, the programming language Python was used to interface between the laptop, FPGA and target board.

The capture configuration in the ChipWhisperer capture program was as follows:

- The digital signal was amplified by 28.0586 decibels (dB) with a low gain setting;
- The falling edge trigger logic was used to prompt the capturing of the power consumption;
- Followed by the clock frequency which is multiplied by four, this multiple the base frequency 3.375 MHz to 14.3 MHz. Additionally, all samples were captured synchronously as discussed in Section 4.4.2.

The first set of experiments was to determine if subtle changes in the power consumption of a smartcard occur as the input changes of the same application. The test program

[1]Java Card 2.2.1 framework – `https://goo.gl/aKUJq7`

consisted of the English alphabet, based on the input issued via Python the test program would output various characters of the English alphabet with different character lengths. While the program executed on the smartcard, the power consumption was captured.

The first procedure was to determine a baseline. The baseline power consumption was obtained by outputting 0's and it's power signature recorded. Once the baseline was established, the character A of various output lengths was captured. These lengths were 18, 49, 79 and 89. The resultant graph of this first experiment is seen in Figure 5.1.

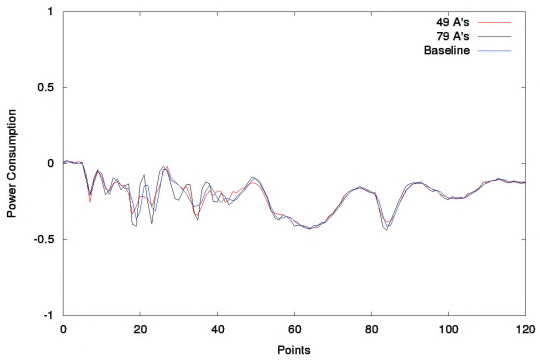


Figure 5.1: Power consumption as the output of A changed.

The figure illustrates three power traces. These three traces are the data from 49 A's, 79 A's and the baseline power trace. It is observed that the output of 49 A's and the baseline were very similar. However, in the area before the 40 point mark, the 49 A's had a slight increase in power. Comparing 79 A's to the baseline and 49 A's, the power consumption increased. Furthermore, in the area close to the 30 point mark there was an additional power spike that the 49 A's and the baseline did not have.

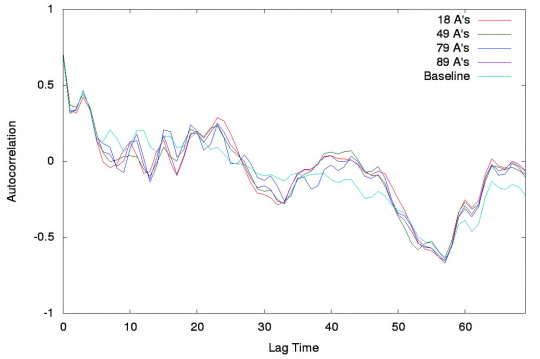


Figure 5.2: Autocorrelation between the various output lengths of As.

The autocorrelation between the various lengths of A's was calculated and is depicted in Figure 5.2. Autocorrelation was utilised to determine if there is a similarity between observations as a function of the time lag between them. Furthermore, it is used to uncover repeating patterns, such as the presence of a periodic signal obscured by noise (Priestley, 1983).

Observing Figure 5.2 the 18, 49 and 89 A's has similar patterns. Although these three instances are akin, it was not identical to the baseline and A's. Monitoring the lag period from 25 40, the power signature of 79 A's differed from the rest. Furthermore, this relationship is evident in Figure 5.1 around the 30 point mark.

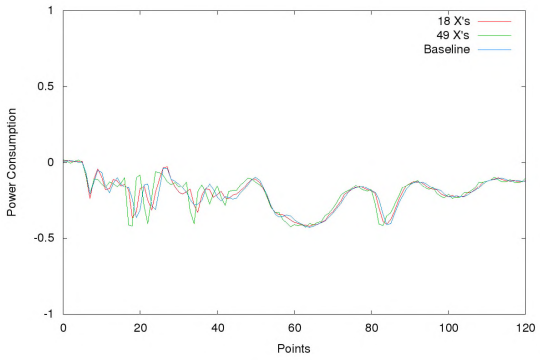
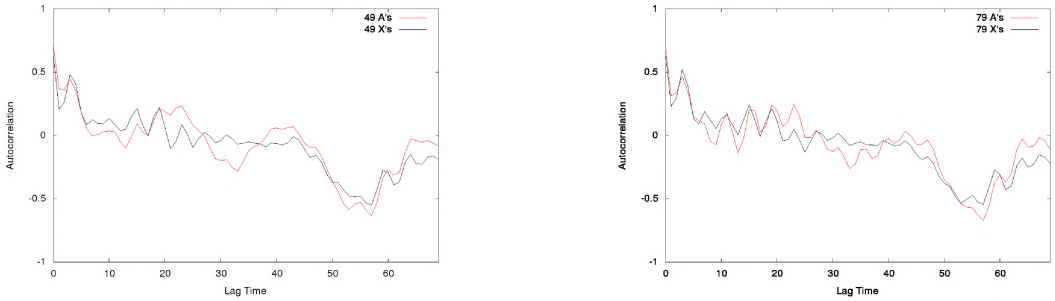


Figure 5.3: Comparison of the power consumption as 18 and 49 X's were used.

The second set of experiments consisted of executing the character X as the output with the length varying from 18, 49, 79 to 89. Figure 5.3 depicts the power consumption as 49 X's and 79 X's were executed on the smartcard with a baseline reference trace. The baseline and 18 X's were similar up until the area close to 40, where 18 X's power usage was slightly more. Comparing the data in Figure 5.3, the output of 49 X's was using more power.

Figures 5.4a and 5.4b illustrates a comparison between the autocorrelation as the input was 49 and 79 Xs and Xs, respectively. The figures depict a clear difference in the power consumption as different characters and lengths are outputted. Thus, various output characters from a smartcard generate a different power signature. Analyzing the code, as the length increases, the output array size increases as well. Having a larger array more bus lines are required to switch from a precharge state to a high state.



(a) 49 A's and X's. (b) 79 A's and X's.

Figure 5.4: Comparison between the autocorrelation of various inputs.

This section demonstrated the power leakage from a smartcard. It has been demonstrated that a different power signature was generated as the input data was changed. The results indicated that more bus lines are required to switch from a precharge state to a high state.

5.1.2 Advance Analysis

This section will discuss the experiments and results obtained as the power consumption of the smartcard were monitored using the different ChipWhisperer smartcard input/output (IO) techniques discussed in Section 4.4. The code that was executed on the smartcard was compiled with two different compilers, the Java Card 2.2.2 and Java Card 3.0.4[2] framework. At the time of these initial experiments, Java Card 3.0.4 was the latest version, currently, today version 3.0.5 is available.

(a) JavaCard 2.2.2. (b) JavaCard 3.0.4.

Figure 5.5: Power consumption using the indirect approach with different compilers.

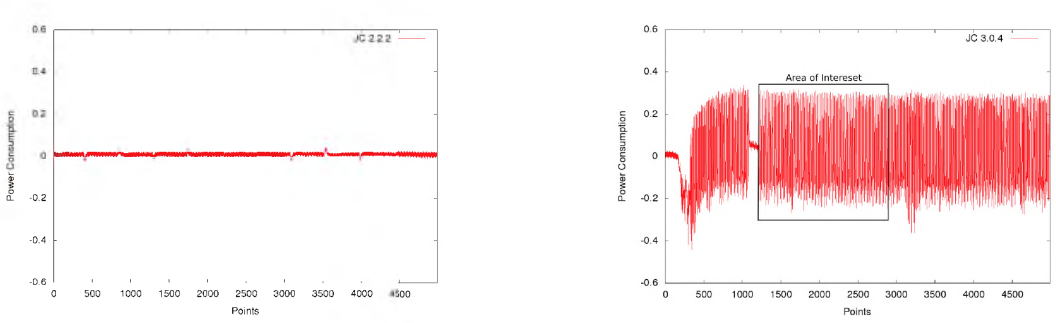
The first set of experiments involved executing an unprotected AES-128 implementation on a smartcard while utilising the two different Java Card (JC) frameworks. The power

[2]Java Card 3.0.4 framework – https://goo.gl/Nh12Pb

consumption was captured via the indirect approach. Figure 5.5 illustrates a comparison between the power consumption using the indirect approach with JC 2.2.2 and 3.0.4 compilers.

The power consumption of the JC 2.2.2 in Figure 5.5a was completely different from the power consumption from the JC 3.0.4 in Figure 5.5b. More power was leaked in JC 3.0.4 as opposed to the JC 2.0.4. However, the increased leakage assisted in the masking of operations. The power consumption of JC 2.2.2 displayed variations in the power, specifically between 2000 – 2500 data points. Although the JC 3.0.4 masked the AES-128 implementation, it will be demonstrated in Section 5.1.3 that key recovery was still possible.

Figure 5.6 depicts the comparison between the power consumption using the direct approach with JC 2.2.2 and 3.0.4 compilers. There was a significant difference between the power consumption between the two implementations. The power consumption of the JC 3.0.4 in Figure 5.6b leaks out more information as opposed to the JC 2.2.2 power consumption. However, the lack of power consumption was not due to the smartcard, it is because the ChipWhisperer has a lack of support for the smartcards using the old JC 2.2.2 framework. This was further evident as different configuration was used in the direct approach and resulted in the same power consumption for the JC 2.2.2 in direct mode.



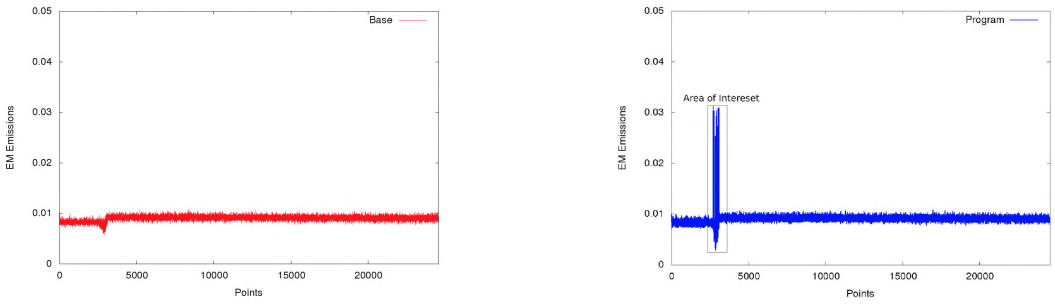
(a) JavaCard 2.2.2. (b) JavaCard 3.0.4.

Figure 5.6: Power consumption using the direct approach with different compilers.

The next stage involved monitoring the EM leakage from a smartcard. The experiment consisted of capturing the EM emissions as the smartcard executed the AES-128 algo-

rithm. The baseline was established by creating a connection between the capture device and smartcard. While there was a connection established the smartcard would remain in in an idle state for five seconds.

Figure 5.7 depicts the comparison between the EM emissions from a smartcard being idle and the cryptographic application that was executed. The base program in Figure 5.7a indicated a small power spike, followed by a constant power source. This was attributed to the smartcard making a connection with the smartcard reader. Moreover, Figure 5.7b illustrates a greater power spike than before, this was related to the actual AES-128 algorithm executing on the smartcard.



(a) Base. (b) Executed program.

Figure 5.7: Comparison between the EM emissions from a smartcard.

5.1.3 Recovering Secret Information from Smartcards

This section utilises the data captured as the AES-128 cryptographic algorithm was executed in the previous section. To increase the degree of difficulty, the JC 3.0.4 data was utilised as the AES-128 process was masked. To achieve key recovery these emissions was used input data for the Correlation Power Analysis (CPA) attack as discussed in Section 3.5.

The indirect setup approach was used as it has been mentioned in Section 4.4 that the adversary can insert the smartcard PCB into any commercial smartcard reader and bypass some of the security features.

While the smartcard PCB was inserted into the reader, the AES-128 cryptographic algorithm was executed. 30 power traces were captured, alongside the corresponding input text. The data was used for the CPA attack as mentioned Chapter 3. As discussed in Section 2.3.1 the secret key comprises of 16 subkeys. Once all 16 subkeys are recovered, the entire secret is considered recovered. The evaluation process mentioned in Section 3.8.4 was followed to ensure the correct secret key was recovered.

No additional signal processing or trace alignment techniques was applied and the first attack resulted in no subkeys being recovered. The techniques discussed in Chapter 4 was followed to align and filter the signal. Firstly, the resync approach of the Sum of Absolute Difference (SAD) was applied.

The SAD method was used on three occasions with different input parameters, which resulted in the recovery of three subkeys as seen in Table 5.1[3].

Table 5.1: Three subkeys that were recovered.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	-	Y	-	-	Y	-	-	-	-	Y	-	-	-	-	-	-	3

Since only three subkeys were recovered, the signal to noise ratio was increased by applying the SG filter with an order of 7 and a frame of 61 and the resyncing technique of SAD and peak detection was applied. The results revealed that an additional 10 subkeys were recovered using this approach, as seen in Table 5.2

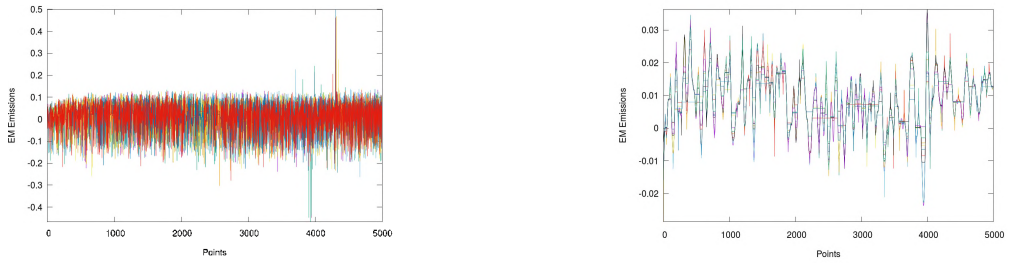
Table 5.2: 13 subkeys that were recovered.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	Y	Y	Y	13

The Elastic Alignment technique discussed in Section 4.2.3 was applied to the data to recover more information from the data. For visual representation the misaligned signal

[3]As this document progresses, the recovered subkeys will be presented in this format. The evaluation of subkeys is discussed in Section 3.8.4

is depicted in Figure 5.8a, followed by the signal after various alignment techniques were applied to the signal in Figure 5.8b.



(a) Misaligned. (b) Aligned.

Figure 5.8: Signal before (a) and (b) after the alignment process.

After applying the Elastic Alignment technique, the aligned data was utilised as input for the CPA attack. The results revealed that the CPA correctly predicted the remaining three subkeys, subkeys 10 –12.

5.1.4 Summary

The results demonstrate that utilising a different compiler and/or smartcard has different power leakages. The ChipWhisperer kit is able to retrieve power and EM information from different smartcards, thus making a good candidate to perform Side Channel Analysis (SCA) attacks. The adversary was able to utilise a fake smartcard, giving him the capabilities to insert the fake smartcard PCB into real systems with the intent of malicious activities by pretending to be the real card.

The EM emissions were captured from the smartcard while it executed the AES-128 cryptographic algorithm. Digital processing techniques such as the sum of absolute difference, peak detection and Elastic Alignment were applied to the captured traces. Once the data was aligned it was utilised as input for the CPA attack. The results revealed the secret key used by the AES-128 encryption algorithm to encrypt data.

Although, this research was able to perform a successful SCA attack against a smartcard. The implementation of a multi-threading software based countermeasure could not be

implemented as the Java Card has no support for multi-threading. It is mentioned that the Java Card 3.x platform has support for multi-threading. However, this is only for server sided applications, thus the smartcard itself cannot execute instructions in a threaded environment (Sun Microsystems, 2008). This research aims to move from smartcards to a pure microcontroller where there is more flexibility in terms of implementing a multi-threading software based countermeasure.

5.2 Microcontrollers

This section will discuss the experiments carried out on the microcontrollers mentioned in Section 4.4.1 in order to recover the AES-128 encryption key. The approach followed to develop a prototype software based countermeasure for microcontrollers is discussed in Section 5.2.1. This is followed by, a demonstration of a Simple Power Analysis (SPA) attack in Section 5.2.2 where a real-world scenario is used as an example. Section 5.2.3 will focus on the data acquired from the power consumption and Section 5.2.4 targets the data acquired from the EM emissions to reveal the secret AES-128 encryption key. The initial work in this section has been previously published by Frieslaar and Irwin (2016a,b,c)

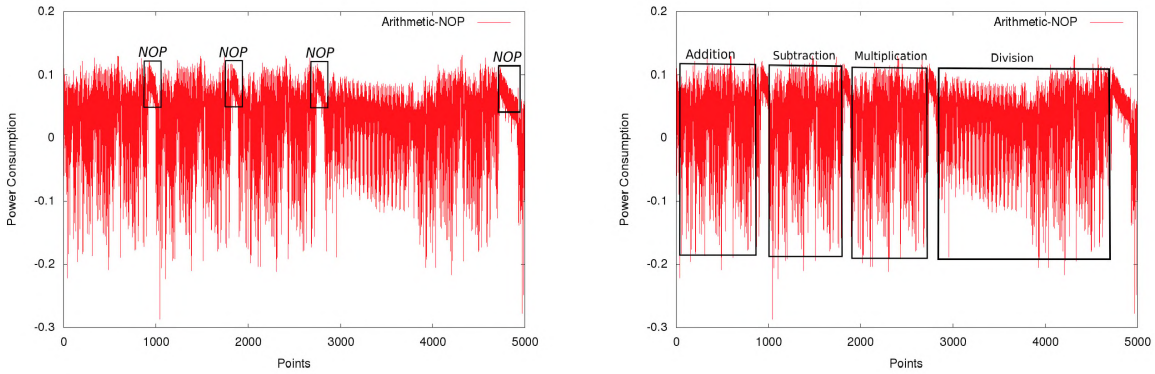
5.2.1 Developing a Prototype Countermeasure

This section discusses the preliminary experiments and results to determine the impact of various noise or mathematical operations has on the power leakage and the establishment of a software countermeasure based on multi-threading. Three vital questions are intended to be answered:

1. What type of mathematical instructions should be used?
2. Where should the additional noise be inserted?

3. What should the runtime of each noise operation be?

The first group of experiments focused on establishing the type of power consumption the various mathematically operation would have on the microcontroller leakage. Power measurements were captured from the ATmega328p microcontroller as the various mathematical operation executed.



(a) *NOP* displayed. (b) Break down of operations.

Figure 5.9: Power traces as the mathematically instructions were executed.

The power traces in Figure 5.9 illustrates the process as the mathematical instructions of addition, subtraction, multiplication and division were executed. After each function the *NOP* command was executed. The *NOP* informs the microcontroller to perform a no operation, this no operation takes 1 clock cycle and assists in analysing power traces as seen in Figure 5.9a by the box encapsulating the *NOP* operation.

It becomes possible to visually locate the addition, subtraction, multiplication and division operations as seen in Figure 5.9b which visually illustrates with each box where the various arithmetic operations occurred. The figure depicts that the addition, subtraction and multiplication execution are very similar. However, the power consumption for the division operation was completely different from the rest.

Before the insertion of various noise into the AES-128 algorithm. A baseline was established by capturing the power consumption from the ATmega328p microcontroller as the AES-128 executed with no countermeasures in place. The CPA attack was utilised and the evaluation metric mentioned in Section 3.8.4 was followed.

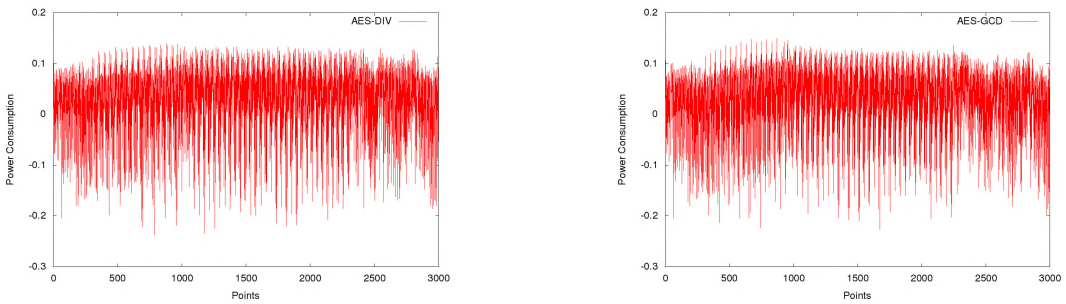
Table 5.3: CPA baseline results.

	Subkey																
Traces	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total(%)**
50	98	98,8	98,7	98,1	96,8	98,3	94,4	98,6	95,9	98,5	98,5	98,8	97,5	98,2	97,2	96,2	97.66

The measurements consisted of 50 power traces, with random plaintext input and the same secret key. Table 5.3 indicates the correlation accuracy for each subkey and the total average correlation for predicting the correct secret key while utilising 50 measurements as input data.

Firstly, it should be mentioned that this was a vanilla implementation of the AES-128 algorithm and all the subkeys were recovered with the CPA attack. It has been demonstrated that only 50 traces were required to achieve the entire secret key at a high overall correlation accuracy of 97.66%. This is extremely useful information as this research will demonstrate that 50 traces were required to recover the AES-128 encryption key from a Raspberry Pi in Section 6.2.

The next set of experiments involved inserting the noise instruction sets after the 15^{th} subkey was calculated in the substitution round. The calculation of the Greatest Common Divisor (GCD) will serve as test case A and the division of numbers will serve as test case B. For now, the focus is on the insertion of noise at various locations within the AES-128 algorithm.



(a) Case A: Division.

(b) Case B: GCD.

Figure 5.10: Power traces after inserting the mathematical instructions.

Figures 5.10a and 5.10b represents the power traces of the AES-128 algorithm at the substitution round with division and the calculation of the GCD after the 15^{th} subkey,

respectively. The leakage can be seen after the 2000 data points in both figures as it is completely different. The results of CPA attack resulted in a 100% success ratio and the entire secret key was recovered, irrespective of the additional power leakage.

The upcoming set of experiments will focus on applying the noise generated by the GCD and division operation at every odd subkey. The CPA attack was applied to the captured power traces of the modified AES-128 algorithm that contained the GCD and division operations. Table 5.4 indicates the correlation accuracy for each subkey and the total average correlation for predicting the correct secret key. The table depicts the accuracies for both test cases of the GCD and division operations, as well as the baseline results in the table.

Table 5.4: CPA attack results as the noise generated are at odd subkeys.

	Subkey																
Method	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Average**
DIV	91,5	88,4	95,2	90,2	93	88	91,7	82	92,7	74,4	91,4	95,7	95,2	89	93,8	90,2	90,15
GCD	89,7	99	94,5	82,2	90,1	98,5	95,2	76,4	92,4	98,4	93,8	92	94,5	97,6	95,4	85,6	92,21
Base	98	98,8	98,7	98,1	96,8	98,3	94,4	98,6	95,9	98,5	98,5	98,8	97,5	98,2	97,2	96,2	97.66

Although there was additional noise produced, all the subkeys were recovered. However, compared to the baseline overall accuracy of 97.66% there is a decrease in the overall correlation accuracy for both the GCD (92.21%) and DIV (90.15%) implementation. It can be further seen that there was a reduction in correlation accuracy per subkey, more notably at the odd number subkeys. This corresponds to the noise generated by the odd numbers of subkeys.

Since the power leakage was remaining constant, and all the subkeys was being recovered it would be unfruitful to apply the same procedure to the even numbers alone. The next step would insert noise at all subkeys. A random selection would select randomly to perform the division or GCD operation at every subkey. Furthermore, on each occasion, the order of noise selected would be different.

The same procedure was followed as mentioned above and 50 traces was captured, followed

by the CPA attack. The results obtained from the CPA attack results depicted in Table 5.5 indicated that only partial recovery was achieved as only one subkey had been recovered.

Table 5.5: One subkey that was recovered.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	1

Table 5.5 illustrates which subkey was recovered, this table format will be utilised across this document to illustrate to the reader which subkey was recovered in the experiments. The results demonstrate that adding changeable noise at the subkey locations effects the CPA from predicting the correct subkey.

It is possible to insert more than one type of noise at a subkey. However, to remain within the limits of the capturing hardware, only one selection is made at each subkey. This limit should not apply to the software developer, thus allowing for the creation of more dynamic threads, generating greater confusion, and increasing the resistance towards the CPA attack.

Now that the research has established that adding various types of noise at the *subbyte* round, the next step was to incorporate it into a multi-threaded environment. The proposed countermeasure utilises the AVR Threads Library as discussed in Section 3.9.

The basic concept is to execute the AES-128 algorithm on one thread, while multiple other threads execute concurrently. These multiple threads would consist of random dummy code, henceforth known as noise threads. These noise threads would be executed at the *subbyte* round as mentioned in Section 3.5 the CPA attack targets this round. Section 3.9.2 previously discussed that inserting noise at the *subbyte* round is a good approach. This would produce different noise threads as the length and type of noise will change on each execution.

The proposed software based countermeasure procedure is depicted in Figure 5.11. Firstly, the algorithm determines the number of threads to be used, by randomly generating a number between $0 - 15$ (x). This number is different from each execution and is used

to generate noise threads at a random subkey $y[x]$. Each $y[\ ]$ value is randomized to ensure the noise threads would occur at different locations per execution. Furthermore, a validation procedure is in place to prevent the noise threads from occurring at the same location from the previous execution. Thus, a dynamic power leakage would be generated per execution as the length and location of the noise threads are different.

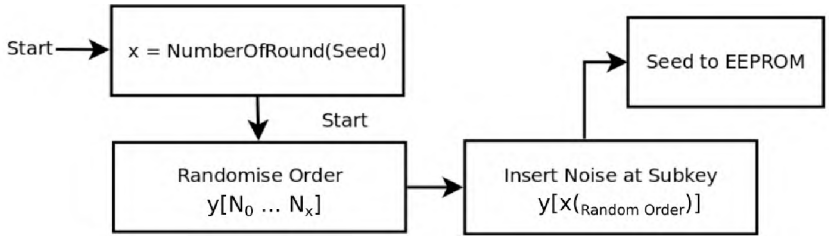


Figure 5.11: Proposed microcontroller countermeasure.

The noise threads have different characteristics and have the ability to execute various mathematical instructions at a different time interval per execution. An example of such a noise thread is the implementation of the greatest common divisor (GCD) or calculating N Fibonacci numbers.

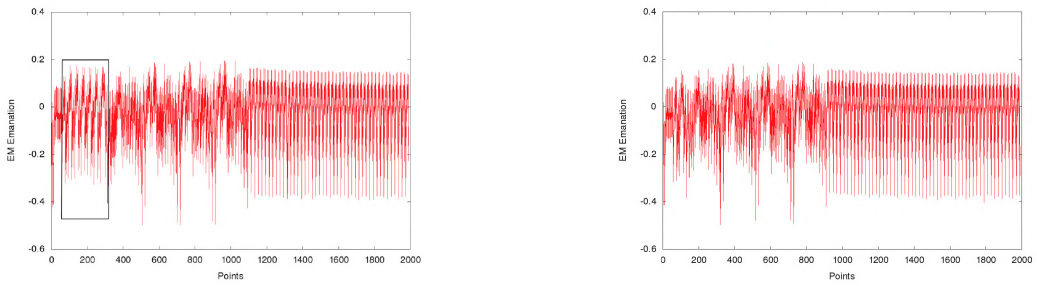
A secondary algorithm is used to insert a value (y) into the electrically erasable programmable read-only memory (EEPROM). The first algorithm would retrieve this value from the EEPROM and use it has a random seed value. Upon completion of the S-box procedure, y would be incremented, XORed and the value would get stored back into memory to be used for the next execution. Thus increasing the confusion since y is XORed and gets passed through the same S-box as the AES-128 algorithm. Additionally, the order of each subkey which is passed through the S-Box is randomized on each occasion.

5.2.2 Simple Analysis

This section discusses the experiments performed on a real world scenario against the *Verifypin* process. It is the algorithm that identifies if the input matches the password of a device Bouder *et al.* (2016). In many smartcards and smartphones, a Personal

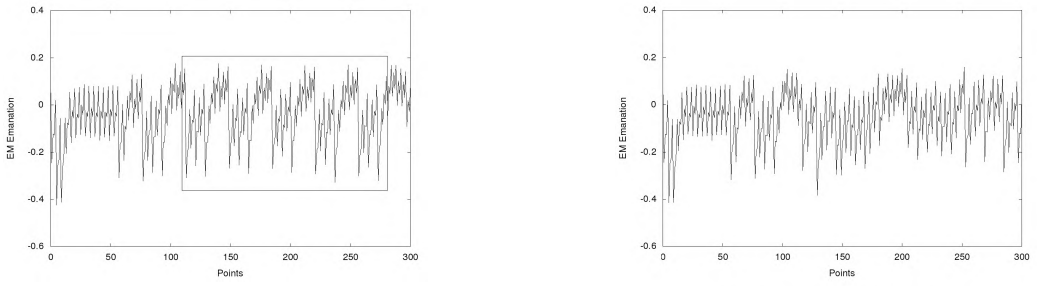
Identification Number (PIN) is used to authenticate the individual. These PIN codes are generally utilised in payment cards, SIM cards and smartphones. Hence, they are targets of choice for malicious adversaries. Recently, Bouder *et al.* (2016) demonstrated the first side channel analysis from electromagnetic emissions on *Verifypin* algorithms. These algorithms have become an interest with regards to SCA resistance. Therefore, this research deems it important to secure *Verifypin* algorithms immediately.

This research developed a pin-acceptance program which instructed the user to enter a five character password, which can be an alphanumeric password. As the input was inserted by the user, the EM emissions of the microcontroller were captured. This emission data was used to perform a simple power analysis attack to recover the password. A layer of security was added to the program by including the countermeasure of random time delays.



(a) Correct password. (b) Incorrect password.

Figure 5.12: EM emissions as the (a) correct and (b) the incorrect password is entered.



(a) Correct password. (b) Incorrect password.

Figure 5.13: Comparison between the EM emanations from points 0 – 300.

Figures 5.12a and 5.12b illustrates the EM emanations from the microcontroller while the correct and incorrect password was inserted, respectively. Observing data points 100 –

300, there was a visible difference in the EM signatures. The adversary could easily locate the occurrence of *Verifypin* and would be able to predict the password. This was further evident in Figure 5.13 which visually represents the EM emanations from points 0 – 300, while the correct and incorrect passwords were entered.

The rectangular block in Figure 5.13a illustrates the change in a byte from 0 to 1 as a correct character was entered. This relates to the process within the algorithm that checks if one character is correct per cycle. The change in bytes appears to shift forward in time by every 36 points and can be seen in Figure 5.14 as the EM emissions changed as more correct characters were entered.

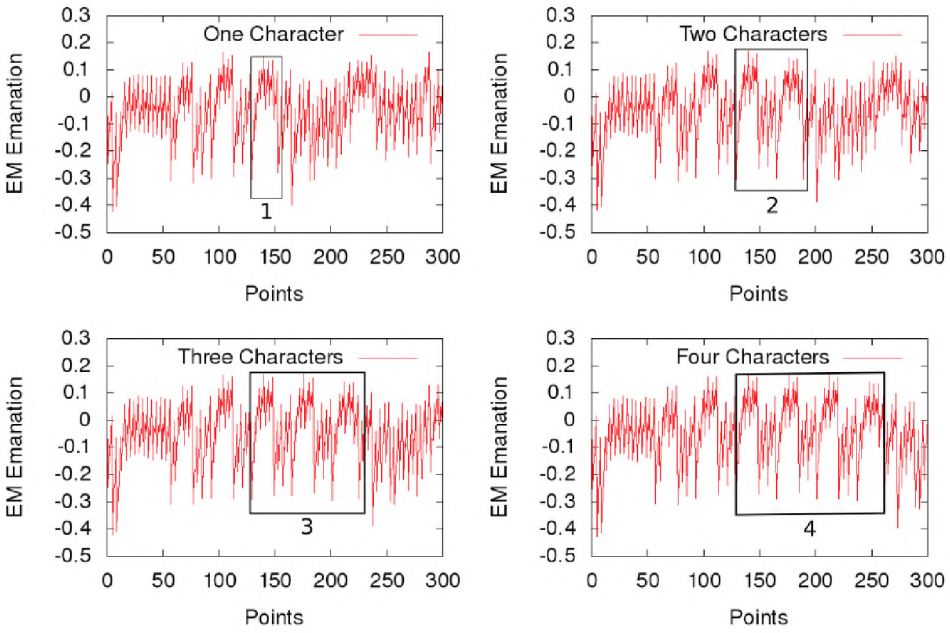


Figure 5.14: Change in EM emanations as correct characters are entered.

A timing attack was carried out as a Python script reiterating through all possibilities per character searches for a change of bytes at a specific point of interest based on the EM data was created. Upon acquiring the first correct character, the subsequent character was attacked by moving forward 36 data points. This process was repeated until all characters of the password were correctly guessed. The timing results of this experiment are depicted in Table 5.6.

These password types are numeric, alphabetical, and alpha-numeric passwords. The at-

Table 5.6: Time taken to predict the correct password.

Password Type	**Time (s)**
Numeric	0:48
Alphabetical	1:12
Alpha-numeric	2:41

tack predicted the correct password under three minutes when the password was alpha-numeric. Only 48 seconds were required to predict a numeric password. Following a successful attack, the known countermeasure of random dummy instructions was inserted into the pin-acceptance program.

Figure 5.15 displays a comparison between the EM emissions as the correct and incorrect passwords are entered with the known countermeasure of dummy instructions in place. Although dummy instructions are executed, the adversary is still capable of locating the *VerifyPin* procedure as seen by the rectangular block in Figure 5.15a. This EM spike is not present as the incorrect password is entered in Figure 5.15b.

(a) Correct password. (b) Incorrect password.

Figure 5.15: EM emissions as the (a) correct and (b) incorrect password was entered.

The next experiment consisted of implementing the proposed countermeasure within the pin-acceptance program. The EM emissions as the correct and incorrect passwords were entered with the proposed countermeasure in place is illustrated in Figures 5.16a and 5.16b respectively.

Based on Figures 5.16a and 5.16b, the EM emissions were very similar in both figures. This relates back to the threaded framework utilising a 128 byte stack. Since the size of the stack was identical to each thread, the advisory was unable to visibly see a difference

(a) Correct password. (b) Incorrect password.

Figure 5.16: EM emissions as the (a) correct and (b) the incorrect password was entered.

when entering the correct and incorrect password. The adversary was unable to determine the correct region of interest, thus he was unable to create an automated timing attack.

The previous results indicated that a multi-threaded framework was able to obfuscate the adversary from locating the points of interesting. However, further mitigation was applied as more threads were randomly inserted at execution time. These threads are known as added noise threads which were discussed in Section 5.2.1.

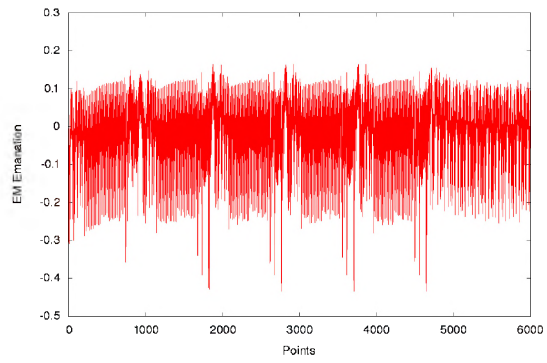


Figure 5.17: EM emanations as dummy threads are inserted randomly.

Figure 5.17 illustrates the EM leakage as the pin-acceptance program is executed with additional noise threads. As the location of *VeriyPin* changes on each occasion, the timing attack becomes near to impossible as the adversary would not know where to look and the EM leakage is identical.

Based on the above results the verifying process is segmented and each character check is given its own individual thread and the order of each thread is randomised. This further assists in the obfuscation of the process where the algorithm confirms a correct character. Additional noise threads can be added to increase the entropy level.

Table 5.7: Average execution time.

Programs	**Time (ms)**
Pin-Acceptance	113
Pin-Acceptance + Multi Thread, 3 Threads	117
Pin-Acceptance + Verification Threads, 8 Threads	125

Table 5.7 illustrates the average time taken in Milliseconds (ms) as one occurrence of the pin-acceptance program, with and without the various countermeasures were executed. The execution time of the encapsulated pin-acceptance multi-threaded program took 117ms, as opposed to the original implementation of 113ms. The proposed countermeasure where the verification of characters was in a threaded design has a 12ms increase from the vulnerable pin-acceptance program. Although the execution time increased, the verification process has become more secure.

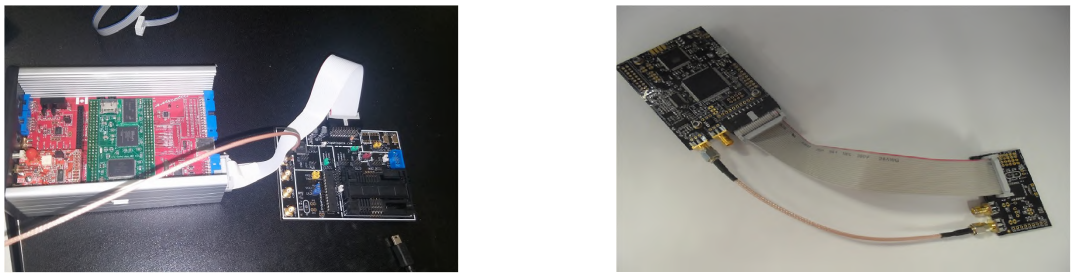
This section demonstrated that the proposed countermeasure was effective against the basic side channel analysis of a timing attack. The countermeasure prevents the adversary from obtaining the password from the *VerifyPin* process. The *VerifyPin* process was modified to include each character check into a thread, this increased the security with a minimal time delay. Having demonstrated the ability to mitigate timing attacks, this research will now proceed to implement and counter against more advanced attacks such as CPA and template attacks.

5.2.3 Power Analysis

This section will elaborate on the experiments and results obtained using the data acquired from the power consumption of both the ATmega328p and ATxmega128D4 microcontrollers.

The aim of the first experiment was to determine a baseline and demonstrate that this research was able to use previous knowledge in the field of SCA attacks to capture leakage from a microcontroller and perform a successful key recovery. Both the ATmega328p and

ATxmega128D4 microcontrollers were used for this experiment and no software countermeasures were in place. The equipment setup for extracting the power consumption for both microcontrollers is depicted in Figure 5.18.



(a) Atmega328p. (b) ATxmega128D4.

Figure 5.18: ChipWhisperer Capturer Rev2 (a) and ChipWhisperer Lite (b) setup.

As both microcontrollers were executing the cryptographic algorithm, the power traces, plaintext and corresponding ciphertext were captured for each execution. The process was repeated 50, on each occasion the same secret key used to encrypt data was used. However, on each execution, the plaintext was generated randomly. Each set consisted of 50 power traces, one trace can be seen in Figure 5.19. Once the data was acquired, the Correlation Power Analysis (CPA) attack was carried out.

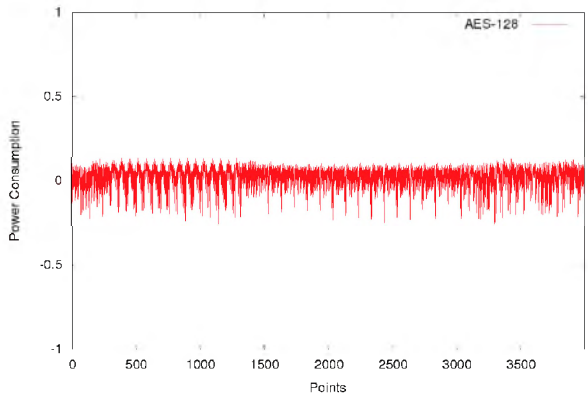


Figure 5.19: Power leakage as the AES-128 algorithm is executed on the ATmega328p.

The consistent variables for experiments in this section are as follows: a low gain setting was used with the digital signal amplified by 34.5039 Decibel (DB)); the rising edge logic level is used to trigger the capture of power traces; and finally, the clock frequency was multiplied to increase from 7.375 Megahertz (MHz) to a frequency of 29.5 MHz and all measurements were captured synchronously. These set variables assisted in increasing the

power output for better readings as discussed in Section 4.4.2.

The data points in Figure 5.19 were used as the input for the CPA attack. The CPA results against both microcontrollers are depicted in Table 5.8, as no countermeasure were used. All subkey values are in percentage.

Table 5.8: Results for the CPA attack against both microcontrollers.

	Subkey																
MCU	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
ATmega328p	98,0	98,8	98,7	98,1	96,8	98,3	94,4	98,6	95,9	98,5	98,5	98,8	97,5	98,2	97,2	96,2	97,6
ATxmega128D4	71,7	86,4	88,7	85	84,7	84,8	75,8	91,0	82,7	71,3	80,2	85,3	75,4	79,4	81,8	87,1	82,0

In both experiments the entire secret key was recovered, thus the Table 5.8 depicts the correlation accuracy of each subkey and the average accuracy of the system. The ATmega328p average accuracy was 97.66% and an average accuracy of 82% was achieved for the ATxmega128D4. The table further demonstrates that a correct key guess was achieved with 71% at subkeys 0 and 9.

The success ratio of the CPA attack against both the ATmega328p and ATxmega128D4 devices executing an unprotected AES-128 algorithm is depicted in Figure 5.20. The success ratio was calculated by utilising Equation 3.10. More importantly, only 50 power traces were required to recover the secret key from both devices.

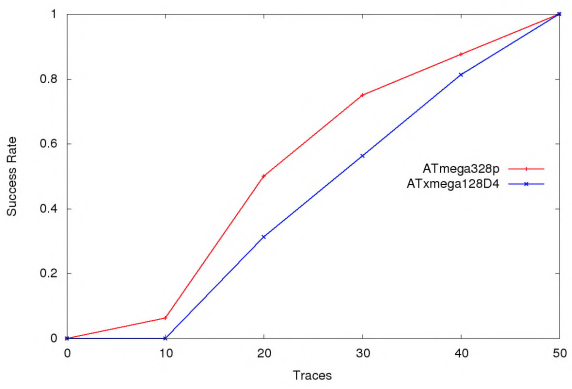


Figure 5.20: Success ratios for both the ATmega328p and ATxmega128D4 devices.

The next set of experiments implemented the proposed countermeasure on the ATmega328p and ATxmega128D4, respectively. Each experiment consisted of an additional

25 test cases and in each test case, the secret keys were changed as the secret key was generated by[4]. This website generated true random numbers based on atmospheric noise. The results of this experiment is illustrated in 5.9.

Table 5.9: Result of the CPA attack over 25 cases.

MCU	Secret Key Predicted	Number of Subkeys Predicted	Subkey Prediction Ratio (%)
ATmega328p	No	2	0.5
ATxmega128D4	No	1	0.25

Table 5.9 consists of five columns each depicting a different result. The first column refers to the microcontrollers used in the test. The second denotes whether the secret key was recovered. Followed, by the next column which indicates the number of subkeys recovered over all cases. Finally, the prediction rate for recovering one subkey is depicted in column four. The subkey prediction rate is calculated by dividing the number of correct subkeys by the total number of subkeys (Total number of subkeys = 400: 25 different secret keys multiplied by 16 subkeys) and multiplying the result by 100.

All the secret keys were incorrectly guessed as seen in Table 5.9, thus the proposed countermeasure was mitigating a CPA attack. Only two subkeys out of 400 subkeys were correctly predicted based on the power traces data from the ATmega328p microcontroller. Therefore, the probability of one correct subkey was 0.5%. The results indicate that the countermeasure prevented the secret keys from being correctly predicted on the ATxmega128D4. Only one subkey out of 400 subkeys was predicted correctly and the probability of one correct subkey was 0.25% on the ATxmega128D4 microcontroller.

Observing Table 5.10, the CPA attack predicted the incorrect secret keys on all six occasions. The results for both microcontrollers have a similar pattern, as the number of sample traces increases the correlation average accuracy to predict one subkey decreases.

The scalability was tested as the number of the trace was increased from 50 – 200. On each increment of 50 power traces, the data would be used as input for the attack procedure

[4]Random.org – https://www.random.org/randomness/

Table 5.10: Result of the CPA with the number of traces increasing.

MCU	Traces	Secret Key Predicted	Number of Subkeys Predicted	Subkey Prediction Ratio (%)
ATmega328p	100	No	0	38.9
	150	No	0	31.8
	200	No	0	27.7
ATxmega128D4	100	No	0	37.4
	150	No	0	30.9
	200	No	0	26.3

for both microcontrollers. Furthermore, the timings of each execution were recorded and are depicted in Table 5.11.

Table 5.11: Execution time of the cryptographic algorithms.

MCU	Time (ms)
ATmega328p	102
ATmega328p + Countermeasure	170
ATxmega128D4	115
ATxmega128D4 + Countermeasure	198

Table 5.11 displays that the ATmega328p micrcontroller executing the proposed countermeasure had an increase of 68 *ms* and the ATxmega128D4 micrcontroller had an increase of 84 *ms*. Finally, both microcontrollers had a minimal execution overhead.

Only the power consumption was utilised in this section and the results indicate that this research was able to replicate existing SCA attacks and perform a successful CPA attack based on power consumption of two different microcontrollers. The strength of a CPA attack was demonstrated as it required only 50 power traces for both microcontrollers to recover the secret key.

The results demonstrate that the proposed countermeasure obfuscated and prevented full key recovery from a CPA attack. The power leakage model of the CPA relies on the power traces to have minimal changes as each subkey is attacked as explained in Section 3.5. The proposed countermeasure produced dynamic power traces on each execution, causing

confusion to the CPA attack model and reducing the correlation accuracy for predicting a correct subkey.

Although the countermeasure was a success, it was a vanilla experimental setup as no existing countermeasures were compared and minimal power traces (200) were used.

5.2.4 Electromagnetic Analysis

It has been demonstrated in Section 5.2.3 that a successful CPA attack was achieved by utilising the data from the power consumption to retrieve the AES-128 encryption secret key. However, the main focus of this research is in the Electromagnetic (EM) field. Therefore, the remainder of this section will focus on experiments and results obtain in the EM field.

Figure 5.21 depicts the hardware setup that captured EM emissions: On the left, the ChipWhisperer Lite connected to the PCB containing the Atmega328p processor; A low noise amplifier is connected to an EM probe placed over the processor.

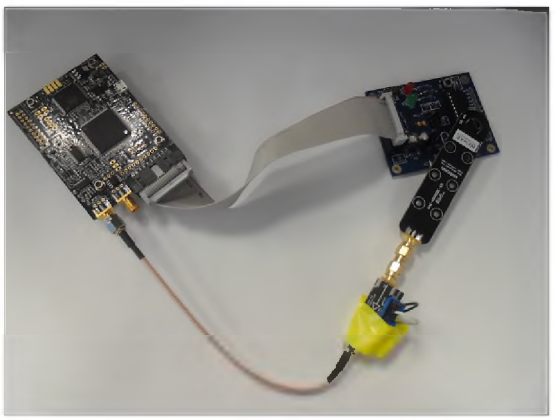


Figure 5.21: Hardware setup to capture EM emissions.

The first set of experiments was to determine the breaking point of an unprotected implementation of the AES-128 algorithm. While the ATxmega128D4 was executing the cryptographic algorithm the EM emanations were captured via the EM probe and ChipWhisperer. The resulting data was used as input for the CPA attack and the results are depicted in Table 5.12.

Table 5.12: CPA results against an unprotected implementation of AES-128.

Traces	Recovered Keys	Success Rate
410	9	0.5625
450	10	0.625
480	12	0.75
1050	14	0.875
1550	16	1

Table 5.12 consists of three columns. The first column indicates the number of traces required to recover a certain number of subkeys, followed by the number of subkeys keys recovered, and finally, the third column represents the success rate. Detail analysis is depicted in Table 5.13 as to which subkey was located.

Table 5.13: Subkeys recovered as the CPA attack was utilised.

	Subkey															
Traces	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
410	Y	Y	Y	Y	Y	-	Y	-	-	Y	-	-	-	-	Y	Y	**9**
450	Y	Y	Y	Y	Y	-	Y	Y	-	Y	-	-	-	-	Y	Y	**10**
480	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	-	-	-	Y	Y	Y	**12**
1050	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	Y	Y	Y	**14**
1550	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	**16**

Based on the Table 5.12 and 5.13, the entire secret key was recovered. It is shown that only 1550 EM traces were required to perform as successful CPA attack while utilising the EM emissions.

The next experiment utilised the template attack as an attempt to recover the secret key against an unprotected implementation of the AES-128 algorithm. The training set for the template consisted of 50 000 EM traces as training samples. The results of the template attack are displayed in Table 5.14.

Table 5.14 depicts the results of a template attack against an unprotected implementation of AES-128 using 50 000 training samples and Table 5.15 illustrates the position each subkey was recovered. Table 5.14 has the same setup as Table 5.12. The results display that as 50 000 samples were used, only 383 EM traces were required to successfully recover the entire secret key.

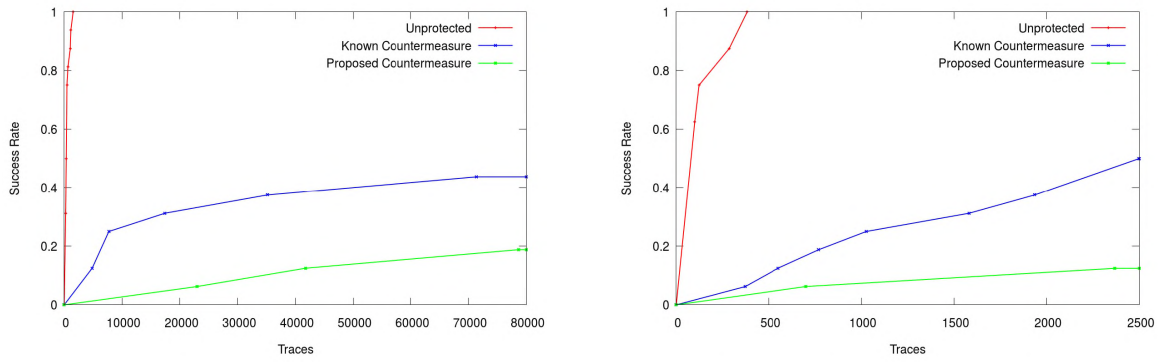
Table 5.14: Results of a template attack against an unprotected implementation.

Traces	Recovered Keys	Success Rate
88	9	0.5625
100	10	0.625
123	12	0.75
286	14	0.875
383	16	1

Table 5.15: The subkeys recovered as the CPA attack was utilised.

	Subkey																
Traces	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
88	Y	Y	-	-	Y	Y	Y	Y	-	-	-	Y	-	-	Y	Y	**9**
100	Y	Y	-	-	Y	y	Y	Y	-	-	-	Y	-	-	Y	Y	**10**
123	Y	Y	-	-	Y	Y	Y	Y	-	Y	Y	Y	-	Y	Y	Y	**12**
286	Y	Y	-	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	**14**
383	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	**16**

The known countermeasures of hiding and shuffling as discussed in Section 3.9.2 as well as the proposed countermeasure, was implemented on the ATxmega128D4 microcontroller. Both the known and proposed countermeasures were attacked with the CPA and template attacks respectively.



(a) CPA attack results.

(b) Template attack results.

Figure 5.22: Success rate of the CPA attack (a) and the template attack (b).

Figure 5.22a displays the results of the CPA attack against the unprotected, known countermeasure and proposed countermeasure. In both scenarios, the entire secret was not recovered as existing and the proposed countermeasure was utilised. However, more subkeys

were recovered while the known countermeasures were used, as opposed to the proposed countermeasure as only three subkeys were recovered, which relates to a success rate of 0.1875 (18.75%).

The success ratio of the template attack against the various implementations of the AES-128 algorithm is depicted in Figure 5.22b. Both countermeasures prevented the recovery of the entire secret key. Although both countermeasures were successful, the proposed countermeasure success rate for template attack was less than that of the hiding and shuffling countermeasure. The proposed countermeasure prevented more information from being used to recover secret information as opposed to the known countermeasures.

The research was able to successfully obtain the secret key on an unprotected device using fewer than 1600 EM traces as input for a CPA attack. Moreover, the research demonstrated a successful recovery of the secret key using a template based attack (Schindler *et al.*, 2005).

The proposed countermeasure demonstrated the ability to prevent the CPA attack from predicting the correct key as opposed to the known countermeasure of hiding and shuffling. The proposed countermeasure was able to mitigate more secret information being recovered from a template attack as opposed to the known countermeasure. It was shown that the template attack achieved a success rate below 0.2.

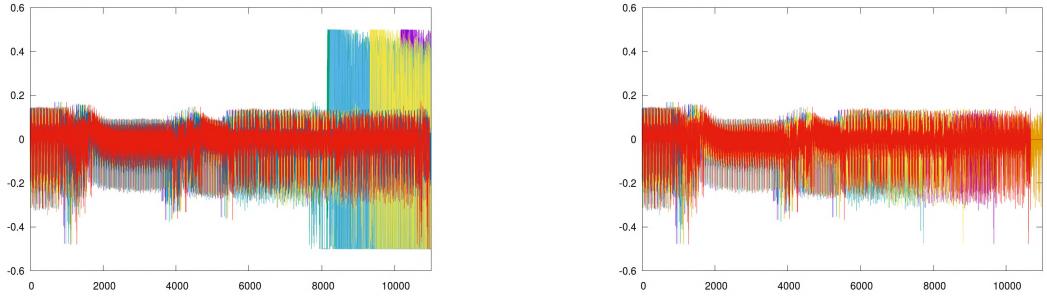
An additional 50 000 EM traces were collected and used with the previous 50 000 to generate a template of a 100 000 traces as input for the template attack, the results are depicted in Table 5.16.

Table 5.16: Results of a template attack.

Traces	**Recovered Subkeys**	**Success Rate**
3	4	0.25
4	6	0.375
8	8	0.5
13	13	0.8125
16	15	0.9375
18	16	1

The secret key was recovered with only 18 testing traces. This is a significant difference when compared to using a template of 50 000 input traces as seen in Table 5.14 which required 383 testing traces. The more samples are used for training the less testing samples are required. It is possible to create a template that requires one test sample. However, at least a million training samples would be needed (Chari *et al.*, 2003).

The proposed countermeasure has a write to memory adding confusion to the leakage. However, after the completion of the substitution round, the write to memory has a high power surge. The post processing phase was reconfigured to remove these memory spikes. Figure 5.23 illustrates the comparison between 10 captured EM traces from the proposed countermeasure with and without the memory signature.



(a) Original signal. (b) No memory spikes.

Figure 5.23: Power consumption (a) with and (b) without the memory power signature.

After 8000 data points, a huge power spike can be seen in Figure 5.23a. This spike in power correlates to the algorithm storing the random seed in memory. Figure 5.23b depicts the EM traces as the memory spikes were removed. Upon acquiring the modified traces, the data was used in the template attack and the results are displayed in Table 5.17.

Table 5.17: Success ratio after writes to memory was removed.

Traces	**Recovered Subkeys**	**Success Rate**
6	1	0.0625
56	2	0.125

Observing Table 5.17, only two subkeys were recovered. This is still less than the subkeys recovered as the known countermeasure was in place as seen in Figure 5.22b. The testing

traces required to recover the two subkeys were reduced as only 56 traces were required as opposed to 2500. Removing the memory spikes increased the prediction rate.

The Elastic Alignment technique was applied to the captured traces as the known countermeasures were enabled. The resulting data were used for a CPA attack and the results are displayed in Table 5.18.

Table 5.18: CPA results against the known countermeasure with elastic alignment.

Traces	**Recovered Subkeys**	**Success Rate**
20	1	0.0625
40	4	0.25
60	6	0.375
100	10	0.625

Table 5.18 indicates a great improvement towards recovering the subkeys as the Elastic Alignment technique was used. Comparing earlier results in Figure 5.22a as nearly 80 000 traces were required the Elastic Alignment reduces the traces required and achieves a success rate of 62.5% i.e recovered 10 subkeys with only 100 traces as opposed to requiring 100 000 miss-aligned traces. These results demonstrated the strength of using the Elastic Alignment technique on miss-aligned traces. This is extremely significant as a 100 traces were captured in a matter of seconds as opposed to capturing 100 000 traces which on average took 10 hours. The subkey details are provided in Table 5.19.

Table 5.19: Ten subkeys recovered.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	Y	Y	Y	-	-	Y	Y	Y	-	-	-	Y	Y	Y	Y	-	10

As mentioned in Section 4.1.4, the Savitzky & Golay (SG) digital filter is capable of increasing the signal to noise ratio, thus the SG filter was applied after the Elastic Alignment process. The resultant data was sent to the CPA attack and the results are depicted in Figure 5.24.

The figure consists of the results from the CPA against the unprotected implementation, the known countermeasure data which has been aligned with elastic alignment, and the

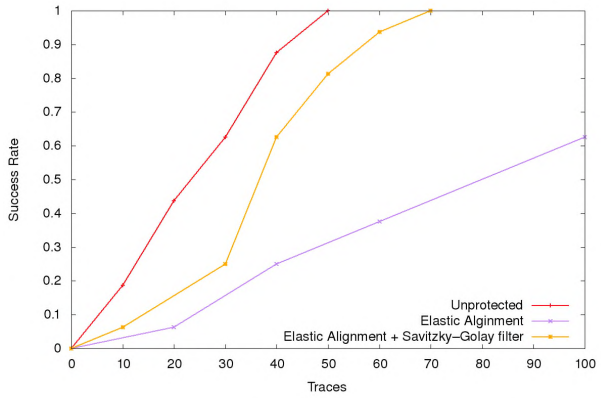


Figure 5.24: The results after the additional signal processing was applied.

elastic alignment that has passed through the SG filter. Figure 5.24 demonstrates that using the elastic alignment achieved a success rate of 0.625 (62.5%). As both the Elastic Alignment and the SG filter was utilised together the entire secret key was revealed. Compared to the unprotected implementation only 10 additional traces were required. Utilising alignment and noise reduction techniques, the adversary was able to recover the entire secret key whilst three different existing countermeasures was used. i.e: hiding; shuffling and random delays. Only 70 traces were required to perform a successful CPA attack.

Table 5.20 depicts the recovered subkeys as different digital processing techniques were applied. It has been demonstrated that using the Elastic Alignment and the SG filter together reduces the required traces to perform a successful CPA attack. Therefore, it is only natural to apply these techniques to the proposed countermeasure data and perform a template attack against it. 100 000 training and 100 test samples were used for the template attack and the results are depicted in Figure 5.25.

Table 5.20: Recovered subkeys as different digital processing techniques were applied.

	Subkeys Recovered																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total
Elastic Alignment	Y	Y	Y	-	-	Y	Y	Y	-	-	-	Y	Y	Y	Y	-	10
Elastic Alignment + S&G Filter	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	16

Figure 5.25 illustrates these results of a template attack against the unprotected, known

countermeasures, and proposed countermeasures implementation. The Elastic Alignment and SG filter were applied to the data. After applying the Elastic Alignment and the SG filter to the known countermeasure data, the entire secret key was recovered (100% success rate) with only 32 testing samples. The trace alignment and denoising techniques were applied to the proposed countermeasure and the results indicated that only two subkeys (12.5% success rate) were revealed, as seen in Table 5.21.

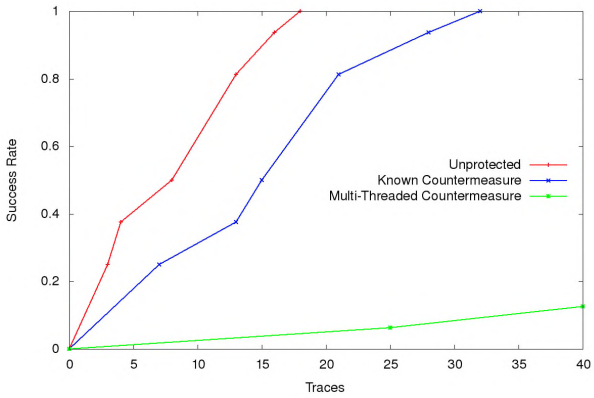


Figure 5.25: Results for template attack after the additional signal processing was applied.

Table 5.21: Recovered subkeys for different countermeasures.

	Subkeys Recovered																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Known	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	16
Proposed	-	Y	-	-	-	-	-	-	-	-	-	-	-	-	-	Y	2

Additionally, the proposed countermeasure mitigates these additional signal processing techniques. This further reinforces that the proposed countermeasure was capable of withstanding various types of attacks CPA and template attack with added trace alignment and denoising techniques.

This section has demonstrated the ability to retrieve the secret key on an unprotected AES-128 implementation as well as a protected implementation with the hiding, shuffling, and random delays countermeasure. In addition, the full key recovery from the proposed countermeasure was unsuccessful.

The Elastic Alignment technique depends on the traces to be similar as it requires a reference trace to reconstruct the signal. Since the proposed countermeasure has dynamic

traces, it is difficult to establish a good reference trace and it becomes difficult to obtain good alignment. The additional digital filtering becomes ineffective as data of the proposed countermeasure varies and problematic to align.

The proposed countermeasure was successful in mitigating information being leaked from the microcontroller. The proposed countermeasure's unique designed allowed the algorithm to resist the CPA and template attack. Furthermore, additional signal enhancement techniques had little effect on the proposed countermeasure.

5.3 Additional Attacks

This section describes additional experiments performed in this research against other cryptographic implementations. These implementations are the AES–256 and the Data Encryption Standard (DES) cryptographic algorithms. The proposed countermeasure will be applied to these algorithms. The DES algorithm has been selected as many companies still have legacy systems in place that utilise a form of DES, such as the electronic payment industry (Mayes, 2017).

5.3.1 AES–256

This sections discusses the experiments and results carried out on the AES–256 cryptographic implementation.

5.3.1.1 Attack

The procedure to capture the power consumption from the attack device is the same as in Section 5.2.3. additionally, only 100 traces were captured. Figure 5.26 illustrates one trace of the power consumption of the AES–256 implementation.

The CPA attack would use the input traces from data points 2900 – 4200 and, the Hamming weight model in this scenario would use the inverse S-box. The results obtained

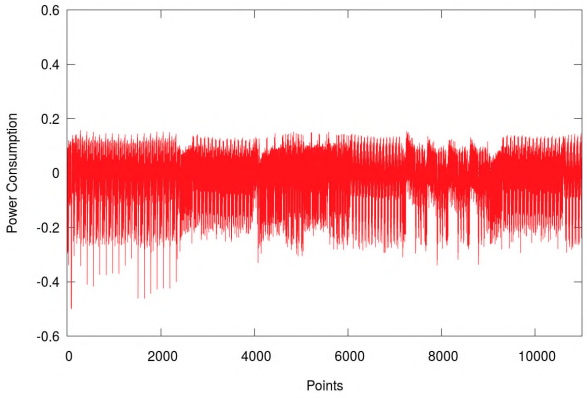


Figure 5.26: Power consumption of the AES-256 implementation.

from the CPA attack for this round recovered 16 bytes as expected. Now that the 14th round key was recovered. The next step is to determine the 13th round's hypothetical key. It was noted that the traces becomes unsynchronized after the 7000 data points as seen in Figure 5.27.

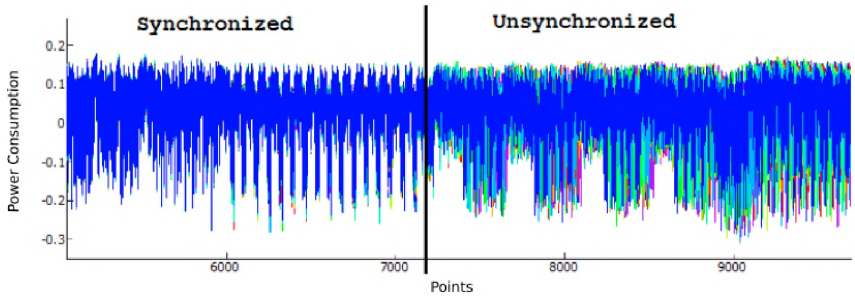


Figure 5.27: 10 traces as the AES-256 implementation was executed.

Before initiating the attack against the 13th round. The traces were aligned using the SAD technique as discussed in Section 4.2.1. The H.W model was modified to include the entire key recovered by the CPA attack on the 14th round as the decryption code of the AES-256 uses an inverted *MixColumns()*. Once the H.W model was configured to cater for the inverse *MixColumns()*, the CPA attack was performed.

The results of this CPA attack provides the hypothetical key for the 13th round. Following the procedure, the next step was to apply *MixColumns* and *ShiftRows* to the hypothetical key to form the actual value of the 13th round key. The resultant value was the real 13th round key and it was used with the 14th round key to recover the full secret key.

Since the key for round 13 and 14 were recovered. The next procedural step was to reverse the key scheduling algorithm to calculate the 0 and 1st round key based off the 13th and 14th round key. The ChipWhisperer software environment consists of a key schedule calculator. The values from rounds 13 and 14 were inserted into the calculator and it automatically displayed the entire key schedule and the initial encryption key.

5.3.1.2 Defence

Now that the research has the ability to successfully recover the secret key from the AES - 256 implementation, the proposed countermeasure of this research was implemented as is at the 13th round attack location. The power consumption as the noise was inserted is illustrated in Figure 5.28.

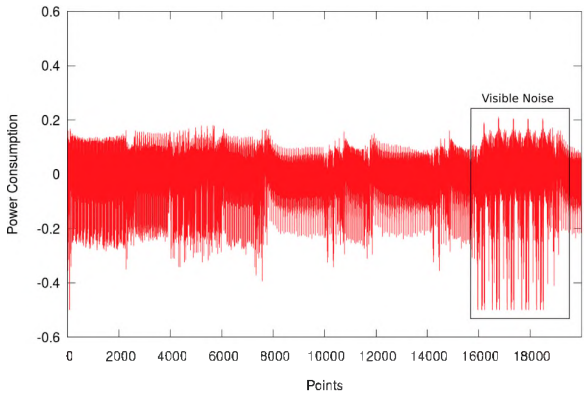


Figure 5.28: Power consumption as visible noise is inserted.

It is depicted that between 16 000 – 18 000 points, large power spikes appeared. The code was analyzed (Appendix D.12 – D.14) and it was determined that the mathematical operations within a *printf* statement had caused these power spikes. These power spike can be clearly seen by the adversary and he would have the ability to remove them. The mathematical instructions was removed (Appendix D.14) from the *printf* statements and the noise generator applied once again applied to the 13th round attack location.

Figure 5.29 displays the traces as the noise generator was applied to the 13th round attack location. Firstly, the noise became more consistent with the leakage produced from the AES-256 procedure. Secondly, the proposed countermeasure prevented any subkeys from

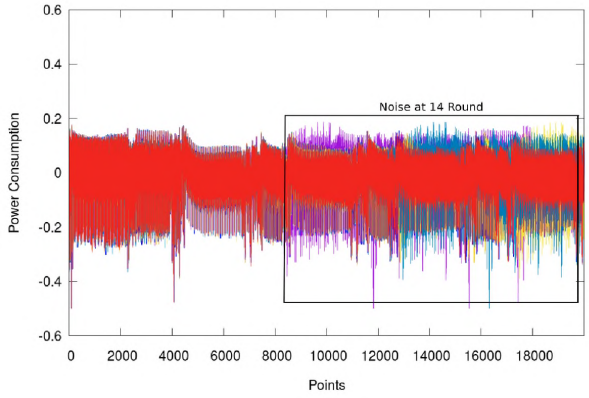


Figure 5.29: 10 traces as the noise generator was applied to the 14$^{\text{th}}$ round attack location.

being recovered at the 13$^{\text{th}}$ round attack location. However, since there was no noise at the 14$^{\text{th}}$ round, the 14$^{\text{th}}$ round's key was still recovered.

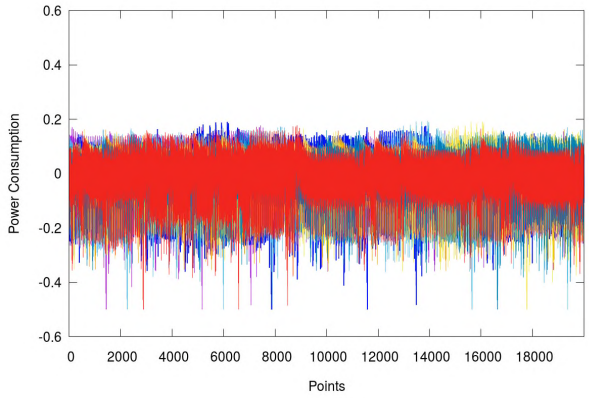


Figure 5.30: 10 traces as the noise generator was applied to the 13$^{\text{th}}$ and 14$^{\text{th}}$ rounds.

For completeness, the proposed countermeasure was applied to both rounds. the noise generator's impact on the traces is illustrated in Figure 5.30. The data was used as input for the CPA attack and the results indicated that the incorrect key was predicted in the 14$^{\text{th}}$ round. Since the 14$^{\text{th}}$ round is incorrect the remainder of the attack failed and the entire secret was not recovered.

This section has demonstrated that this research was able to perform a CPA attack on the AES-256 implementation and successfully recover the secret key. The proposed countermeasure was implemented and it was displayed that the noise threads were visible in the case of performing mathematically instructions within a *printf*. The instructions were modified to perform calculations without the *printf*. The results have indicated that

the proposed countermeasure has prevented an attack on the 13^{th} and 14^{th} round of the AES-256 implementation to prevent the secret key being recovered via an SCA attack.

5.3.2 DES

This section discusses the attack on the DES encryption algorithm. Although DES is a legacy cryptographic scheme, many people still utilise this encryption scheme. Thus, this research will perform the CPA attack on the DES implementation, followed by inserting the proposed countermeasure.

5.3.2.1 Attack

The same procedural setup to capture the power consumption discussed in Section 5.2.3 was used. The DES algorithm was flashed onto the microcontroller and only 100 traces were captured. The power consumption of the DES cryptographic algorithm for one trace is depicted in Figure 5.31.

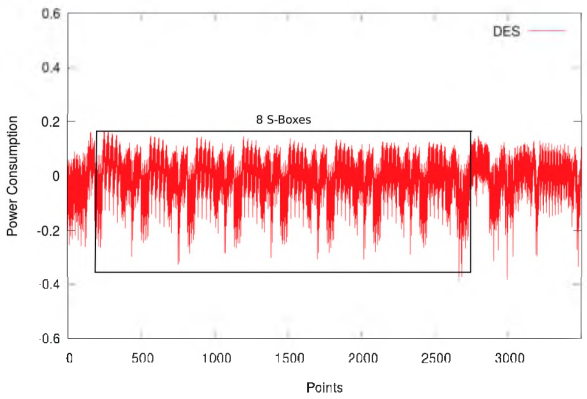


Figure 5.31: Power consumption of the DES cryptographic algorithm.

The attack focuses on attempting to recover the first round key. This key has only 48bits i.e: 8 S-Boxes x 6 bits each. The CPA attack was used and the first round key was recovered. Using this information, the first round key can be mapped using the DES Key Scheduler to obtain the original secret key used for encryption.

5.3.2.2 Defence

The proposed countermeasure was implemented at each calculation of the S-Box. The power consumption is depicted in Figure 5.32 and the noise can be clearly seen. The adversary could remove the noise or use the data untouched for the input to the CPA attack, as the noise is constant and repeatable. Upon further analysis, it was determined that *printf* had a larger spike than the DES procedure. This is the same case as displayed in Figure 5.32. The CPA was performed on this data and the secret key was still recovered.

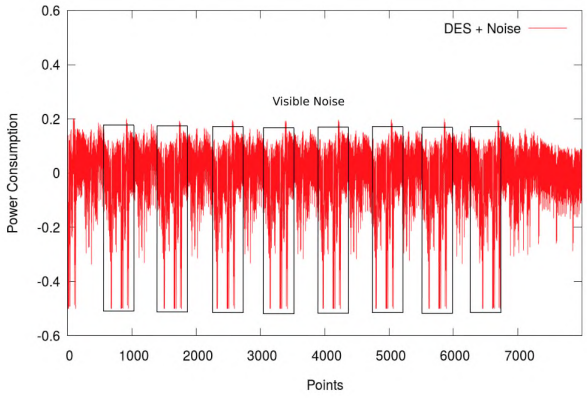


Figure 5.32: Visible noise within the DES implementation.

The same approach in Section 5.3.1 was followed as the mathematical instructions within the *printf* was modified to execute without the *printf* function. The CPA attack was performed with this new input data and the results indicated that no subkeys were recovered and the countermeasure had mitigated the CPA attack.

The research was able to recover the first round key of the DES implementation using the SCA attacks. Upon acquiring the key, it was used to recover the full encryption key. Furthermore, the proposed countermeasure was tweaked to match the DES leakage and prevented the SCA attack against the implementation.

5.4 Summary

This chapter discusses the attacks and defence capabilities of this research against smartcards and microcontrollers. A basic Side Channel Analysis (SCA) attack against smart-

cards demonstrated that different power signature was generated as the input data was changed. The results indicated that more bus lines are required to switch from a precharge state to a high state from a smartcard. This information was utilised to investigate the effects of using two different Java based compilers to compile code for the smartcard. The results demonstrate that utilising a different compiler and/or smartcard resulted in different power and EM emissions leakage. The EM emissions were captured from the smartcard as the AES-128 encryption algorithm was executed. This was followed by digital signal processing techniques to increase the signal-to-noise ratio and recover the secret key used by the AES-128 encryption algorithm from a smartcard.

Although this research was able to perform a successful SCA attack. The implementation of a multi-threading software based countermeasure could not be implemented as the smartcard had no support for multi-threading. Therefore, this research moved from smartcards to a pure microcontroller where there is more flexibility in terms of implementing a multi-threading software based countermeasure.

The microcontrollers that were targeted was the ATmega328p and ATxmega128D4. The power consumption of these microcontrollers was monitored and captured whilst the AES-128 encryption algorithm executed. The captured data was used as input for a Correlation Power Analysis (CPA) attack. The results indicated that this research was able to replicate existing SCA attacks and perform a successful CPA attack based on power consumption of two different microcontrollers. The impact of a CPA attack was demonstrated as it required only 50 power traces for both microcontrollers to recover the full secret AES-128 encryption key.

The research introduced a novel countermeasure by utilising multi-threads as software countermeasure to mitigate SCA attacks on microcontrollers. The results demonstrate that the proposed countermeasure obfuscated and prevented full key recovery from a CPA attack. The power leakage model of the CPA relies on the power traces to have minimal changes as each subkey is attacked. The proposed countermeasure produced dynamic power traces on each execution, causing confusion to the CPA attack model and reducing the correlation accuracy for predicting a correct subkey.

The proposed countermeasure was implemented in the *VerifyPin* process. This process does a verification check on a password entered into the device. The countermeasure prevented the adversary from obtaining the password from the *VerifyPin* process. The *VerifyPin* process was modified to include each character check into a thread, this increased the security with a minimal time delay.

The research moved away from capturing data from the power lines of a device and focused on capturing the EM emissions produced by the device. Existing countermeasures such as hiding, shuffling and time delays were incorporated with the AES-128 implementation and compared to proposed countermeasure. The results of this research demonstrated the ability to retrieve the secret key from an unprotected AES-128 implementation as well a protected implementation with the hiding, shuffling, and random delays countermeasure in place. The full secret key recovery from the proposed countermeasure was unsuccessful as the proposed countermeasure's unique designed allowed the algorithm to resist the CPA and the template based attack.

Additional signal enhancement techniques such as the Elastic Alignment technique was utilised to increase the success ratio and decrease the complexity of the attack. This was demonstrated as only 100 traces were required to recover the secret information as opposed to utilising 100 000 traces with no additional signal processing. The results demonstrated that full key recovery was possible on existing countermeasures. However, the proposed countermeasure withstood these additional procedures.

Although the AES-128 encryption algorithm was a major aspect of this research. Additional experiments were performed to evaluate the proposed countermeasure for microcontrollers. An unprotected AES-256 implementation was implemented on the microcontrollers and successful key recovery was achieved by utilising the CPA attack. The proposed software based countermeasure was implemented within the AES-256 algorithm and prevented a SCA attack on the 13^{th} and 14^{th} round of the AES-256 implementation to prevent the secret key being recovered via an SCA attack. The research was able to recover the first round key of the DES implementation using the SCA attacks. Upon acquiring the key, it was used to recover the full encryption key. The proposed counter-

measure was configured to match the DES leakage and prevented the SCA attack against the implementation.

The analysis and findings in this chapter will be utilised in Chapter 6 to benchmark against a Raspberry Pi, where existing and new attack vectors will be utilised in the recovery of secret information. Finally, an in depth discussion with regards to the proposed software based countermeasure is presented in Chapter 7.

Theres no silver bullet solution with cyber security,
a layered defence is the only viable defence.

James Scott

6

Raspberry Pi

THIS chapter discusses the experiments and results as Side Channel Analysis (SCA) attacks were performed against a Raspberry Pi. The chapter further focuses on the development of a potential multi-threading software based countermeasure to mitigate SCA attacks against a Raspberry Pi.

The capturing of Electromagnetic (EM) data from a Raspberry Pi and the conversation of this raw data into meaningful data is discussed in Section 6.1. This is followed by Section 6.2 where the meaningful information is utilised as input for the attack procedure with the hopes of recovering the secret key from the AES-128 algorithm.

The effects of various compilers and compiler optimizations on the EM information leaked is discussed in Section 6.3. It will be demonstrated that the various compilers and optimization flags can lead to different secret information being recovered.

The proposed multi-threading software based countermeasure for the Raspberry Pi to mitigate SCA attacks is elaborated in Section 6.4. This section has multiple iterations of improving the proposed countermeasure and attacking the improved versions of the countermeasure. The chapter concludes in Section 6.5 with an in depth analysis on the novel attack methods and multi-threaded software based countermeasure.

6.1 Obtaining Electromagnetic Information

A general approach taken by this research to capture EM information from a Raspberry Pi and transform the EM raw data into meaningful information is discussed in this section. In order to achieve this process and recover the secret key from the EM data, the procedure depicted in Figure 6.1 was followed.

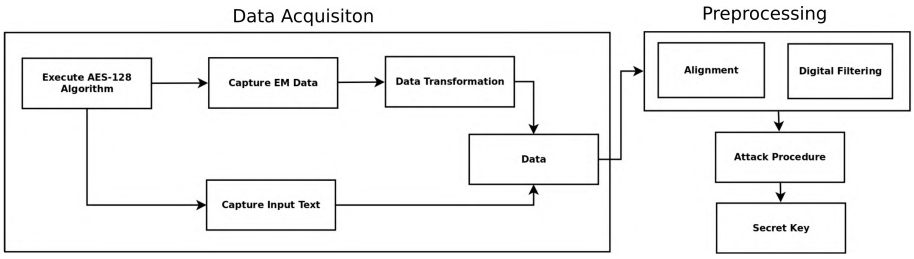


Figure 6.1: Process to capture, transform and recover secret information.

The procedure consists of three main elements: the data acquisition, preprocessing and attack procedure. The preprocessing procedure which consists of trace alignment and digital filtering has been elaborated in Section 4. The attack procedure of utilising the Correlation Power Analysis (CPA) technique to recover the secret keys from the EM data will be discussed in Section 3.5. Section 6.1.1 will discuss the process to capture EM emissions, followed by a discussion on the appropriate location to place the EM probe in Section 6.1.2. The attack procedure and techniques applied to retrieve secret information will be discussed in Section 6.2.

6.1.1 Data Acquisition

This section focuses primary on the capturing and transformation process of EM emissions from the Raspberry Pi. A detailed procedure for capturing and transforming the EM data can be seen in Figure 6.2. This process forms part of the capture and transformation procedures as seen in Figure 6.1.

The process commences by capturing raw signals from the Raspberry Pi while utilising the Funcube Dongle Software Defined Radio (SDR) as introduced in Section 4.4.3. GNURadio

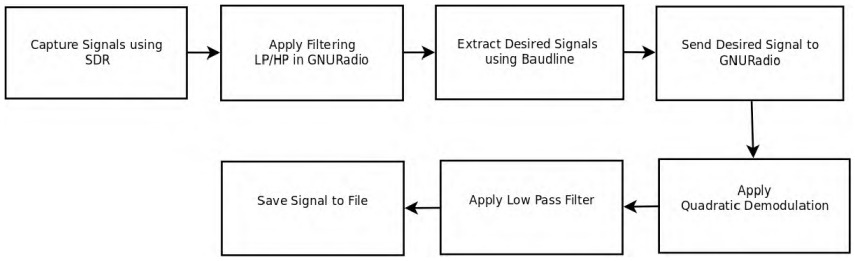


Figure 6.2: Process used to extract useful information from the Raspberry Pi.

was used to apply a low and high pass filter on the raw signals within the target frequency of 600Mhz as seen in Section 4.1.3. The resultant signal was sent to Baudline where the region of interest was extracted. The region of interest was sent back to GNURadio where quadratic demodulation and a low pass filter was applied. The resultant signal was sent to the preprocessing phase where trace alignment and denoising techniques introduced in Chapter 4 were applied. The exact techniques utilised will be discussed in Section 6.2.

As introduced in Section 4.4.3 this research utilised two Raspberry Pi's The first device served as the victim, while the secondary Pi was the attacker. The lubuntu 14.04[1] Operating System with the Linux 3.18.0-20-rpi2 kernel[2] were used. To limit the CPU from using internal step-up controls to adjust power and CPU frequency the victim's maximum CPU frequency was configured to 600 MHz. Limiting the frequency is a viable option as it has been mention in Section 3.1.2. Additionally, no adjustments were made to the secondary device. The experimental setup can be seen in Figure 6.3.

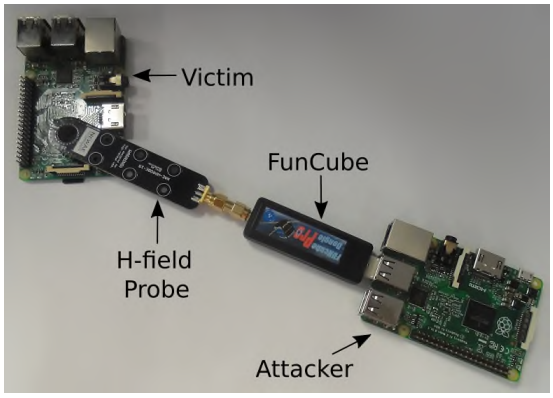


Figure 6.3: Experimental setup.

[1]lubuntu 14.04 – http://lubuntu.net/blog/lubuntu-1404-trusty-tahr-released
[2]Linux 3.18.0-20-rpi2 kernel – https://gist.github.com/pdp7/440f522ff7060ab9ae72

There are three stages to which the EM data can be analysed (Nakano *et al.*, 2014). The first stage consists of computing the Fast Fourier transform (FFT) over the baseband waveform. This process assists in establishing a frequency signature for various operations, as different operations would produce a specific pattern. Figure 6.4 illustrates the raw signal after it had been processed through an FFT within GNURadio. The signal was obtained by monitoring the desired 600 MHz frequency in real time via the SDR. This research followed the approach by Balasch *et al.* (2015) and Genkin *et al.* (2016) to limit the CPU frequency to the base frequency of the device. Therefore, the 600 MHz frequency was selected as it was the base frequency of the Raspberry Pi 2.

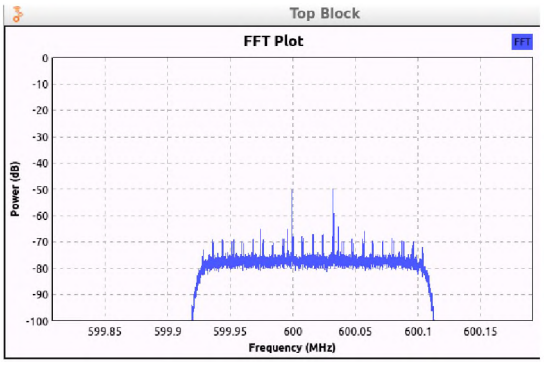


Figure 6.4: Fast Fourier Transform (FFT) around the 600 MHz signal.

The second stage is to select the region of interest at the point of leakage. This is done visually as it can be seen in the time domain in Figure 6.5. The region of interest is displayed in Figure 6.5 by a rectangle encapsulating the peak signal. This illustrates to adversary the location in time and the type of EM signature produced.

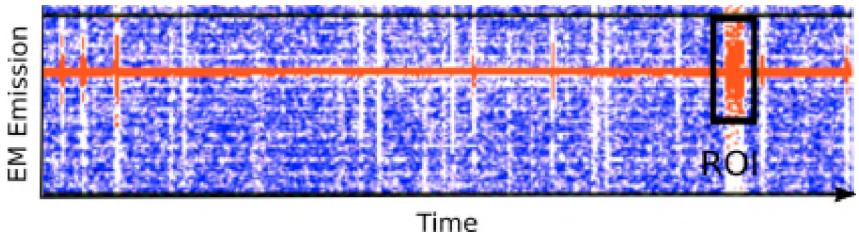


Figure 6.5: Region of interest highlighted in the amplitude.

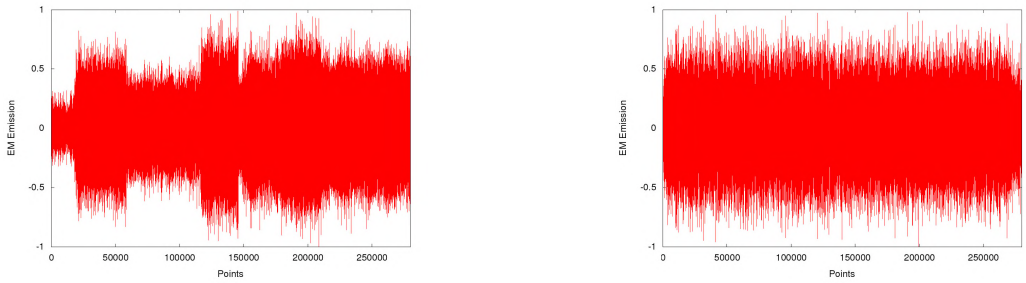
Digital filtering and denoising techniques are used to remove noise and increase the signal-to-noise ratio in the third stage. These techniques have been discussed in Chapter 4

While the test algorithms were executed on the victim's device, the EM data was captured at 384 kHz as oppose to capturing at 44.1 kHz or 192 kHz as discussed in Section 4.4.3, as sampling at a higher rate allows for more data to be obtained. Digital filtering was applied to keep all signal information between 0 – 50 kHz, while the rest was of signal frequency, was discarded. The resultant signal was sent to Baudline where the point of interest was extracted and sent back to GNUradio for demodulating and filtering.

6.1.2 EM Probe Placement

This section discusses the experiments conducted to establish the preferred location to place the EM probe in order to recover usable information. In order to capture and detect the EM leakage from the Raspberry Pi, the process introduced in Figure 6.1 was followed.

Based on the literature review in Chapter 3 there are three possible locations to place the EM probe. The locations are the voltage regulator, the CPU, and a target component on the device, i.e: the cryptographic hardware processor. In this research, the EM probe was placed over the CPU and voltage regulator as various test code was executed. The test code consisted of executing various arithmetic operations. It is noted that the Raspberry Pi does not have a dedicated hardware processor for cryptographic operations.



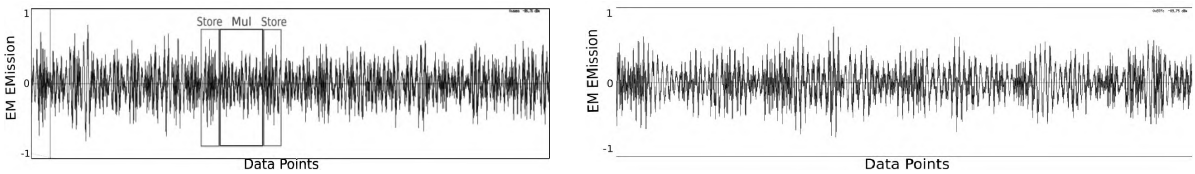
(a) CPU. (b) Voltage regulator.

Figure 6.6: EM emissions from the (a) CPU and (b) voltage regulator.

One of these arithmetic operations was to multiple numbers within a loop that executed 50 000 iterations. Figures 6.6a and 6.6b illustrate the EM emissions from the CPU and voltage regulator, respectively. While the code was executing, the EM emissions was captured from both the CPU and voltage regulator.

Figure 6.6a illustrates that the leakage from the CPU was variable and this would allow for the detection of specific operations in the code. Additionally, the leakage from the voltage regulator depicted in Figure 6.6b is constant, thus locating a region of interest becomes difficult.

A zoomed in view of the EM emissions is depicted in Figure 6.7 from the Baudline. The data from the voltage regulator in Figure 6.7b displays the signal as repeatable and constant over time, which relates back to Figure 6.6b. The process of storing a number, followed by multiplying the number can be visibly seen in Figure 6.7a by the highlighted boxes.



(a) CPU. (b) Voltage regulator.

Figure 6.7: EM emissions from the (a) CPU and voltage regulator (b) in Baudline.

The experiments in this section clearly indicate that it is more beneficial to extract data from the Raspberry Pi's CPU than the voltage regulator.

6.2 Acquiring Secret Information

The research utilises the knowledge described in Sections 6.1.1 and 6.1.2 to implement a procedure that extracts the EM emissions and utilises the information to recover the secret key of the AES-128 cryptographic implementation on a Raspberry Pi. The initial experiments and results in this section have been published in Frieslaar and Irwin (2017d).

The first experiment involved capturing the EM emissions as the victim executed the AES-128 algorithm of the libcrypto++ library[3]. The libcrypto++ library was chosen as it is a compatible cryptographic library used on Android smartphones.

[3]libcrypto++ – https://www.cryptopp.com/wiki/Linux

The victim executed the AES-128 encryption algorithm 30 times while the EM emissions were captured. After each execution of the cryptographic algorithm, there was a two second waiting period before the next execution took place. This is to cater for the built in interrupt timers of the Raspberry Pi. For this experiment, no trace alignment and denoising techniques were applied. The data was utilised in the CPA attack and the results revealed that only one subkey was recovered. Although one subkey was recovered this indicates that the Raspberry Pi was leaking out secret information.

Now that the research has established the Raspberry Pi leaks out subkeys, the Elastic Alignment technique mentioned in Section 4.2.3 was applied to the 30 traces and the resultant aligned data was used as input for the CPA attack. This approach provided in the recovery of six subkeys as seen in Table 6.1.

Table 6.1: Six subkeys that were recovered.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	Y	Y	-	-	-	-	-	Y	Y	-	-	-	-	Y	Y	-	6

Since the system was able to retrieve six subkeys based on Elastic Alignment alone, it is possible to retrieve the entire secret key if good alignment was achieved as discussed in Section 3.8. In order to achieve good alignment and recover secret information the procedure in Figure 6.8 was followed.

Firstly, the signal is segmented into three smaller partitions. Each partition is aligned by using the Elastic Alignment technique. The resultant signal is concatenated into one signal and is used as input data for the attack process or it can be sent to the denoising procedure. If the attack procedure is successful, i.e: secret information has been recovered, exit the routine, or else move to the resync procedure. Peak detection and the sum of absolute differences (SAD) techniques are used to compensate for trigger jitter and phase shift as discussed in Section 4.2. Once the resyncing is completed the adversary could either send the data to the attack procedure or apply denoising techniques to the data. Another scenario is to apply Elastic Alignment before sending the data to the attack procedure. The attack procedure utilises the CPA attack algorithm as discussed in Section 3.5.

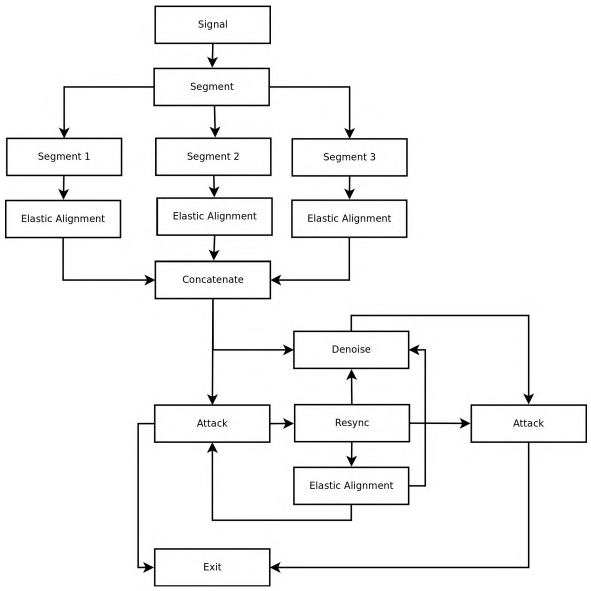
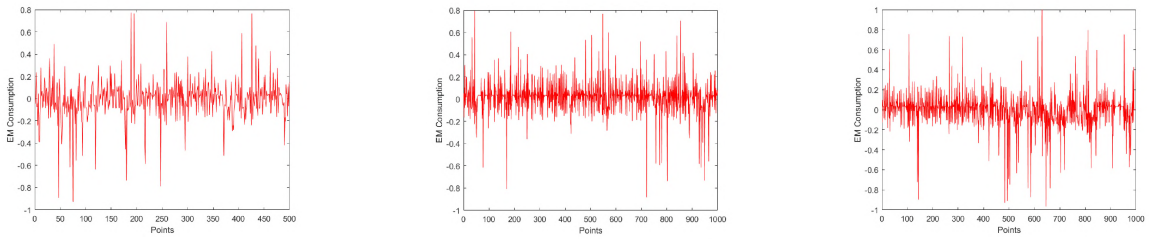


Figure 6.8: Flow diagram to align the signal and recover secret information.

A new experiment was conducted, which involved capturing a 100 new traces and following the procedure depicted in Figure 6.8. Firstly the demodulated signal was segmented into three. Each sample trace consists of 2500 points, thus the three groups ranged from points 0 500, 501 1500, and 1502 2500. as seen in Figure 6.9.



(a) Segment A (0 500). (b) Segment B (501 1500). (c) Segment C (1501 2500).

Figure 6.9: Three segmented signals ranging from various data points.

The Elastic Alignment technique was applied to the concatenated signals. As a result, the signals were merged back into one signal. The aligned data was sent to the attack procedure where the CPA attack was applied.

The design of the system allowed for various combinations to be performed, in order to recover the subkeys as mentioned earlier, two additional experiments were conducted. Firstly the data was sent to the attack procedure when no alignment and denosiong was

applied and secondly, the attack procedure was directly utilised after the signals had been concatenated. Since the CPA attack allows to target each subkey individually, the system was designed to apply the various alignment techniques at different stages. Following this approach, the system was able to successfully retrieve 12 of the 16 subkeys as depicted in Table 6.2.

Table 6.2: Subkeys recovered as different scenarios were utilised.

	Subkey Recovered																
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	Total
No Alignment	-	-	-	-	-	-	-	-	Y-	-	-	-	-	-	-	-	1
Elastic Alignment	-	Y	-	-	Y	Y	-	-	Y	-	-	-	-	Y	Y	-	6
Combined Techniques	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	Y	Y	Y	12

Table 6.2 illustrates that subkeys 9 – 12 were not recovered, thus reiterating the need to have good alignment in order to retrieve the remaining subkeys. The colour shading in the table represents a correct subkey predicted that was not located in the previous row. The results indicate that the Raspberry Pi was vulnerable to SCA attacks even though it is a complicated high powered device executing multi-cores with advance power management. The success ratio of the CPA attack with the addition of various alignment techniques are depicted in Figure 6.10.

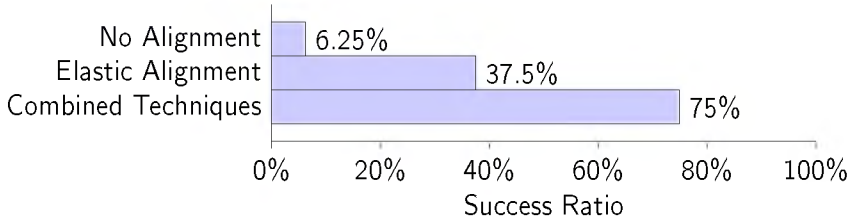


Figure 6.10: Success ratio of the CPA attack with various alignment techniques.

The *No Alignment* technique represents the results as no alignment was applied, followed by the results achieved as *Elastic Alignment* was applied on the segmented signals, finally, the results of applying various combinations of elastic alignment and resyncing techniques to align the signal are categorized under *Combined Techniques*.

Based on Table 6.2 and Figure 6.10 it was demonstrated the research was able to recover one subkey without any digital modification such as trace alignment. By applying the

Elastic Alignment technique on its own, six subkeys were recovered, this was followed by the recovery of 12 of the 16 subkeys as multiple techniques were utilised. Hitherto, there has not been any research that has utilised EM emissions to recover secret information from a Raspberry Pi.

6.2.1 Brute Force Attack

Although only 12 of the required 16 subkeys were recovered in the previous section. Brute forcing the remaining 4 subkeys becomes a trivial task. The attack procedure included a *C++* program to brute force the four remaining subkeys. Since each subkey could be 256 possibilities as discussed in Section 3.5, a quadruple nested loop was created. Inside the loop, the permutation of the four subkeys was generated and compared until the correct subkey was achieved.

The results obtained from the brute force attack indicated that the remainder of the subkey was recovered in six minutes. Further testing was carried out and the correct last four subkeys were changed to determine the impact on the timings. In this scenario it only took one minute to recover the remaining subkeys. Analysing the code, it was determined that the position in which the vector that stored the 256 possible outputs per subkey effected the timings. Therefore, the first test was reassessed. In this case the nested loop will be searched bottom-up, instead of top-down. The result indicated that one minute was required in this case.

Since this research is based on multi-threading, it is only fitting to create a new brute force program that will contain a multi-threading solution. Two threads would be created, one thread would search through the vector top-down, and the other thread would search bottom-up. Once the correct subkeys are recovered, the program will exit.

Table 6.3 illustrates the time taken to brute force four subkeys in seconds using various approaches. The first column in the table represents the method used to brute force the subkey. The methods consist of the top-down approach, bottom-up, and the multi-

Table 6.3: Time taken to brute force four subkeys.

Method	Location	Time (s)
Top Down	Top	3.3
	Mid	58.5
	Bot	362.3
Bottom Up	Top	384.4
	Mid	57.8
	Bot	2.2
Multi-Thread	Top	2.5
	Mid	44.3
	Bot	2.3

threading approach. The general location of the subkeys in the vector is depicted in column two. Finally, column three displays the time taken to guess the correct subkeys.

A trend occurs in both the top-down and bottom-up approach as indicated by the results. If the location of the subkeys are at the far end of the vector and the search starts at the near end of the vector, over six minutes is required to guess the correct four subkeys. If the start of the search and the location of subkeys are close to each other, between 2–3 seconds was required. Additionally, if the location of the subkeys is in the middle of the vector, under one minute is needed to obtain the correct subkeys. The multi-threading approach increases the attack capabilities by reducing the time taken to obtain the correct subkey. The average time to guess 1–3 subkeys were calculated and the results are depicted in Table 6.4.

Table 6.4: Average time to brute force three subkeys.

Subkeys Required	Time (μs)
1	0.076
2	5
3	560

The result in the table is a combination of the subkeys being located in various positions in the vector. The average time is calculated. It is shown that an average of 560μs were required to guess three remaining subkeys. As the required subkeys decrease the time

complexity decreases. This is further evident when comparing the results with Table 6.4 as 6 minutes was required to obtain four subkeys. Although this research knows the secret key, it is trivial to stop searching for subkeys when the correct subkey is retrieved. However, if the key was unknown, knowing when to stop searching becomes a daunting task and is beyond the scope of these experiments and research.

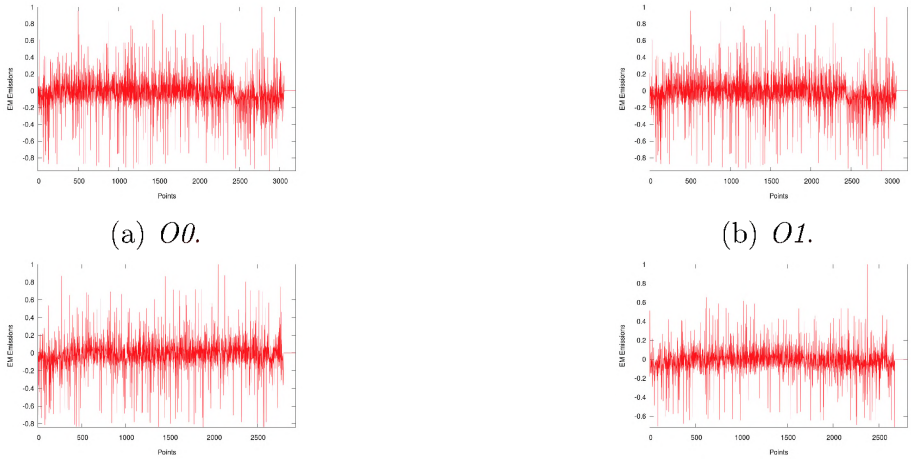
The AES-128 encryption algorithm of the libcrypto++ library executing on a Raspberry Pi was shown to be vulnerable towards the CPA attack as 12 of the 16 required subkeys were recovered. The result indicated that subkeys located at position 9 – 12 were not recovered. In order to retrieve the remaining subkeys, a brute force attack was performed. The results demonstrated that an average time of 28 seconds was required to brute force the remaining four subkeys.

6.3 Electromagnetic leakage from Compilers

This section discusses the impact various C/C++ compilers have on the EM spectrum. The compilers used by this research was the gcc, g++, and clang compilers as discussed in Section 3.12. As mentioned in Chapter 3, the effects of the GNU and clang compilers with different optimization levels have on the cryptographic binary at runtime will be investigated. The initial experiments and results in this section have been published in Frieslaar and Irwin (2017a).

As the theme of this research is multi-threading, the experiments in this section will encapsulate the AES-128 algorithm within a multi-threaded environment. The first set of experiments focused on compiling the cryptographic program with g++ while utilising the optimizations flags *O0* – *O3*. The optimized cryptographic program was executed on the Raspberry Pi and the EM emissions were captured during runtime. Figure 6.11 depicts the EM leakage at runtime from the cryptographic program that was compiled with g++ under different optimization flags.

It is observed a similar pattern occurs from Figures 6.11a – 6.11c, these are the optimizations flags of *O0*, *O1*, and *O3*. However, the optimization flag of *O2* resulted in a different



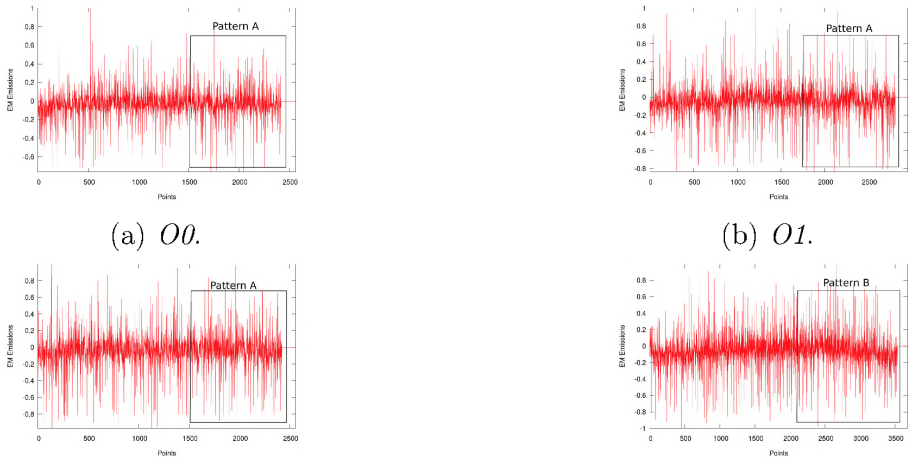
(a) *O0*. (b) *O1*.

(c) *O3*. (d) *O2*.

Figure 6.11: EM leakage as program was compiled with g++.

EM signature than the rest which was evident in Figure 6.11d as the EM leakage was less profound.

Figure 6.12 illustrates the EM leakage from the cryptographic program compiled with gcc using different optimization flags. Figures 6.12a – 6.12c depicts that utilising the optimization flags of *O0* –*O3*, has a similar EM signature. However, Figure 6.12d – *O3* flag – demonstrates a different EM signature from the rest. More data points are used, thus relating to a greater power draw as more EM leakage was detected.



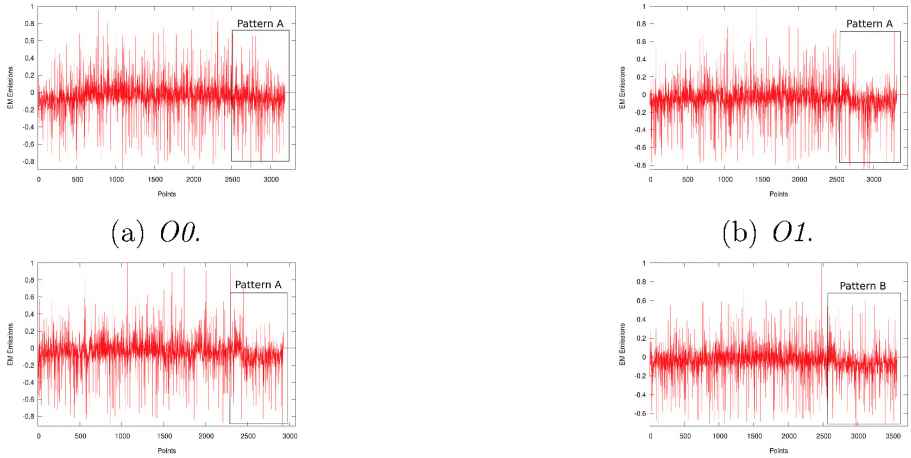
(a) *O0*. (b) *O1*.

(c) *O2*. (d) *O3*.

Figure 6.12: EM leakage as program was compiled with gcc.

The third set of experiments focused on the effects the clang compiler had on the EM

signatures. The clang optimizations range from *O0* – *O3*. The EM leakage from the cryptographic program compiled with clang using different optimization flags is depicted in Figure 6.13. Figures 6.13a – 6.13c illustrates the optimization flags of *O0* – *O2*, had a similar EM signature. However, Figure 6.13d – *O3* flag – demonstrated a different EM signature from the rest. A greater power draw was generated by the Raspberry Pi as seen by the increase in EM leakage, i.e: the data points end at 3500.



(a) *O0*. (b) *O1*.

(c) *O2*. (d) *O3*.

Figure 6.13: EM leakage from the cryptographic program compiled with clang.

The time taken in seconds to compile the AES-128 threaded algorithm on various compilers utilising different optimization flags was recorded and is depicted in Table 6.5. It is indicated the the clang compiler is the fastest compiler out of three. In terms of time, the gcc and g++ compilers perform relatively the same.

Table 6.5: Average time to compile source code.

O. Flags	**g++**	**gcc**	**clang**
0	13.05	11.79	09.14
1	16.55	16.40	09.69
2	22.26	22.43	10.24
3	22.69	22.70	10.41

Time in seconds.

The assembly code was analyzed as there is a strong correlation between the compiler and the architecture's machine code instructions. Listing 6.1 and 6.2 starts by illustrating the point of spawning the *pThread* inside the main method for the g++ assembler code.

Listing 6.1: g++ optimization flag *O0*.

```
1  bl   0 <pthread_create>
2  • str   r0, [r7, #24]
3  ldr r3, [r7, #24]
4  cmp r3, #0
5  ...
6  ...
7  ble.n 97c <main+0x24>
8  bl   0 <clock>
9  str r0, [r7, #28]
10 ldr r3, [r7, #28]
11 vmov  s13, r3
12 • vcvt.f32.s32 s14, s13
13 ldr r3, [r7, #20]
14 vmov  s13, r3
15 vcvt.f32.s32  s15, s13
```

Listing 6.2: g++ optimization flag *O3*.

```
1  bl   0 <pthread_create>
2  • mov   r4, r0
3  cbnz  r0, 90 <main+0x90>
4  bl   0 <clock>
5  movw  r1, #0
6  movt  r1, #0
7  mov r6, r0
8  vmov  s13, r6
9  vldr  s15, [pc, #80]  ; b8 <main+0xb8>
10 • vcvt.f32.s32 s0, s13
11 vmov  s13, r5
12 vcvt.f32.s32  s14, s13
13 vsub.f32  s0, s0, s14
14 vdiv.f32  s0, s0, s15
15 vcvt.f64.f32  d0, s0
```

Listing 6.1 displays that utilising the optimization *O0* resulted in the use of the ***str***, ***load***, and ***cmp*** instructions as opposed to only using a ***mov*** instruction as seen in Listing 6.2, as seen in line numbers 2 – 4. Before reaching the instruction set at line number 12 in Listing 6.1 and line number 10 in Listing 6.1 the optimization *O3* only had two instructions where as with the optimization *O0* enabled, many instructions were required. This relates back to the data in Figure 6.11d as fewer data points were used and the power spikes were fewer as the optimization flag *O3* was used. This is further confirmed from Table 3.3 in Section 3.12.3 as optimization *O3* reduces execution time.

Listing 6.3: clang optimization flag *O0*.

```
1   bl   0 <pthread_create>
2   str r0, [fp, #-28]
3   ldr r0, [fp, #-28]
4   cmp r0, #0
5   ...
6   ...
7   vldr  s2, [sp, #36] ; 0x24
8   vdiv.f32  s0, s2, s0
9   • vstr s0, [sp, #32]
10
```

Listing 6.4: clang optimization flag *O3*.

```
1   bl   0 <pthread_create>
2   mov r5, r0
3   cmp r5, #0
4   bne e4c <main+0x170>
5   bl   0 <clock>
6   ...
7   vldr  s2, [pc, #276]
8   vdiv.f32  s0, s0, s2
9   • vcvt.f64.f32 d0, s0
10
```

The comparison between the assembly code using optimization flags *O0* and *O3* for the clang compiler is displayed in Listings 6.3 and 6.4.

A reoccurring theme is illustrated as the code indicates that utilising the optimization flag *O3* produces less machine code. Although the code is shorter, there are additional instruction set used in optimization *O3*, as seen at line number 10 in Listing 6.4. The ***(vcvt.f64.f32)*** instruction set converts between single-precision and double-precision numbers.

This explains why the EM signature in Figure 6.13d was different from the rest of the EM signatures produced by the clang compiler as double precision uses twice as many bits as single, 32 bits for a single and 64 bits for a double. As more bits are used, more bus lines are required, hence more power(data points) are generated.

This section discussed the effects the gcc, g++, and clang compilers had on the EM signature of a cryptographic program. The AES-128 cryptographic program within the multi-threaded framework of *pThreads* was used as the base program. As the program was compiled, different optimization flags were used. The results indicated that using the optimization flag of *O3* changed the EM signature of the program. The assembler code was analysed and the g++ implementation displayed that fewer instructions were required which related to less EM emissions leaked.

Although fewer instructions were used by the clang compiler the EM trace had more data points which relate back to Table 3.3 in Section 3.12.3 as it is possible that the compiler optimization would increase the runtime of the executable binary. The analysis demonstrated that using the optimization flag of *O3* in the clang compiler had enabled the usage of double precision numbers which required more bits to be used, thus the Raspberry Pi was utilising more power and leaked out more EM emissions.

6.3.1 Recovering Secret Information

This section discusses the experiments performed against the various compiler optimizations to recover secret information based on the EM emissions from the cryptographic executable that was compiled under different parameters.

The experimental setup was as follows. The EM emissions from the Raspberry Pi was captured while the AES-128 cryptographic binary executed. The executable binary was compiled with the different compilers and compiler optimizations as mentioned in Section 3.12.3. The execution and capturing was repeated 30 occasions, hence 30 EM traces were captured. Once the data was collected the preprocessing procedure in Figure 6.2 was applied and the resultant signal was used as input for the CPA attack.

Firstly, ten traces were used as input for the CPA attack and the results are depicted in Table 6.6. The table comprises four columns and rows. The first column represents the optimization flag used by the compiler, followed by columns two – four, depicting the number of subkeys recovered from the EM data.

Table 6.6: Subkeys recovered utilising 10 traces.

O. Flags	g++	gcc	clang
-O0	7	7	5
-O1	5	7	3
-O2	3	3	1
-O3	2	2	5

Table 6.6 reveals an interesting pattern: as more optimization were used the key recovery decreased, i.e: seven subkeys were recovered while no optimization (*O0*) was used and two subkeys were recovered as *O3* optimization was used for the gcc/g++ compilers. The clang compiler had a decrease in the number of subkeys recovered as more optimizations were used. However, *03* optimization for the clang compiler demonstrated that the same number of subkeys was retrieved as when no optimizations were used. This relates back to Table 3.3, where level three optimization increases the execution time of a program and was further evident in Figure 6.13d where more data points are produced in the EM signature. Finally, the clang compiler with *O2* optimization revealed the least subkeys.

Table 6.7 and 6.8 depicts the results of the CPA attack as additional EM data was added to input data for the attack. The structure of the table is the same as that of Table 6.6. The results in both tables revealed a reoccurring theme, as more input data was used for the CPA attack, the greater the number of subkeys was recovered. The green shading in the tables represents the fewest subkeys recovered.

Table 6.7: Subkeys recovered utilising 20 traces.

O. Flags	g++	gcc	clang
-O0	9	10	6
-O1	7	9	5
-O2	6	5	2
-O3	4	5	6

Table 6.8: Subkeys recovered utilising 30 traces.

O. Flags	g++	gcc	clang
-O0	12	12	9
-O1	10	11	7
-O2	8	8	5
-O3	7	7	9

As the optimization levels increase fewer subkeys were recovered as opposed to using no optimizations flags as seen by the highlighted boxes in the tables. Additionally, the clang compiler revealed the fewest subkeys recovered as optimisation *O2* was utilised. However, as clang optimisation *O3* was utilised the same results were achieved when utilising no optimisation as depicted by the red shaded boxes in the Tables.

The subkeys that were recovered in the previous experiments were further analysed. Tables 6.9 and 6.10 depicts the recovered subkeys for the GNU and clang compiler under the different compiler optimizations as 30 EM traces were utilised as input data.

Table 6.9: Subkeys recovered from the g++ compiler over the various optimizations.

	Subkey																
Flag	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
-O0	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	-	-	-	Y	Y	Y	12
-O1	Y	Y	Y	-	Y	Y	-	Y	-	Y	Y	-	-	Y	Y	Y	11
-O2	Y	Y	-	-	-	-	Y	Y	-	-	-	Y	-	Y	Y	Y	8
-O3	Y	Y	Y	-	-	-	Y	Y	-	-	-	-	-	Y	Y	-	7

Table 6.9 depicts that utilising no optimizations for the GNU compiler subkeys 0 – 8, and 13– 15 were recovered. Although, fewer subkeys were recovered while utilising optimizations *O1* and *O2*, subkeys 9–11 were retrieved. A similar observation is made for the

results of the clang compiler as seen in Table 6.10 where the number of subkeys recovered are decreasing. However, different subkeys are being recovered.

Table 6.10: Subkeys recovered from the clang compiler over the various optimizations.

	Subkey																
Flag	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
-O0	Y	Y	Y	Y	Y	-	Y	-	-	Y	-	-	-	-	Y	Y	9
-O1	Y	-	-	-	-	-	-	Y	-	Y	Y	-	-	Y	Y	Y	7
-O2	-	-	-	-	-	-	Y	Y	-	-	-	-	-	Y	Y	Y	5
-O3	-	Y	Y	Y	Y	-	Y	Y	-	-	-	-	-	Y	Y	Y	9

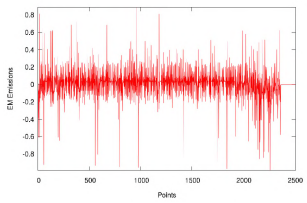
This section has demonstrated the ability to recover partial AES-128 cryptographic keys from a Raspberry Pi. The cryptographic program was compiled with various C/C++ compilers and optimization levels. The CPA attack revealed that the GNU compilers achieved a similar result with respect to the number of subkeys recovered. However, it was demonstrated that the subkey location changed as different optimizations were used. Summing up, the recovered subkeys, this research was able to retrieve 15 of the 16 AES-128 subkeys, with only subkey 12 not being recoverable. The fewest number of subkeys was recovered when the clang compiler was used with *O2* optimization. Although utilising higher levels of optimization, the results revealed that *O3* optimization for clang had the same effect as using no optimizations.

This research is the first of its kind to clearly demonstrate that the various compiler optimizations affect the EM signature of the binary executable. This is important as this information can be utilised by developers and system engineers to defend against known and future SCA attacks by optimizing compiler parameters in a configuration that can emit EM emissions to obfuscate adversaries or design a specific compiler that will enable an executable SCA resistant.

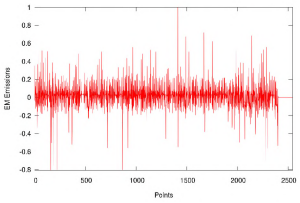
6.4 Developing a Countermeasure

This section discusses the proposed software based countermeasure to mitigate the SCA attacks against the Raspberry Pi. The proposed countermeasure utilises multi-threads to purposely leak out obfuscated information while the AES-128 algorithm cryptographic process is executing. Both the proposed countermeasure and the AES-128 algorithm would execute in a multi-threaded environment. Various multi-thread Application Program Interface (API)s are investigated. These APIs are *pThreads*; C++11 multi-threads; Threading Building Blocks (TBB) and OpenMP threads as discussed in Section 3.11. The initial experiments and results in this section have been published in Frieslaar and Irwin (2017b).

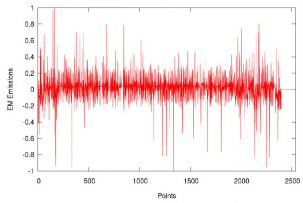
Experiment one comprised of the AES-128 algorithm encapsulated into the four multi-threaded APIs within one thread. The EM emissions were captured and is depicted in Figure 6.14. A better visual description can be seen in Figure 6.15 as the four implementations are overlaid onto one figure.



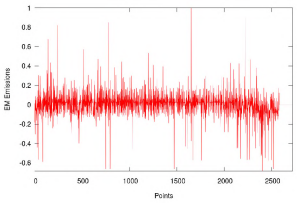
(a) POSIX implementation.



(b) C11 implementation.



(c) OpenMP implementation.



(d) TBB implementation.

Figure 6.14: EM leakage of the four cryptographic threaded implementations.

The results of this experiments illustrates a very similar pattern across all four implementations as seen in Figures 6.14a – 6.14a. This result was expected as only one thread was executing, which was the AES-128 cryptographic algorithm.

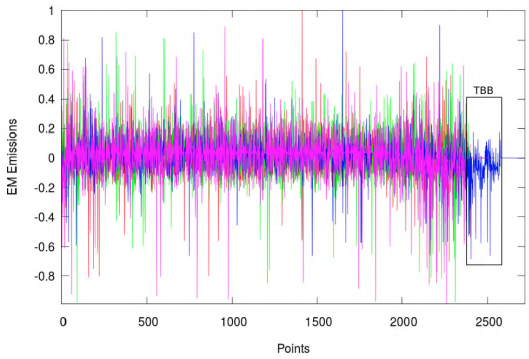
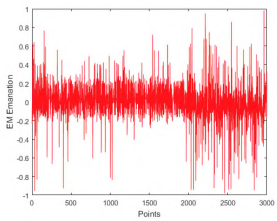


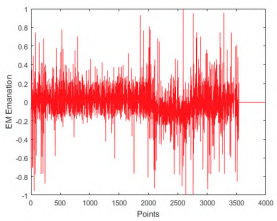
Figure 6.15: Overlaid representation of the EM leakage.

It is seen in Figures 6.14d and 6.15 the TBB implementation ends at a later stage than the other implementations. Even though the other implementations were similar in shape the TBB implementation in Figure 6.14d had a larger output, i.e: TBB had more than 2500 data points whereas the other implementations had less than 2500 data points. This is associated with the fact that TBB takes longer to execute the code and will be discussed later.

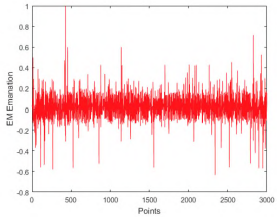
For the second experiment, the AES-128 algorithm was encapsulated into the four threaded API's with an additional noise thread. The noise thread calculated the first 1000 Prime numbers. The two threads executed in parallel with no synchronization and no thread had priority over the other. The EM emissions of the experiment are depicted in Figure 6.16.



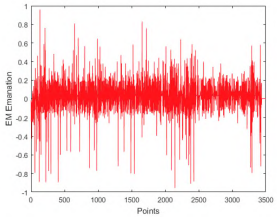
(a) POSIX implementation.



(b) C11 implementation.



(c) OpenMP implementation.

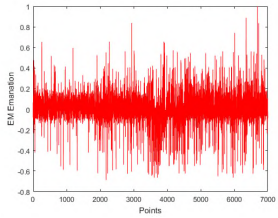


(d) TBB implementation.

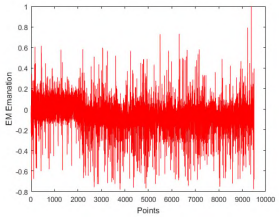
Figure 6.16: EM leakage with an additional thread calculating prime numbers.

Comparing Figures 6.14 and 6.16, the first notable finding is that the data points in Figure 6.16 has increased which was due to the added threaded as the data points are far greater than 2500. It is also noted that each implementation had a different EM signature.

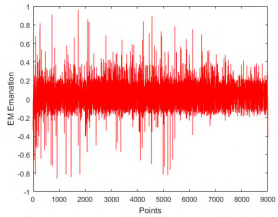
The third experiment changed the noise thread from calculating prime numbers to calculating the first 1000 Fibonacci numbers and the results of the EM emissions is depicted in Figure 6.17. The calculation of the Fibonacci sequence has increased the leakage dramatically, for example, the C11 implementation has seen an increase of 6500 data points.



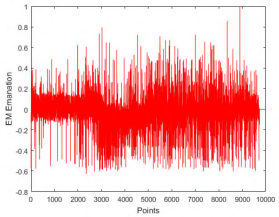
(a) POSIX implementation.



(b) C11 implementation.



(c) OpenMP implementation.



(d) TBB implementation.

Figure 6.17: EM leakage with an additional thread calculating Fibonacci numbers.

From Figures 6.17a – 6.17d the spikes in the EM emissions has increased as well as the execution time as compared to the findings of using one thread in Figure 6.14. Across all experiments – Figures 6.14 – 6.17 – the *pThread* implementation had the fewest number of data point while the TBB implementation had the largest data points. This can be further correlated to the average execution time depicted in Table 6.11.

The results continue by illustrating the average recorded execution times in milliseconds for the four thread techniques using the various programs in Table 6.11. The first column represents the four multi-thread implementations, followed by the execution time of cryptographic algorithm using one thread; the cryptographic algorithm with prime numbers executing on an additional thread; and finally the cryptographic algorithm alongside the Fibonacci threaded implementation. As mentioned before the implementation of the

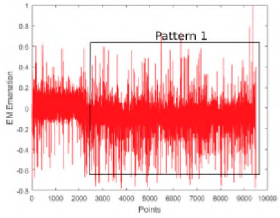
Table 6.11: Average run time using the four different multi-thread techniques.

	Threaded-Crypto	**libcrypto & Prime Threaded**	**libcrypto & Fibonacci Threaded**
C11	2,779	16,075	152,025
OpenMP	2,449	15,586	129,320
pThreads	0,593	13,826	117,252
TBB	6,377	31,602	151,316

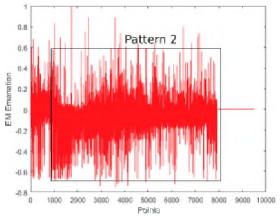
Time in milliseconds.

pThread was the fastest and resulted in the fewest EM leakage and the TBB implementation was the slowest with the highest EM leakage. Additionally, when computing the Fibonacci sequence 1000 instances the C11 is slower than the TBB.

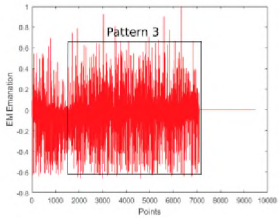
Additional analysis was performed. However, for this occasion only one multi-threaded API was selected. Multiple instances of calculating the Fibonacci sequence analogised the AES-128 execution was performed and the EM data was captured. Figures 6.18a – 6.18d depict that there are three distinct patterns and only Figure 6.18b and 6.18d have similar patterns. Figures 6.18a and 6.18c have a different pattern and end at a different location in time.



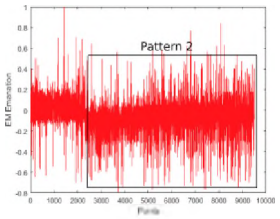
(a) POSIX implementation.



(b) C11 implementation.



(c) OpenMP implementation.



(d) TBB implementation.

Figure 6.18: EM leakage of the C11 implementations with an additional noise thread.

The CPU cores were monitored and the analysis indicated that the built in power management uses different cores on each execution of a program. Figures 6.19a and 6.19b

(a) Execution one.

(b) Execution two.

Figure 6.19: CPU process displayed by htop.

depicts the program htop[4] which monitors the CPU usage and other resources.

In the first execution, the second core was not used at all and more usage was derived from the cores 2 and 3 as seen in Figure 6.19a. On the second execution, all the cores were used and the utilisation of the cores are varied as depicted in Figure 6.19b. The multi-core processors distribute resources to other existing cores in order to complete the task effectively. During the cryptographic operation, the process was moved from one core to another due to the built in system interruptions and power management of the Raspberry Pi.

The results indicated that different patterns in the EM field occur as various multi-threaded APIs were used. The various multi-thread technique had variations in frequency and shape. The Raspberry Pi's power management system used different cores and resources on each execution which resulted in different workload and producing different EM signatures. The results demonstrated that *pThread* thread implementation has the fastest runtime execution, thus it produced the least amount of EM leakage as compared to the TBB implementation that leaked out the most EM emissions.

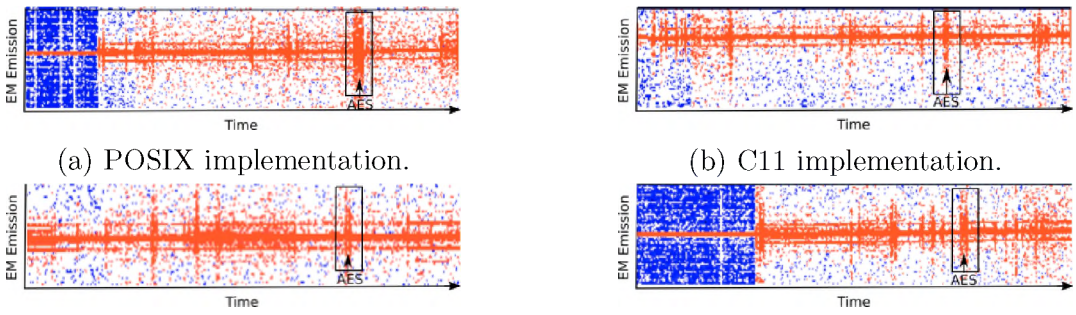
6.4.1 Developing a Noise Generator

The upcoming experiments will focus on the developmental of a software based noise generator to run hidden in the background as a daemon while the AES-128 cryptographic

[4]htop – http://hisham.hm/htop/

algorithm executes. Henceforth, the daemon noise generator will be referred to FRIES noise. The FRIES noise generator consisted of executing various arithmetic instructions in an infinite loop. These operations consisted of calculating the Fibonacci sequence, prime numbers, and other arithmetic instructions. The initial

The first experiment was to let the noise generator execute alongside the AES-128 cryptographic algorithm in a multi-threaded environment. Figure 6.20 displays The EM leakage as the AES-128 algorithm was executed on the four different threaded environments while the FRIES noise generator was executing in the background.



(a) POSIX implementation.

(b) C11 implementation.

(c) OpenMP implementation.

(d) TBB implementation.

Figure 6.20: EM leakage with a noise generator.

The annotations in Figures 6.20a–6.20d indicates the location of the AES-128 cryptographic algorithm in the time domain. This indicates that the process of the cryptographic algorithm is visible in the amplitude domain and the adversary would be able to extract useful information. However, there were occasions the cryptographic execution was not detected visibly.

The results indicated that as the prime numbers were executing from the FRIES noise generator, the cryptographic implementation was hidden. An additional set of experiments was carried out that configured the application to randomly select N and calculate that N amount of prime numbers while the cryptographic algorithm executed. The EM emissions of this experiment are displayed in Figure 6.21.

Observing the figure, the first rectangle represent the EM emanation as the prime numbers were calculated. The AES algorithm was executed within that time frame and it cannot

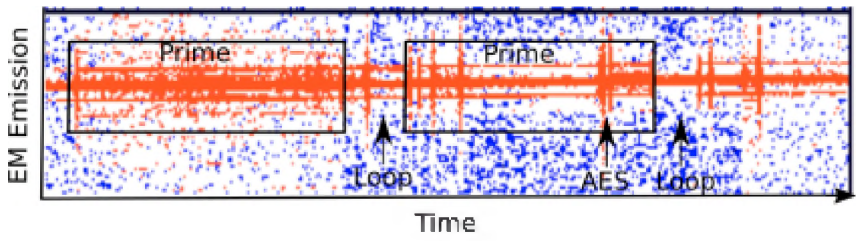


Figure 6.21: EM leakage as a noise program was executing.

be explicitly seen. However, in the second rectangle, the AES execution becomes visible during prime calculation. The data and code had been analysed in order to determine why in certain cases the cryptographic algorithm was visible. The analysis pointed in the direction that when low or few prime numbers were calculated there was a window where the cryptographic algorithm and nested loops could be visually detected as depicted by the annotations in Figure 6.21.

In order to determine why at certain cases the cryptographic algorithm was visible, the data was further analysed. The findings implied as few prime numbers were calculated there was a window where the cryptographic algorithm and nested loops could be seen visibly. This led the research to establish an additional experiment by calculating prime numbers after a 100 to 1000 repeatedly.

It is noted that the location of the AES-128 algorithm could not be located using the above approach. The EM emissions of the experiment in Figure 6.22, which is taken from Baudline as it has a better visual output. The waveform indicates that the signal is repeated over time, thus it is extremely difficult to locate the exact point of execution of the AES-128 algorithm.

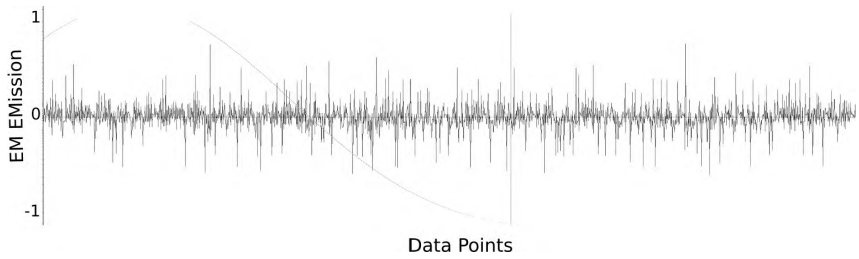
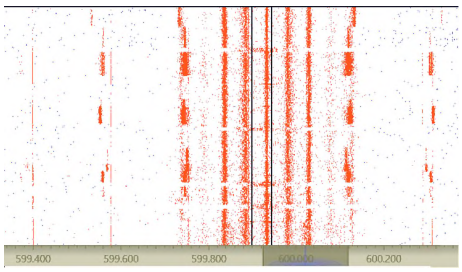


Figure 6.22: EM leakage depicting a repeatable waveform.

This research has identified additional data by monitoring the EM emissions across the

spectrum. As more power was used by the Raspberry Pi, more leakages appear across the spectrum. This can be related to voltage harmonics.

It is further evident, as more power and/or cores are used by the CPU in Figure 6.23a more data was available across the spectrum as seen in Figure 6.23b.



(a) The EM leakage across the spectrum.

(b) The CPU usage displayed by htop.

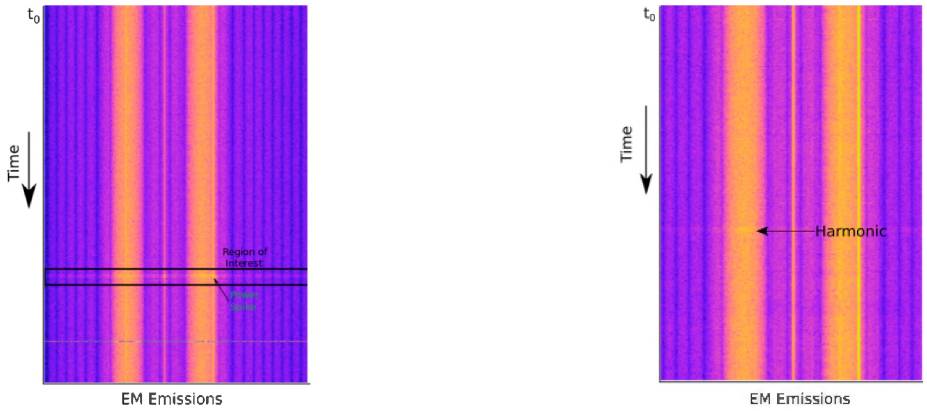
Figure 6.23: leakage (a) across the spectrum and (b) the core usage from the device.

6.4.2 Attacking the Proposed Countermeasure

This section discusses the attacks against the proposed FRIES noise generator introduced in Section 6.4.

Since the FRIES noise generator acts as a daemon by continuously executing in the background. The EM emissions were captured at various time intervals as the encryption process took place. The same procedure in Section 6.2 was utilised to evaluate the EM emissions. Each sample was 10 seconds in length. Referring back to Figure 6.2, one of the processes was to locate and extract the desired signals using Baudline. However, the noise generator obfuscates the detection of the region of interest. Figure 6.24 depicts a comparison between the signals in the time domain displayed in Baudline with and without the noise generator.

Figure 6.24a displays a power spike in the frequency as no countermeasure was in place. This was used as the region of interest and the data extracted revealed sensitive information as discussed in Section 6.1.1. The power spike has a thicker imprint, thus making it easier to view. However, Figure 6.24b imprint was constant over time, which was due

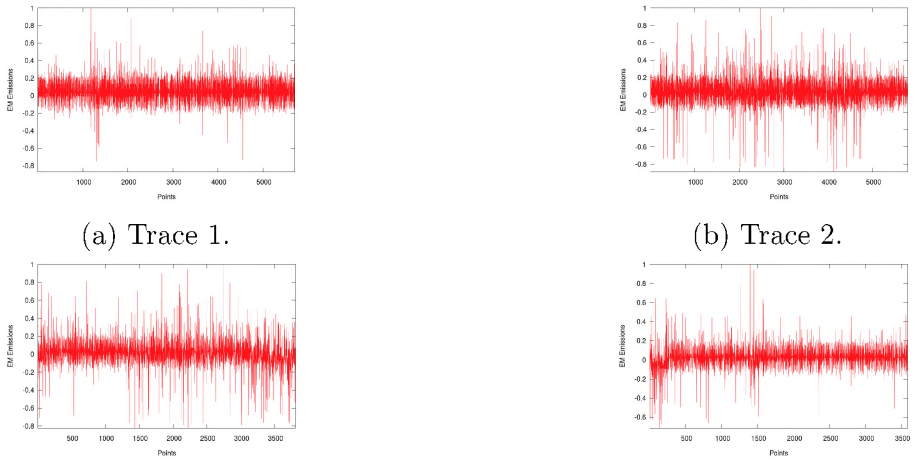


(a) Signal with no countermeasure. (b) Signal with the noise daemon.

Figure 6.24: EM emissions as (a) no countermeasure and (b) the FRIES noise generator was used.

to the noise generator. Although the imprint was constant, there was a faint harmonic spike. This research takes the role of the adversary and does not know the exact timing of the encryption process and the faint harmonic spike would be used as the region of interest going forward.

The same experimental setup was followed as discussed in Section 6.2 and 50 samples were collected. It was noted that after demodulation the signals had various EM spikes and were no longer consistent as before when the proposed countermeasure was not in place.



(a) Trace 1. (b) Trace 2.

(c) Trace 3. (d) Trace 4.

Figure 6.25: EM leakage as the FRIES noise was in place while.

Figures 6.25a – 6.25d demonstrates the variation in signal as the FRIES noise was in

place while the cryptographic algorithm executed. Each trace was different with regards to the shape and amount of leakage produced. The task to locate the exact location of the AES-128 visually became daunting.

Based on the EM leakage in Figures 6.25a – 6.25d the traces were manually inspected to locate any form of repeatable patterns that could be used as . The patterns were removed from the original data and used to form new traces. The resultant trace data was used as input for the CPA attack.

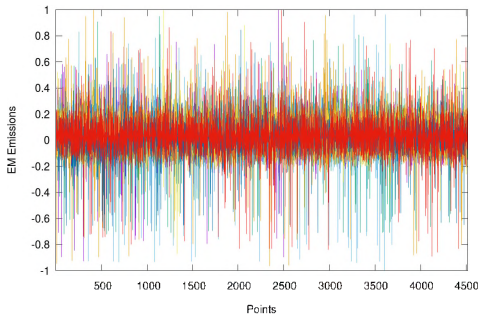


Figure 6.26: 10 traces manually reconstructed.

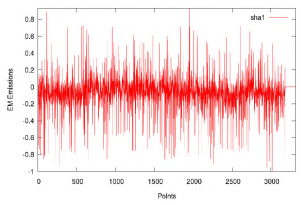
Figure 6.26 depicts 10 traces of the new data that was used for the attack. The results from the CPA indicated that the attack was unsuccessful and no subkeys were recovered. The next step was to apply the aligning techniques discussed in Section 6.2 to the data. The results acquired demonstrated that after the alignment techniques were applied, no further subkeys was recovered. Although, no subkeys were recovered the possibility to reverse engineer the EM emissions from the prime numbers is a strong possibility.

6.4.3 Improving the Proposed Countermeasure

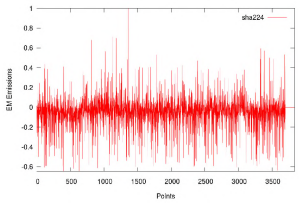
The FRIES noise was modified by replacing the prime number sequence with the Secure Hash Algorithms (SHA). The first set of experiments involved creating an EM profile for the various hashes functions. The SHA-1, SHA-224, SHA-256, and SHA512 functions were used. While the Raspberry Pi executed the cryptographic implementation and the new version of the FRIES noise generator, the EM emissions were captured. The capturing of the EM emissions remained the same as discussed in this section and the hash functions

are taken from the libcrypto++ library. The initial experiments and results in this section have been published in Frieslaar and Irwin (2017c, 2018).

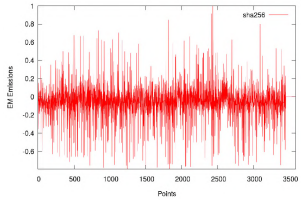
Figure 6.27 depicts the EM leakage from the Raspberry Pi as the various SHA functions were executed. As the different hash functions were used the EM signature changed, especially that of the SHA-512 function as seen in Figure 6.27d



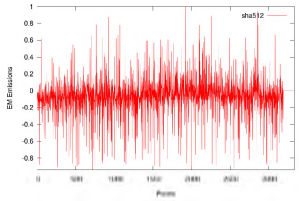
(a) SHA-1.



(b) SHA-224.



(c) SHA-256.



(d) SHA-512.

Figure 6.27: EM leakage for the various SHA functions from the libcrypto++ library.

Since the SHA functions are secure than the calculation of prime numbers. The noise daemon was configured to execute the SHA-512 hash function in an infinite loop. The process of generating the hashes is depicted in Figure 6.28.

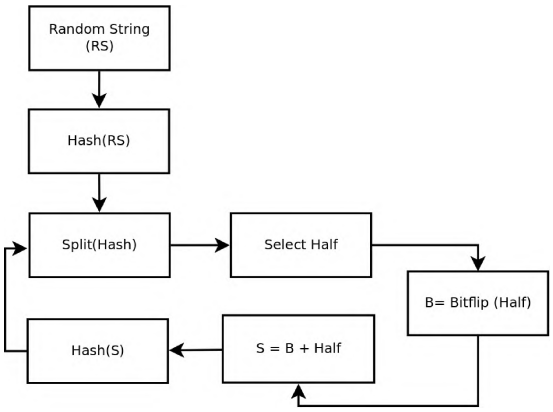


Figure 6.28: Process to generate random hashes for the FRIES noise generator.

A random alphanumeric string was generated and used as input for the SHA-512 hash function. The resultant hash value is taken and segmented into two. On each loop, a

random decision was made to take the upper or lower half of the segmented hash value. The segmented hash value was bit-flipped and appended to the previous half not selected. The resultant string was used for the SHA-512 hash function. This process was repeated indefinitely, thus increasing the entropy and generating random secure hashes.

While the FRIES noise daemon executed in the background the AES-128 algorithm was randomly executed over time. It was noted that after 40 generated hashes the CPU had reach 100% usage and started slowing the execution and the device down. In this slow down state, the process of the AES-128 algorithm could be clearly seen which is depicted in Figure 6.29.

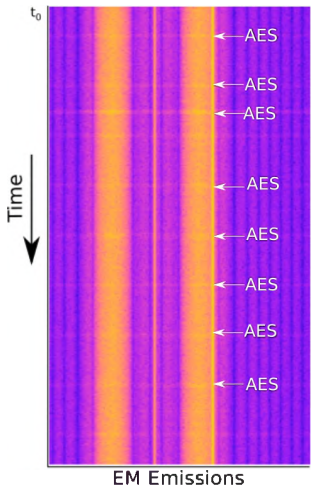
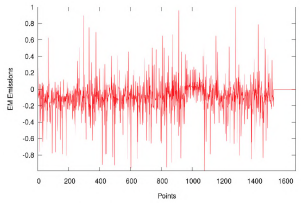


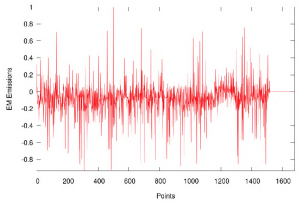
Figure 6.29: Runtime of the AES-128 algorithm while the FRIES noise was executing.

The OpenSSL libraries that included these secure hash functions were investigated. As mentioned before the same setup was used to capture the EM emissions while the various hash functions executed. Figure 6.30 depicts the EM leakage for the various SHA functions from the OpenSSL library.

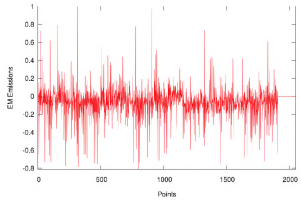
It is depicted that as the different hash functions are used the data points increase in length, as seen in Figure 6.30a (1500 points) to Figure 6.30d (2600 points). The increase in data points is related to the output size of each hash function, as the SHA-1 function has an output of 160 bits and the SHA-512 has 512 bits. The block size of the SHA-512 function is 1024 bits whereas the rest as a block size of 512.



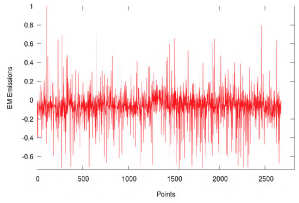
(a) SHA-1.



(b) SHA-224.



(c) SHA-256.



(d) SHA-512.

Figure 6.30: EM leakage for the various SHA functions from the OpenSSL library.

The FRIES noise generator was re-coded to utilise the native OpenSSL libraries. The first experiment involved stress testing the Raspberry Pi by executing the program in the background in an infinite loop. The test indicated the CPU usage was also at 100% load. However, there was no lag from the device and the AES-128 cryptographic process remained hidden. More hashes were being calculated per second as opposed to libcrypto++ library. This process demonstrated that the OpenSSL library was well suited to be implemented as a noise generator.

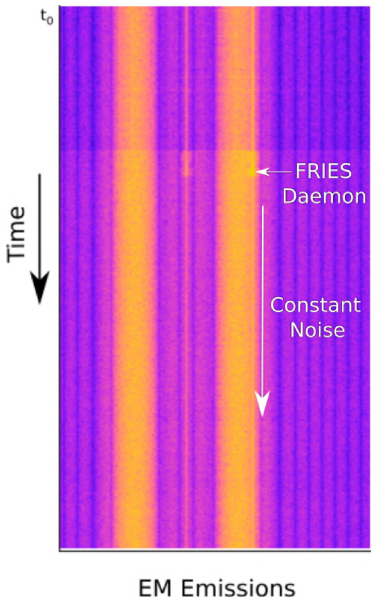


Figure 6.31: EM emissions depicting the startup point for the FRIES noise generator.

While the new implementation of FRIES noise was executing in the background the AES-128 program was executed on multiple occasions over time. The EM signature of the execution is depicted in Figure 6.31.

The spike observed in the figure which is annotated by the arrow pointing from the daemon is the startup process of the FRIES noise. Once the FRIES noise generator had been initialized a constant EM leakage was produced. The execution of the AES-128 algorithm could not be seen in the amplitude domain.

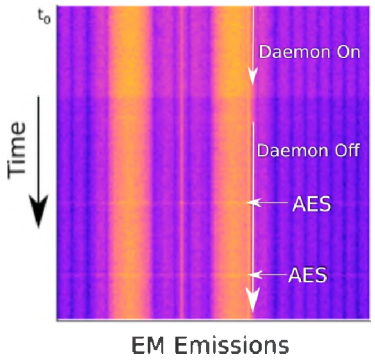


Figure 6.32: EM frequency as the FRIES noise is switched on and off.

A secondary experiment was conducted by alternating between the daemon being on and off. The EM data was captured and within this period the AES-128 cryptographic program was executed. Figure 6.32 illustrated the time domain of the EM signature. While the daemon is executing in the background, there are no additional scan lines. The figure further demonstrates that while the daemon is off the AES-128 cryptographic execution can be seen as annotated by the "*AES*" an arrow pointing to the spike in the EM spectrum.

6.4.4 Attacking the Improved Countermeasure

Although, the encryption process was well hidden the EM data from the previous section was further analysed. It was noticed that out of 80 samples, there were ten traces that gave off the slight harmonic noise. These traces were extracted and analysed. Figure 6.33 depicts the EM emissions for the harmonic leakage. It can visibly be seen that each trace

has various different outputs. The data was used as input for the alignment procedure as discussed in Section 6.2. The resultant aligned data was used in the CPA attack and the results indicated no secret information was revealed.

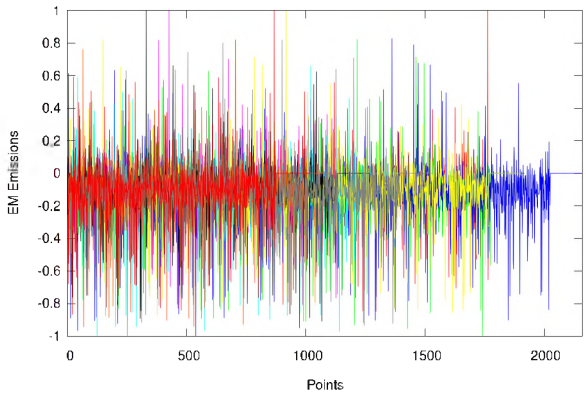


Figure 6.33: EM emissions acquire from harmonic leakage.

Statistical analysis was performed on the EM data. Two data sets were used. The first set being the data while the daemon was executing and the second set as the daemon was off. The following statistical values were calculated: the variance, covariance, mean and standard deviation as depicted in Table 6.12.

Table 6.12: Statical analysis while monitoring FRIES.

	Variance	**Covariance**	**Mean**	**Standard Deviation**
Daemon On	0.0114	0.0108	0.0251	0.1457
Daemon Off	0.0212	0.0251	-0.0714	0.1041

In order to determine the difference between the two signals, the variance of each set is summed. The variance is 0.0326. In addition, the baseline AES-128 cryptographic signal was used as a template for a cross-correlation test to determine whether the AES-128 EM signature was within the signal while the noise generator was enabled.

Figure 6.34 displays the Cross-correlation between the cryptographic template and the signal produced by the FRIES noise generator. The x-axis is a representation of time at the point of the AES-128 cryptographic signature being similar to that of the EM signature from the noise generator. The figure reveals that cross-correlation test could

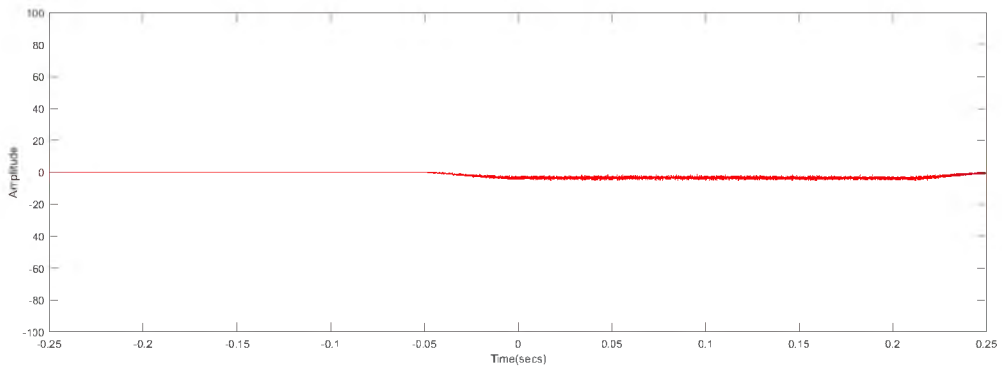


Figure 6.34: Cross-correlation results.

not determine a correlation between the AES-128 EM signature and the EM signature produced by the generator as there are no profound spikes in the figure.

6.4.5 Summary

A daemon noise generator known as the FRIES noise generator was introduced in this section. The results demonstrated that the FRIES noise generator had varied effects on the radio spectrum. In the first iteration of the FRIES noise generator, it was possible to visually detect the location of the execution of the AES-128 cryptographic algorithm. However, 64% (18/50) of the time it could not be visually detected. It was identified that the calculation of prime numbers was obfuscating the cryptographic algorithm. The analysis revealed high prime numbers were calculated there was a window where the cryptographic algorithm could not be detected visibly.

To improve the obfuscation of the FRIES noise generator, the prime number sequence within the FRIES noise generator was replaced by the libcrytpo++ implementation of SHA-512 hash functions. The results indicated that the CPU was utilising a 100% of resources and the device started to lag and slow down. In this slow down state, the AES-128 encryption program could be visibly seen and extracted.

The FRIES noise generator was re-coded to utilise SHA-512 hash functions within the OpenSSL libraries. A stress test was performed where the noise generator would execute indefinitely and the results demonstrated that the CPU usage was at full load and all cores

of the Raspberry Pi were being utilised. However, there was no lag from the device which resulted in more hashes being calculated per second and there was no clear indication of the AES-128 cryptographic execution.

Statistical analysis was performed on the FRIES noise generator, more importantly, the cross-correlation between the FRIES noise and a non protected AES-128 cryptographic implementation. The results revealed that there was no correlation between the two sets of EM signatures. Therefore, FRIES noise generator with the OpenSSL library is well suited to obfuscate EM SCA attacks.

6.5 Advance Analysis

Following the recovery of 12 subkeys in Section 6.2, this research aimed to uncover the remaining subkeys without utilising brute force. Firstly, the EM data was recaptured under the following conditions: The input data used in the encryption process would be more controlled as opposed to the randomisation of the input text in Section 6.2. The data would consist of 50 subsets, each set contains 10 EM traces with its own input key and output cipher. The EM signals would be averaged and combined to form a new data set with only the averages of the signals.

The signal and digital processing techniques were kept the same and it was applied to the 50 average EM signals. The resultant data was utilised as input for the CPA attack. The results of the CPA revealed that as before only 12 subkeys were recovered. The same 12 subkeys as depicted in Table 6.2 from Section 6.2 was recovered.

The second approach was to implement the approach of Genkin *et al.* (2017). Although their research involved recovering RSA secret keys, this research followed their signal processing procedure to determine if it was possible to recover the remaining secret keys with a different approach. Their approach is as follows:

1. The raw signal was passed through a Finite Impulse Response (FIR) filter (Neuvo *et al.*, 1984) with a cutoff frequency of 50kHz in the sidebands.

2. A high pass filter was applied to the resultant signal.

3. Hilbert transformation was applied to demodulated signal.

4. Finally, the demodulated signal was passed through a low pass filter.

Following their signal processing procedure, the data were aligned and the resultant data was sent to the attack procedure. The attack procedure revealed that only three subkeys were recovered as depicted in Table 6.13.

Table 6.13: Three subkeys that were recovered.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	-	Y	-	-	Y	-	-	-	Y	-	-	-	-	-	-	-	3

Based on the above results, this research decided to return to the original signal processing approach and experiment with the low and high pass filters. The design was adapted to firstly processes the raw signal through an FIR filter where 50 kHz would be kept in the sideband and the rest discarded. The resultant signal can be seen in Figure 6.35.

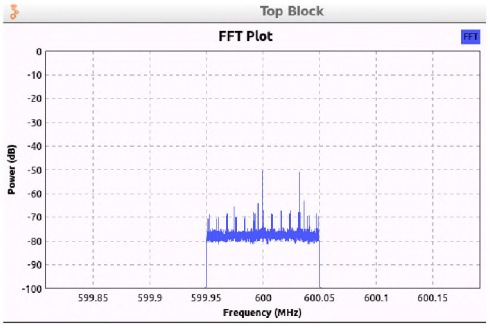


Figure 6.35: The signal after the FIR filter was applied.

The resultant signal was sent to a low and high pass filter where the frequency between 0 – 5 kHz was rejected and the frequency between 5 – 50 kHz was kept. Quadratic demodulation was applied to the signal which was followed by a bandpass filter that allowed frequencies between 5 – 50 kHz to be kept. The digital processing procedure remained untouched and the signals were passed through the alignment process. The aligned signals were utilised as input for the CPA attack and the results are depicted in Table 6.14.

Table 6.14: 12 subkeys that were recovered.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	Y	Y	Y	-	-	Y	Y	Y	Y	Y	Y	Y	Y	-	-	Y	12

The results in Table 6.14 displays that 12 subkeys were recovered. However, it was 12 different subkeys as opposed to Table 6.2. Interestedly, the subkeys 9 – 12 that was not recovered originally in Section 6.2 was now recovered.

The next experiment focused on modified the frequencies to be retained and rejected. The frequencies between 0 – 5 kHz and 10 – 50 kHz was kept and the remainder rejected. Table 6.15 indicates the subkeys recovered based on the input data as the frequency 5 – 10 kHz was removed.

Table 6.15: 15 subkeys that were recovered.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	15

The results from Table 6.15 illustrates that 15 subkeys had been recovered. This is three more subkeys than the initial experiments as discussed in Section 6.2.

Now that it has been established that removing certain frequencies can result in better results. The next experiment removed the frequencies between 10 – 15 kHz and kept the frequencies between 0 – 10 kHz and 15 – 50 kHz.

Table 6.16: 15 subkeys that were recovered with an alternative result.

	Subkey																
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
Recovered	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	15

Table 6.16 indicates the subkeys recovered based on the input data as the frequency 10 – 15 kHz was removed. Yet again 15 subkeys were recovered. It is observed that subkey 4 which was not recovered in the previous experiment had now been recovered. However, in this instance, subkey 9 was not recovered.

Since the removal of the frequencies between 5 – 10 and 10 – 15 kHz were overlapping with regards to the subkeys recovered. An additional experiment was carried out which removed all frequencies between 0 - 15 kHz, i.e: only frequencies between 15 – 50 KHz was kept. Following this approach this research was successfully able to recover the entire secret key used for the AES-128 cryptographic algorithm. Figure 6.36 depicts the number of subkeys as various frequencies were omitted and Figure 6.37 the change in the FFT as the frequencies were omitted.

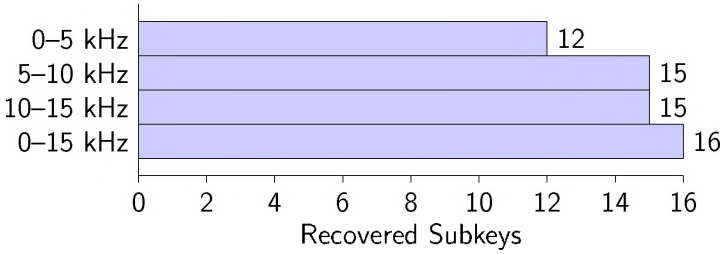
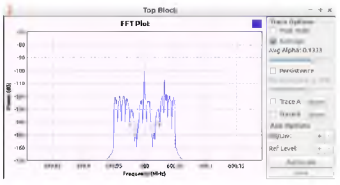
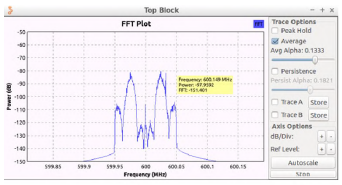


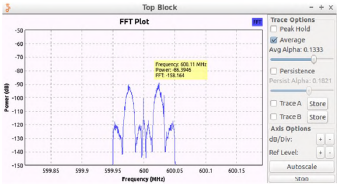
Figure 6.36: Number of Subkeys recovered as various frequencies were omitted.



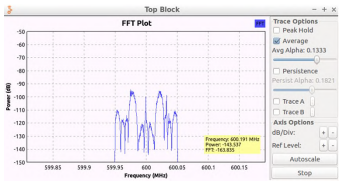
(a) 0 – 5 kHz.



(b) 5 – 10 kHz.



(c) 10 – 15 kHz.



(d) 0 – 15 kHz.

Figure 6.37: Signal represented by the FFT as various frequencies were omitted.

This research continued by performing a deeper analysis of the data by segmenting the 50 traces into 5 subsets of 10 traces each. Each stack would be sent as input data for the CPA attack. The results of the attack are depicted in Table 6.17.

The results in Table 6.17 indicates that utilising 10 traces, five subkeys were recovered, eventually as 50 traces were utilised the entire subkey was recovered. This trend relates back to the early experiments with the microcontrollers as more input data was used, more subkeys were recovered. In addition, 50 traces were also required to recover the full

Table 6.17: Subkeys recovered as various traces were utilised.

		Subkey																
Traces	**(Set)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
10	**(A)**	-	-	Y	-	Y	-	Y	-	Y	-	-	-	Y	-	-	-	5
20	**(B)**	Y	Y	Y	-	Y	-	Y	-	Y	-	-	-	Y	-	-	-	7
30	**(C)**	Y	Y	Y	-	Y	Y	-	Y	Y	-	Y	Y	Y	-	-	-	10
40	**(D)**	Y	Y	Y	-	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	-	-	12
50	**(E)**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	16

secret key when the microcontrollers were attacked. It is further observed that subkeys 2, 4, 8, and 12 were recovered in all the subsets and only subkeys 4, 10, 15 and 16 were recovered when 50 traces were used. Finally, The table depicts that subkey 6 was lost as 30 traces was used.

The input traces were reduced to determine the minimum traces required to recover the full AES-128 encryption key from a Raspberry Pi. The results illustrate that 14 subkeys were recovered as 45 input traces were used and 15 subkeys were discovered as 48 traces were utilised as highlighted by the box in the figure. The entire secret was uncovered as 50 input traces were used. Therefore, the minimum EM traces required were 50 aligned traces.

Table 6.18: Subkeys recovered as 45 – 50 input traces were utilised.

	Subkey															
Traces	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
45	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	14
46	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	14
47	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	-	Y	14
48	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	15
49	Y	Y	Y	-	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	15
50	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	16

The next experiment was to shuffle the order of the input traces. The subsets containing traces 11 – 20, and 21 – 30 were rotated with each other. The results are depicted in Table 6.19.

Table 6.19: Subkeys recovered as various traces were utilised with subsets shuffled.

		Subkey																
Traces	**(Set)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
10	**(A)**	-	-	Y	-	Y	-	Y	-	Y	-	-	-	Y	-	-	-	5
20	**(C)**	-	-	Y	-	Y	-	Y	-	Y	-	-	Y	Y	Y	-	-	7
30	**(B)**	-	-	Y	-	Y	-	-	-	Y	Y	Y	Y	Y	Y	Y	Y	10
40	**(D)**	Y	Y	Y	-	Y	-	-	-	Y	Y	Y	Y	Y	Y	Y	Y	12
50	**(E)**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	16

The results revealed that different subkeys were being recovered as opposed to the subkeys recovered in Table 6.17. Subkeys 13 – 15, and 11 were now being recovered. However, subkeys 0, 1, 5 and 7 were not able to be recovered. After 30 input traces, 10 subkeys were recovered. This serves as a base for the remaining 20 input traces as it can be seen that utilising 40 traces subkeys 15 and 16 remained whereas in the first experiment – Table 6.17 – these subkeys were not recovered.

The next experiment consisted of rotating subsets D and E. The results are depicted in Table 6.20, interestingly the results remained the same as displayed in Table 6.17. This can relate back to the previous experiment where after 30 input traces the CPA utilises the data as a base and it recovered the remaining subkeys easier.

Table 6.20: Subkeys recovered as subsets were shuffled again.

		Subkey																
Traces	**(Set)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
10	**(A)**	-	-	Y	-	Y	-	Y	-	Y	-	-	-	Y	-	-	-	5
20	**(B)**	Y	Y	Y	-	Y	-	Y	-	Y	-	-	-	Y	-	-	-	7
30	**(C)**	Y	Y	Y	-	Y	Y	-	Y	Y	-	Y	Y	Y	-	-	-	10
40	**(E)**	Y	Y	Y	-	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	-	-	12
50	**(D)**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	16

Further analysis was performed on subset C and it was uncovered that the EM trace data within this subset contained a lot of variations within the EM spectrum. Subset C was recaptured and put through the same digital processing techniques as mentioned in this section. The resultant data was integrated as the new subset C. The CPA attack was

performed against the new data and after 30 traces subkey 6 was not lost as with the experiments in this section. The results are depicted in Table 6.21.

Table 6.21: Subkeys recovered with better input data.

		Subkey																
Traces	**(Set)**	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**Total**
10	**(A)**	-	-	Y	-	Y	-	Y	-	Y	-	-	-	Y	-	-	-	5
20	**(B)**	Y	Y	Y	-	Y	-	Y	-	Y	-	-	-	Y	-	-	-	7
30	**(C)**	Y	Y	Y	-	Y	Y	Y	Y	Y	-	Y	Y	Y	-	-	-	10
40	**(D)**	Y	Y	Y	-	Y	Y	Y	Y	Y	-	Y	Y	Y	Y	-	-	12
50	**(E)**	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	16

6.5.1 Countermeasure

Following the recovery of the entire subkey, the EM data was recaptured while the proposed countermeasure of the FRIES noise generator was running. The same signal processing procedure used in the previous section – Section 6.5 – to recovery the entire secret was utilised for this experiment.

Firstly, it is noted that while the noise FRIES generator was executing, the process of the AES-128 algorithm could not be seen visibly, which is depicted in Figure 6.38. Nevertheless, the entire signal had to be utilised as input. The results achieved from the attack procedure indicated that no subkeys were able to be recovered. This demonstrates that the proposed FRIES noise generator mitigates a SCA attack and prevents the recovery of the AES-128 encryption key.

Since this research has established that it was possible to recover the entire secret key. The microSD card was replaced with an identical mircoSD card. However, in this case the lubuntu 16.04.2[5] Operating System with the Linux 4.4 kernel[6] was installed. The Operating System was changed as it was the latest version that works with both the Raspberry 2 and the new Raspberry 3.

[5]lubuntu 16.04.2 – https://goo.gl/4LjwXV
[6]Linux 4.4 kernel – https://github.com/raspberrypi/linux

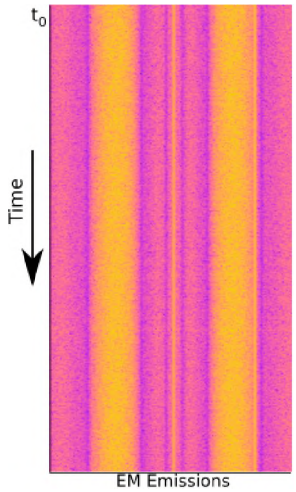


Figure 6.38: EM spectrum while the noise generator was executing in the background.

The same procedure to capture and processes the EM emissions from a Raspberry Pi was followed. The attack procedure revealed that the entire secret key was still recoverable. This demonstrates that the EM leakage was not dependent on the Operating System. However, the leakage was more from the physical structure of the device.

6.5.2 Additional Tests

Further analysis was performed on the signal itself by applying a Chi-square test. This test has been used in hardware solutions to evaluate a countermeasure as discussed in Section 3.10. The EM signature of the iterations of the proposed countermeasure was compared to the original EM signature of the AES-128 cryptographic process and the results are depicted in Table 6.22.

Table 6.22: Results of the Chi–square test.

	Probability								
Countermeasure	**0.99**	**0.95**	**0.9**	**0.75**	**0.5**	**0.25**	**0.1**	**0.05**	**0.01**
FRIES									X
Prime Numbers					X				
Prime Numbers Visable		X							
Multi-Threading		X							
AES Plain	X								

Firstly the EM signature was compared to itself and the resulted indicated that there is a 99% probability that the signal is within the same signal as depicted by *AES Plain*. The first iterations of the proposed countermeasure as only multi-threads were utilised depicted that there was a 95% probability that the cryptographic signature is within that signal and the prime number generator when the signal was visible depicts the same results. Interestingly, when the AES-128 cryptographic thread was not visible the statistical test revealed that there was a 50% probability that the AES-128 cryptographic signal was within the noise generated. The final iteration of the FRIES noise generator which had the OpenSSL implementation revealed that there was only a 1% probability that the cryptographic EM signature was located within the noise.

The experiments included a final test where the system was configured to automatically locate a specific histogram based on the frequencies that were leaking information. The reference histogram was the EM signature of the AES-128 cryptographic execution. The program was set to locate this reference histogram within the FRIES noise generated. The results revealed that the automotive system could not locate this signature with the final iteration of the FRIES noise generator. Finally, an in depth discussion with regards to the proposed countermeasure is presented in Chapter 7.

6.6 Summary

This chapter focused on two main themes. Firstly the recovery of the AES-128 cryptographic keys from a Raspberry Pi via a SCA attack and secondly, the development of a multi-threading software based countermeasure to mitigate SCA attacks against the Raspberry Pi.

The Raspberry Pi 2 executing the AES-128 algorithm within the libcrytpo++ library was evaluated. The EM emissions were captured from the device while it executed the cryptographic algorithm. The raw captured EM emissions were processed through various trace alignment and denoising processes such as Elastic alignment and the SG filter. The resultant signal was sent to the CPA attack where the results indicated that the AES-128

encryption algorithm of the libcrypto++ library was vulnerable against the CPA attack as 12 of the 16 subkeys were recovered. Although the subkeys located at position 9 – 12 were not recovered, a brute force attack was applied to recover the remaining subkeys. The remaining subkeys were recovered via brute force with an average time of 28 seconds.

The AES-128 encryption algorithm was encapsulated into various multi-threading APIs. The multi-threaded APIs were *pThreads*; C11 threads; TBB; and OpenMP. The results indicated that there were variations in shape and the amount of EM leakage occurred as various multi-threaded APIs were used. It was demonstrated that the *pThread* API had the quickest runtime execution and had the least amount of EM leakage as opposed to the TBB implementation that took the longest and produced more EM leakage.

Additional experiments investigated what effects the gcc, g++, and clang compilers had on the EM signature of a cryptographic program. The AES-128 cryptographic program within the multi-threaded framework of *pThreads* was used as the base program. As the program was compiled, different optimization flags were used. The results indicated that using the optimization flag of *O3* changes the EM signature of the program. The assembler code was analysed and the g++ implementation displayed that fewer instructions were required which related to less EM emissions leaked. Although fewer instructions were used by the clang compiler there was more EM leakage which ties back to Table 3.3 as it was possible that the code would have a longer run time. The analysis showed that using the optimization flag of *O3* in the clang compiled had enabled the usage of double precision numbers which required more bits to be used, thus more power was used y the device resulting in a greater EM leakage.

The CPA attack revealed that the GNU compilers (gcc and g++) achieved a similar result with respect to the number of subkeys recovered. However, it was demonstrated that the subkey location changed as different optimizations were used. Summing up, the recovered subkeys, this research was able to retrieve 15 of the 16 AES-128 subkeys, with only subkey 12 not being recoverable. The fewest number of subkeys was recovered when the clang compiler was used with *O2* optimization. Although utilising higher levels of optimization, the results revealed that optimization *O3* for clang had the same effect as

using no optimizations.

A daemon noise generator known as the FRIES noise generator was introduced as a software based countermeasure. The results demonstrated that the FRIES noise generator had varied effects on the radio spectrum. In the first iteration of the FRIES noise generator, it was possible to visually detect the location of the execution of the AES-128 cryptographic algorithm. However, 64% (18/50) of the time it could not be visually detected. It was identified that the calculation of prime numbers was obfuscating the cryptographic algorithm. The analysis revealed as high prime numbers were calculated there was a window where the cryptographic algorithm could not be detected visibly.

To improve the obfuscation of the FRIES noise generator, the prime number sequence within the FRIES noise generator was replaced by the libcrytpo++ implementation of SHA-512 hash functions. The results indicated that the CPU was utilising a 100% of resources and the device started to lag and slow down. In this slow down state, the AES-128 encryption program could be visibly seen and extracted.

The FRIES noise generator was re-coded to utilise SHA-512 hash functions within the OpenSSL libraries. A stress test was performed where the noise generator would execute indefinitely and the results demonstrated that the CPU usage was at full load and all cores of the Raspberry Pi were being utilised. However, there was no lag from the device which resulted in more hashes being calculated per second and there was no clear indication of the AES-128 cryptographic execution.

Statistical analysis was performed on the FRIES noise generator, more importantly, the cross-correlation between the FRIES noise and a non protected AES-128 cryptographic implementation. The results revealed that there was no correlation between the two sets of EM signatures. Therefore, FRIES noise generator with the OpenSSL library was well suited to obfuscate EM SCA attacks.

Following the recovery of 12 subkeys in the initial experiments, this research aimed to uncover the remaining subkeys without utilising brute force. The filters were adjusted to omit frequencies ranging from 0 – 5 kHz, 5 – 10 kHz, 10 – 15 kHz. The results

demonstrated that 12 different subkeys were recovered as the frequencies from 0 – 5 kHz were omitted. As the frequencies between 5 – 10 kHz were omitted, 15 subkeys were recovered, with subkey 5 being unrecoverable. This followed by the omission of frequencies ranging from 10 – 15 kHz. In this instance, 15 subkeys were recovered. However, subkey 4 was recovered and subkey 9 could not be recovered. Since the removal of the frequencies between 5 – 10 and 10 – 15 kHz are overlapping each other with regards to the subkeys recovered. A final experiment was carried out which removed all frequencies between 0 - 15 kHz, i.e: only frequencies between 15 – 50 KHz were kept. Following this approach this research was successfully able to recover the entire secret key used for the AES-128 cryptographic algorithm with only utilising 50 input EM traces.

The research continues by carrying out further analysis as the 50 EM traces were segmented into 5 subsets of 10 traces each. Each subset was sent to the attack procedure where the CPA attack was applied and the results demonstrate that 4, 6, 9, 11, and 15 subkeys were recovered as 10, 20, 30, 40, and 50 traces were utilised as input data for the CPA attack respectively. This trend relates back to Chapter 5 as more input data was used, more subkeys were recovered of a microcontroller. The order of subsets B(10 –20 traces) and C(20 –30 traces) were rotated and the results revealed that different subkeys were being recovered. However, changing the order of subsets D and E, had no effect on the subkeys recovered. The analysis showed that the CPA utilises the early subset data as a base and it recovers the remaining subkeys easier.

Another statistical analysis was performed on the FRIES noise generator, more importantly, to determine the Chi-square test which included the FRIES noise and a non protected AES-128 cryptographic implementation. Firstly the EM signature was compared to itself and the resulted indicated that there was a 99% probability that the signal was within the same signal. The first iterations of the proposed countermeasure as only multi-threads were utilised depicted that there was a 95% probability that the cryptographic signature is within that signal and the prime number generator when the signal was visible depicted the same results. The final iteration of the FRIES noise generator which had the OpenSSL implementation revealed that there was only a 1% probability that the cryptographic EM signature was located within the noise.

The experiments and result in this chapter has successfully demonstrated that it was possible to recover the entire AES-128 secret key from the Raspberry Pi. A multi-threaded software based countermeasure to produce EM noise while the cryptographic implementation was executing was a viable and secure option for both microcontrollers and a Raspberry Pi. A final discussion with regards to the impact of this research is presented in Chapter 7.

The world is very different now. For man holds in his mortal hands the power to abolish all forms of human poverty, and all forms of human life.

John F. Kennedy

7

Discussion

THIS chapter reflects on the work conducted in this research, with particular focuses on the previous two chapters. The significance of this research is presented in Section 7.1 which is followed by a discussion in Section 7.2 on the impact this research has on current and future work. The chapter concludes in Section 7.3 with a summary of this research.

7.1 Significance

The primary focus of this research was the development of a software based countermeasure against Side Channel Analysis (SCA) that utilised multi-threading. A software solution was selected as it can be easily modified and rolled out to various devices, even with the possibility of remote updates. A hardware solution is tedious, as it requires users to return their devices or purchase a more expensive device, or on-site modifications. Hardware countermeasures can easily become obsolete whereas a software approach, can continuously be updated and easily maintained which can provide assurance to the end user that no additional cost would occur. This reduces the cost to manufactures as re-designing hardware is expensive.

Although, this research targeted microcontrollers and a Raspberry Pi, the scalability of this novel countermeasure can be applied to other devices that may require protection,

such as smartphones, embedded systems, Internet of Things (IoT) infrastructure and fully-featured platforms. The proposed countermeasure is not limited to an embedded device but can be applied to high end desktops and servers where critical information is produced and stored, thus requiring to be secured.

Attacks against cryptographic systems, especially the side channel leakage of cryptographic information have been widely researched for decades. This research has contributed to the protection of the cryptographic process more specifically the AES-128 cryptographic algorithm. However, this novel countermeasure is not limited to cryptography alone, as it can be applied to other operations such as a login process of a device.

Since this is a software based solution, developers can integrate this solution either within or alongside their own applications. Their application can have the ability to execute the noise generator at program startup or at critical segments of the algorithm which is depicted in Listing 7.1 where a snippet of the code illustrates executing the FRIES noise generator, followed by the cryptographic algorithm.

Listing 7.1: Scenario one of utilising the FRIES noise generator.

```
1 thread t1(FRIESNoise);
2 thread t2(doCrypto);
3
4 t1.join();
5 t2.join();
```

This feature is not limited to developers. However, the end user can utilise the noise generator as a standalone application in scenarios where the user activates the noise generator and proceeds by making phone calls, sending encrypted messages or even just browsing the Internet on their devices as depicted in Figure 7.1.

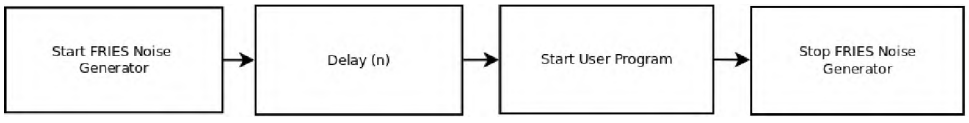


Figure 7.1: Scenario two of utilising the FRIES noise generator.

The diagram indicates a waiting period of two seconds, this is to cater for the initial startup of the FRIES noise generator as discussed in Section 6.4.2.

As mentioned in Section 6.4.1 the FRIES noise generator required additional resources to be utilised. This resulted in the utilisation of all the Central Processing Unit (CPU) cores on the Raspberry Pi. However, users have the option to execute the noise generator only when required for example utilising a banking application on a smartphone with the noise generator. In scenarios where individuals are less concerned about power consumption and battery drain, the noise generator can execute continuously by starting the FRIES noise program as one would execute a normal binary executable, thus adding noise to any operations carried out by the device. This is beneficial to protect military and government communications from being intercepted or interpreted by foreign entities.

The proposed countermeasure further scales into our interconnected lives as IoT has become prominent. These devices are situated in individuals homes, business and cities. Examples of these interconnected systems are smart grids (Yun and Yuxin, 2010), where these IoT devices automatically control the distribution of electricity in real time to ensure that electricity can be used effectively and efficiently; another example is the control and distribution of natural gas in pipelines (Bonomi *et al.*, 2012) by these devices. Our homes have many sensors that monitor regular domestic conditions (Kelly *et al.*, 2013), such as informing the individual when to take out the trash, purchasing groceries and the temperature (Alam *et al.*, 2012). Therefore, the countermeasure presented in this research takes into consideration IoT devices and can be easily implemented without the individual having to return for example his fridge, as it has become a security risk.

7.2 Impact

As mentioned in the previous section this research has introduced a software based solution to mitigate SCA attacks However, the countermeasure is not limited to cryptographic devices as it can expand to protect other devices from potential other attacks.

The vulnerabilities of IoT devices was demonstrated as they were used in a botnet (Dobbins, 2016) to carry out a Distributed Denial-of-Service Attack (DDoS) attacks on a French hosting provider OVH (Paganini, 2016). Additionally, a DDoS attack was carried

out against DynDNS (Button, 2016) which resulted in major sites and services including GitHub, Reddit, PayPal, Amazon, and thousands more to be inaccessible to billions of people worldwide.

Although, IoT devices are at risk from various attacks. These devices are not the only devices at risks. Programmable Logic Controllers (PLCs) are among other devices at risk (Wei *et al.*, 2010, Amin *et al.*, 2013). Interconnected PLCs, which allow the automation of electromechanical processes such as those used to control machinery on factory assembly lines, amusement rides, or centrifuges for separating nuclear material becomes a target and are potentially vulnerable if not correctly secured.

The impact of obtaining symmetric cryptographic or private keys could lead to a major catastrophe to industrial control hardware, federal systems and smart cities. Once keys are obtained, decrypting information becomes a trivial task. Having *open access* to these systems would allow attackers to take control of a nuclear silo base or a strategic military asset capable of mass destruction. This was further evident when the Iranian nuclear program had been sabotaged by stolen code signing certificates in the Stuxnet attack (Langner, 2011).

An additional example of a vast comprise was the cyber attack on a German steel-mill (Lee *et al.*, 2014). Once the adversaries gained access to the system, they proceeded by taking control of the production management software of the steel mill. Having this level of access, the adversaries had control of the plant's control systems where they methodically destroyed human machine interaction. This resulted in preventing a blast furnace from initiating its security settings in time and caused serious damage to the infrastructure.

Despite the attacks against the Iranian nuclear program (Langner, 2011) and the German steel-mill, the development of these cybernetic attacks can date back to 2007 where the United States, Department of Homeland Security demonstrated how a cyber attack could destroy physical components of the electric grid and leave the nation without power for weeks or months (Salmon *et al.*, 2009).

A few other scenarios would be that the adversary takes control over a medical insulin pumping system where it is possible to distribute incorrect amounts of insulin and produce

a slow and painful death to an individual (Radcliffe, 2012). Additionally, attackers have the ability to attack and take control of a motor vehicle (Ring, 2015, Nie *et al.*, 2017) by disrupting or intercept electromagnetic signals to allow for malicious actions against the electronic control unit of the vehicle which can lead to serious fatalities (Parkinson *et al.*, 2017).

Devices such as satellite TV and other set-top decoders are vulnerable as the attacker would be able to intercept and decrypt the signal using the stolen encryption keys (Macq and Quisquater, 1995). The decrypted signal can be used and distributed to others without a subscription fee, thus resulting in the industry losing revenue. Since the arrival of Digital Economy, the use of IoT, smartphones and other devices has expanded specifically to include tasks such as banking, social networking, e-commerce and bitcoin transactions on a regular basis, Therefore, it is fundamentally important to ensure that all devices are protected.

Cellphone calls would also be exposed and leaked to various third parties if the encryption keys were stolen. In 2015, the hack into Gemalto (Scahill and Begley, 2015) in which billions of mobile SIM cards cryptographic keys were stolen demonstrated the importance and impact of stolen keys. With these stolen encryption keys, intelligence agencies and third parties are able to monitor mobile communications without applying for approval from telecommunication companies or foreign governments. Possessing these keys bypasses the need to obtain a warrant or a wiretap while leaving zero evidence on the networks that communications were intercepted. Irrespective of the civil liberties, the main focus is that mobile communications have been compromised due to stolen keys.

What the adversaries have succeeded in doing with the German steel-mill, the Iranian nuclear program and other attacks must be taken seriously. If cyber attacks are now able to cause damage to infrastructure, then the mass population can be impacted as well. As an attack on an electricity production facility could cause energy supply disruptions for millions of people. In a similar fashion if attackers were to compromise a waste management facility or a hospital consequences could become dramatic, with the threat of contamination or poisoning.

These scenarios are but a few to mention as there are many other cases where stolen encryption keys can or were utilised for bad reasons. We are currently in an era where bullets and bombs are being replaced by bits and bytes to damage and destroy infrastructure with computer code as digital attacks are instant and can occur without warning. The National Security Agency (NSA) and other agencies have moved away from the traditional signal intelligence which consisted of passive interception to a more attack minded approach of infiltration and destroying a system. Recently Symnatec has published a report detailing that the power grids around the world have been infiltrated by adversaries to firstly steel critical information such as technical diagrams, reports, passwords, crypto keys and secondly causing mass destruction by destroying infrastructure via code. Therefore, it is becoming extremely important and critical that all infrastructure be protected

By applying what is learned in this research, developers and end users has the ability to safeguard applications, information, and equipment from sponsored state attacks to ordinary attacks from individuals. The proposed software based countermeasure makes it easier to update the software in the event of a possible breach within the countermeasure, as opposed to a hardware solution, where the entire device has to be recalled or replaced. It may not be a perfect solution. However, this research adds additional layers of security that can be applied to protecting critical infrastructure.

This document is concluded in the next chapter by providing a summary of this research objective and outcomes.

7.3 Summary

This chapter reflects on the work conducted in this research by reviewing the impact, importance, and significance of this research.

In chapter 5 the research introduced a novel countermeasure by utilising multi-threads as software countermeasure to mitigate SCA attacks on microcontrollers. The results demonstrate that the proposed countermeasure obfuscated and prevented full key recovery from

a Correlation Power Analysis (CPA) and template attack. Moreover, it withstood additional signal processing technique such as noise reduction and trace alignment techniques whereas existing countermeasures such as hiding and shuffling techniques fail against these additional digital filtering techniques.

Although the proposed countermeasure was successful in mitigating SCA attacks, modifications had to be introduced to the existing AES-128 algorithm. One of the objectives of this research was to develop a countermeasure that does not modify the existing cryptographic algorithm. However, these experiments and results demonstrated that the possibility of a software based countermeasure utilising multi-threads was possible. Once this was established the research aimed to design a software based countermeasure that did not modify the existing cryptographic algorithm and can execute alongside it.

A software based countermeasure to execute alongside existing cryptographic algorithms was introduced in Chapter 6. The countermeasure is known as the FRIES noise generator. The tool was utilised as a system daemon, allowing the user to execute the noise generator in the background. The final iteration of the FRIES noise generator was implemented using the OpenSSL libraries. The stress test demonstrated that the CPU usage was 100% and all cores were being utilised. However, there was no lag from the device. While these hash function were producing EM noise, the cryptographic implementation of AES-128 algorithm could not be visibly seen thus a basic SCA was mitigated. The strength of the countermeasure was further demonstrated as it withstood statistical attacks such as the CPA attack and added trace alignment techniques.

The proposed countermeasure is easy to implement and be designed on cross-platform devices as this was demonstrated by easily recompiling the source code on a different Operating Systems and illustrated that the FRIES noise generator was still able to protect against a SCA attack.

A final failure test was conducted to determine the point of failure for the proposed countermeasure. It was observed when executing five instances of the proposed countermeasure in parallel, the Raspberry Pi started to fail under load and EM information was leaking

out. However, for large scale systems, the user has the ability to spawn many instances of the proposed countermeasure.

Defending against known and future attacks are much more important than the ability to launch attacks. When it comes down to the world's technological economy, civilization has more to lose as attacks are more prominent, especially as devices contain vulnerabilities to many forms of cyber and SCA attacks. Therefore, we should be developing and focusing on more defensive applications as it would allow for a more trusted and safe environment.

Chapter 8 brings a conclusion to this research by discussing the objectives, techniques carried out to answer the research question, contribution made by this research and finally, directions for future work.

Acquire knowledge and teach it to people.

Umar bin Al-Khattab

8

Conclusion

SEVERAL significant contributions have been made by this research to the field of software countermeasures against Side Channel Analysis (SCA) attacks, particularly with regard to embedded and high frequency devices. This chapter concludes this research with a summary of the research conducted in Section 8.1. Followed by Section 8.2 which elaborates on the research objectives and the novel achievements by this research. The chapter is concluded with direction for future work in Section 8.3.

The main objective of this thesis was to determine whether a software approach utilising multi–threads could potential serve as a countermeasure to defend against SCA attacks. This research investigated various known and existing techniques to mitigate SCA attacks. These techniques were implemented on ATmega microcontrollers and a Raspberry Pi. The results in this research indicated that the proposed countermeasure outperformed existing countermeasure with regards to the secret information recovered.

Additionally, a novel attack methodology was developed where the entire secret AES-128 encryption key was recovered from a Raspberry Pi, which has not been achieved before. The FRIES noise generator was pitted against this new attack vector and other known noise generators. The results exhibited that the FRIES noise generator withstood this attack whilst other existing techniques still leaked out secret information. The visual location of the AES-128 encryption algorithm in the EM spectrum and key recovery was

prevented. The results further demonstrated that the proposed multi-threading software based countermeasure was able to be resistant to existing and new forms of attacks, thus verifying that a multi-threading software based countermeasure can serve to mitigate SCA attacks.

8.1 Chapter Summaries

This thesis commences by providing the reader with a brief background to the problem and the impact that stolen or comprised encryption keys can lead to in Chapter 1. This is followed by the research question and the objectives presented in Chapter 1 with the hope of answering the proposed research questions, more specifically, can a software approach utilising multi-threads serve as a countermeasure against SCA attacks for embedded and a fully featured device?

Chapter two discusses a brief history of cryptography leading up to the modern age of cryptography. The process of cryptography consists of two phases known as encryption and decryption. Encryption transforms *plaintext* (information) into *ciphertext* (meaningless information). To perform encryption or decryption a cipher or cryptographic algorithm is used. The cipher is used in tandem with a secret key which is only known by the sender and receiver. Without the secret key, a third party would be unable to decrypt the information.

Various types of attacks on cryptography are mentioned in Section 2.2.5, followed by the explanation of the AES-128 block cipher that will be a key aspect in this research in Section 2.3. The topic of hash function was discussed in Section 2.4 as it will be utilised to develop a software based countermeasure.

The thesis continues to **chapter three** where an introduction to SCA is presented, including a brief history of SCA and Electromagnetic (EM) attacks against embedded and high frequency devices were presented in Section 3.1. This chapter serves as a bridge to the previous chapter where the design of the AES-128 cryptographic algorithm is used to

locate points of attack and vulnerable areas in the algorithm with regards to the information leaked by the hardware which results in secret information recovered. This is followed by the basics of quantifying device consumption, a brief overview of microcontrollers and how the architecture is linked to the device consumption in Section 3.2. The chapter continues by discussing techniques that can be utilised to recover secret information based on the device consumption such as simple and more advanced statistical analysis from Sections 3.3 – 3.6.

The procedure to implement a SCA attack was discussed in Section 3.8, this included data collection and preparation, good measurement setup, signal processing techniques to enhance the signal and finally, a method to evaluate the success ratio of a SCA attack. SCA countermeasures ranging from hardware, software, and EM based countermeasures, more specifically the hiding and shuffling countermeasure in the time domain was focused on in Section 3.9. The chapter was concluded by detailing potential software that is not commonly utilised as a software based countermeasure but can potentially serve as a countermeasure in Sections 3.11 – 3.12.

The methodology of this research is presented in **chapter four** as digital filtering techniques to improve and assist in the recovery of secret information from both the microcontrollers and the Raspberry Pi was discussed in Section 4.1. Additionally, techniques to increase the signal-to-noise ration such as the Savitzky–Golay filter was discussed and trace alignment techniques such as the Elastic Alignment techniques were detailed in Section 4.2.

The work continues by elaborating how the techniques in Sections 4.1 and 4.2 can be utilised in cohesion to reduce the computational time and input data required to predict the correct secret key in Section 4.3. The hardware and software confrontation setup utilised to captured power and electromagnetic (EM) consumption from both microcontrollers and the Raspberry Pi is discussed in Sections 4.4.

The experiments and results were segmented into two phases. The first phase focused on experiments with smartcards and microcontrollers which was discussed in **chapter five**. The basics of a SCA attacks against smartcards was demonstrated in Section 5.1. The

results indicated that more bus lines are required to switch from a precharge state to a high state as the power signature changed based on the input data. This information was utilised to investigate the effects of using two different compilers to compile code for the smartcard. The results demonstrate that utilising a different compiler and/or smartcard resulted in different power and EM emissions leakage in Section 5.1.2. The EM emissions were captured from the smartcard as the AES-128 encryption algorithm was executed. This was followed by digital signal processing techniques to increase the signal-to-noise ratio and recover the secret key used by the AES-128 encryption algorithm from a smartcard as discussed in Section 5.1.3.

Although this research was able to perform a successful SCA attack on a smartcard, the implementation of a multi-threading software based countermeasure could not be implemented as the smartcard had no support for multi-threading. Therefore, this research pivoted from smartcards to a pure microcontroller where there is more flexibility in terms of implementing a multi-threading software based countermeasure. The two Atmel microcontrollers targeted were the ATmega328p and ATxmega128D4

The research introduced a novel countermeasure by utilising multi-threads as software countermeasure to mitigate SCA attacks on microcontrollers in Section 5.2.1. The proposed countermeasure was implemented in the *VerifyPin* process in Section 5.2.2. This process does a verification check on a password entered into the device. The countermeasure prevented the adversary from obtaining the password from the *VerifyPin* process. The SCA resistance was increased by including each character check into a thread.

The research utilised statistical analysis attacks such as the Correlation Power Analysis (CPA) attack in Section 5.2.3 to recover the secret key of the AES-128 cryptographic algorithm. The results indicated that this research was able to replicate existing SCA attacks and perform a successful CPA attack based on power consumption of two different microcontrollers. The impact of a CPA attack was demonstrated as it required only 50 power traces for both microcontrollers to recover the full secret AES-128 encryption key in Section 5.2.3. The results demonstrate that the proposed countermeasure obfuscated and prevented full key recovery from a CPA attack. The power leakage model of the

CPA relies on the power traces to have minimal changes as each subkey is attacked. The proposed countermeasure produced dynamic power traces on each execution, causing confusion to the CPA attack model and reducing the correlation accuracy for predicting a correct subkey.

The research changed from capturing data from the power lines of a device and focused on capturing the EM emissions produced by the device in Section 5.2.4. Existing countermeasures such as hiding, shuffling, and random delays were added to the AES-128 implementation and compared to proposed countermeasure. With regards to the attack procedure, the template attack was added to recover secret information. The research has demonstrated the ability to retrieve the secret key on an unprotected AES-128 implementation as well a protected implementation with the hiding, shuffling, and random delays countermeasure by utilising EM emission. The full key recovery from the proposed countermeasure was unsuccessful. It was demonstrated that the proposed countermeasure's unique design allowed the algorithm to resist the CPA and template attack. It was further resistant against additional signal enhancement techniques.

Although, the AES-128 encryption algorithm was a major aspect of this research. Additional experiments were performed to evaluate the proposed countermeasure for microcontrollers. An unprotected AES-256 implementation was implemented on the microcontrollers and successful key recovery was achieved by utilising a CPA attack was demonstrated in Section 5.3.1. The proposed countermeasure was implemented within the AES-256 implementation and has prevented an attack on the 13^{th} and 14^{th} round of the AES-256 implementation to prevent the secret key being recovered via an SCA attack. The research was able to recover the first round key of the DES implementation using the SCA attacks in Section 5.3.2. Upon acquiring the key, it was used to recover the full encryption key. The proposed countermeasure was tweaked to match the DES leakage and prevented the SCA attack against the implementation.

Chapter six discusses the process as this research shifted the focus from microcontrollers to a high frequency device such as the Raspberry Pi. The AES-128 encryption algorithm of the libcrypto++ library was shown to be vulnerable to the CPA attack as 12 of the

16 subkeys were recovered in Section 6.2. The result indicated that subkeys located at position 10 – 13 were not recovered. The remaining subkeys were brute forces with an average time of 28 seconds. Secondly, the area above and around the CPU of the Raspberry Pi leaks out critical and secret information as opposed to the voltage regulator. Finally, existing attacks and digital processing technique were used to increase the success ratio and retrieve more subkeys whereas utilising no processing recovered fewer subkeys.

The research conducted additional experiments by investigating the effects the gcc, g++, and clang compilers had on the EM signature of a cryptographic program in Section 6.3. The AES-128 cryptographic program within the multi-threaded framework of *pThreads* was used as the base program. As the program was compiled, different optimization flags were used. The results indicated that using the optimization flag of *O3* changes the EM signature of the program. The assembler code was analysed and the g++ implementation displayed that fewer instructions were required which related to less EM emissions leaked. Although fewer instructions were used by the clang compiler the EM trace had more data points. The analysis demonstrated that using the optimization flag of *O3* in the clang compiled had enabled the usage of double precision numbers which required more bits to be used and more power to be used.

The CPA attack revealed that the GNU compilers achieved a similar result with respect to the number of subkeys recovered. However, it was demonstrated that the subkey location changed as different optimizations were used. Summing up, the recovered subkeys, 15 of the 16 AES-128 subkeys was recovered, with only subkey 13 not being recoverable. Furthermore, the fewest number of subkeys was recovered as the clang compiler was used with *O2* optimization. Although utilising higher levels of optimization, the results revealed that *O3* optimization for clang had the same effect as using no optimizations.

The EM leakage of the Raspberry Pi as the device executing the AES-128 algorithm of the libcrypto++ library in a threaded environment was investigated in Section 6.4 in the hope of developing a multi-threading software countermeasure. The multi-threaded libraries used, were the *pThreads*; C11 threads; TBB; and OpenMP. The results indicate that different patterns in the EM field occur as various threaded APIs were used. The

various multi-thread technique has variations in frequency and shape. The Raspberry Pi's power management system used different cores and resources on each execution. The *pThread* thread implementation has the fastest runtime execution and has fewer data points, with the TBB implementation taking the longest. However, producing more spikes in the frequency.

A daemon noise generator known as the FRIES noise generator was introduced in Section 6.4.1. The results demonstrated that the FRIES noise had different effects on the radio spectrum. Firstly, it was still possible to visually detect the location of the execution of the cryptographic algorithm. However, out of 50 occasions, the cryptographic execution was not detected visually 32 times. It was identified that the calculation of prime numbers would hide the cryptographic algorithm. The analysis revealed as high prime numbers were calculated there was a window where the cryptographic algorithm could not be seen visibly. The prime number sequence within the FRIES noise generator was replaced by the libcrytpo++ implementation of the Secure Hash Algorithm (SHA) in Section 6.4.3. The results indicated that the CPU was utilising a 100% of resources and the device started to lag and slow down. In this slow down state, the AES-128 encryption program could be visibly seen and extracted.

The FRIES noise generator was re-coded to utilise the OpenSSL libraries. The stress test demonstrated that the CPU usage was 100% and all cores were being utilised. However, there was no lag from the device. More hashes were being calculated per second as opposed to libcrypto++ implementation. While the hash function was introducing EM noise, the cryptographic implementation of the AES-128 algorithm could not be visibly seen.

Statistical analysis was performed in Section 6.4.4 on the FRIES noise generator, more importantly, the cross-correlation between the FRIES noise and a non protected AES-128 cryptographic implementation. The results revealed that there was no correlation between the two sets of EM signatures. Therefore, the OpenSSL library is well suited to obfuscate EM SCA attacks.

Additional analysis was performed in Section 6.5 which removed all frequencies between 0 –

15 kHz within the signal data from Section 6.2, i.e: only frequencies between 15 – 50 KHz was kept. Following this approach this research was successfully able to recover the entire secret key used for the AES-128 cryptographic algorithm. Therefore, this research has demonstrated the first successful AES-128 key recovery from a Raspberry Pi.

Once the entire secret key was recovered, the proposed countermeasure was evaluated again. The Chi-square test was used to determine if the EM signature of the AES-128 cryptographic algorithm could be detected even though the frequencies were removed. The results displayed that the AES-128 signature could not be detected. This further emphasizes the strength of the proposed countermeasure.

Although, key recovery was possible the experiments and result in **chapters five and six** had successfully demonstrated that a multi-threaded software countermeasure to produce noise while cryptographic implementation is executing was a viable and secure option for both microcontrollers and a fully featured device such as a Raspberry Pi.

Chapter seven reflects on the work conducted in this research, with particular focuses on **chapters five and six**. Section 7.1 discusses the significance of this research which is followed by Section 7.2 which details the impact this research has on current and future work. The chapter is concluded in Section 7.3 with a summary of this research.

8.2 Research Goals and Achievements

The research in this work is the first of its kind as it investigated the utilisation of multi-threading as a software based countermeasure to mitigate SCA attacks. Multi-threading is mainly used to increase performance and runtime execution of an algorithm or program. This investigation is novel, as there has not been a software based countermeasure relying on multi-threading to mitigate SCA attacks. This research has been tested on Atmel microcontrollers which are commonly used and a more fully featured system in the form of the popular Raspberry Pi.

Reviewing the research questions posed in Section 1.3 the overall theme of this research was to answer the following question:

"*Can a software approach utilising multi-threads serve as a countermeasure against SCA attacks for embedded and a fully featured device?*"

This research can comfortable state **yes** a multi-threading software based countermeasure to mitigate SCA attacks against microcontrollers and a Raspberry Pi is a viable approach. The experiments and results in Chapters 5 and 6 have demonstrated that the proposed countermeasure obfuscates the current "best of breed" detection of the AES-128 algorithm in the EM domain and prevents the full recovery of the AES-128 encryption key.

Section 5.2.4 demonstrates that the proposed software countermeasure implemented on microcontrollers outperforms existing software based countermeasures such as random time delays, shuffling of operations and the insertion of dummy operations. It is further demonstrated that the proposed countermeasure was resistant to digital filtering techniques such as the Elastic Alignment and the Savitzky–Golay filter that aids in the recovery of secret information. Finally, it was demonstrated that the proposed countermeasure withstood the CPA and template attack. These findings are important as it demonstrates that the proposed countermeasure can outperform existing software based countermeasures, as this was another question put forward by this research.

The proposed countermeasure was incorporated alongside other cryptographic implementations such as the AES-256 and the Data Encryption Standard (DES) algorithms. The results in Section 5.3 demonstrated that the proposed countermeasure was able to obfuscate and mitigate key recovery from these algorithms. These results indicate the scalability of the proposed countermeasure as it can be implemented with other cryptographic algorithms, with minimal changes to the proposed countermeasure.

The scalability of the proposed countermeasure was further demonstrated in Chapter 6 as it was implemented on a Raspberry Pi 2. The design of the proposed countermeasure for microcontrollers was integrated within the cryptographic algorithm. However, the experiments in Section 6.4 moved away from tampering with the design of the cryptographic

algorithm and developed a standalone countermeasure that can be enabled at any time. The rationale behind this was to develop a software based countermeasure that could work with other applications and not only cryptographic applications. As of Section 6.4.1, the proposed countermeasure was known as the FRIES noise generator.

A novel attack methodology was developed in Section 6.5 where the entire secret AES-128 encryption key was recovered from a Raspberry Pi. The FRIES noise generator was pitted against this new attack mechanism and the results exhibited that the FRIES noise generator withstood this attack. The visual location of the AES-128 encryption algorithm in the EM spectrum and key recovery was prevented. These results displayed that the proposed multi-threading software based countermeasure was able to be resistant to new forms of attacks, thus answering the final question posed by this research in Section 1.3

8.2.1 Contributions

This research has made many contributions to the research field and a list of publications can be found in Appendix E. Moreover, the major contributions made by this research are as follows:

1. Implemented and demonstrated that a multi-threading software based countermeasure works on a microcontroller by preventing the Correlation Power Analysis (CPA) and template attacks in Chapter 5.

2. Demonstrated that the proposed countermeasure can withstand additional digital filtering techniques which enhance SCA attacks in Section 5.2.4.

3. Section 6.2 demonstrated that the AES-128 cryptographic implementation of the libcrypto++ library was vulnerable to SCA attacks.

4. Determined in Section 6.3 that various compiler optimizations used to compile the cryptographic binary had different effects on the EM spectrum as the binary was executed.

5. Section 6.3.1 illustrated that compiler optimization resulted in the recovery of various subkeys from the AES-128 secret key.

6. Investigated in Section 6.4 the usage of wrapping the cryptographic algorithm within four different multi-threaded libraries.

7. The first to investigate the usage of hash functions to generate EM noise to mitigate SCA attacks against a Raspberry Pi in Section 6.4.3.

8. Section 6.4.3 further exhibited that the OpenSSL implementation of the Secure Hash Algorithm (SHA) outperformed the libcrypto++ implementation of the SHAs in terms of obfuscating secret information.

9. The first of its kind to utilise SCA attacks against a Raspberry Pi and recover the full AES-128 encryption key in Section 6.5.

The most important and novel of these contributions is the introduction of a multi-threading software based countermeasure to mitigate SCA attacks on an embedded device and a Raspberry Pi in Chapters 5 and 6. These multi-threads are comprised of various mathematical operations to generate EM noise which obfuscates the execution of the AES-128 algorithm.

With regards to the microcontroller, this approach introduces dynamic power traces, where the power traces are changed based on the task schedulers conditions. Additionally, it was demonstrated that this novel countermeasure outperforms existing countermeasures of hiding and shuffling against the CPA and template attacks. The multi-threaded countermeasure further demonstrates that it's able to mitigate additional signal processing techniques such as the Elastic Alignment and various digital filtering in Chapter 5.

The novelty of this research is further demonstrated by utilising the EM emissions captured off a Raspberry Pi and applying SCA attacks to recover the full secret key from the AES-128 cryptographic implementation within the libcrypto++ library. It was further demonstrated that this research was able to design a novel countermeasure to prevent this new attack in Chapter 6 in a non-sterile environment as depicted in Appendix A, i.e:

no Faraday cage or special equipment was utilised to block out unwanted EM emissions from other sources.

This research adds an additional contribution by investigating the effects the GNU and clang compilers with different optimization levels had on the cryptographic binary at runtime in Section 6.3. It was demonstrated that each optimization produced various EM signatures. The EM data from the various compilers under different optimization levels was used as input in a CPA attack and partial AES-128 encryption keys were recovered. The CPA attack revealed that the GNU compilers achieved a similar result with respect to the number of subkeys recovered. However, the location of the subkey recovered had varied. Additionally, the least subkeys were recovered while utilising the clang compiler with *O2* optimization. Although utilising more optimization, the results revealed that level *O3* optimization for clang had the same effect as utilising no optimizations.

The usage of wrapping the cryptographic algorithm within four different multi-threaded libraries as opposed to tampering with the cryptographic algorithm was introduced in Section 6.4. These multi-threading APIs are shown to have variations in frequency and shape within the EM spectrum. Additionally, this leakage was used to locate the cryptographic process and recover the secret keys of the AES-128 algorithm. This research adds a further contribution by demonstrating that the CPU of the Raspberry Pi leaks out sensitive information as opposed to the voltage regulator.

Finally, this research implemented a novel EM noise generator known as FRIES in Section 6.4.1 to assist in obfuscating data captured in the EM field. The noise generator comprises of hiding the AES-128 algorithm within the EM noise generated by SHA from the libcrypto++ and OpenSSL library. It was demonstrated that the OpenSSL implementation outperforms the libcrypto++ implantation with regards to the device performance and the EM leakage. While the hash function was generating EM noise, the cryptographic implementation of the AES-128 algorithm could not be visibly seen. Furthermore, the intentional leakage was captured and the results indicated that no secret information was recovered.

8.3 Direction for Future Work

This research has successfully demonstrated that the concept of utilising multi-threading as software based countermeasure to mitigate SCA attacks on microcontrollers and a Raspberry Pi is possible and can be utilised with future products and devices as discussed in Section 7. Nonetheless, there is still provision for this research to expand in the current research domain and into various other domains.

A cost effective analysis on the power draw as the FRIES noise generator is implemented on various other devices could be investigated. This would aid users in the understanding of the power draw, especially those who value battery life of their device, as the proposed countermeasure would be utilised alongside other applications such as banking applications. Based on the cost effective analysis, other hashes and more mathematically operations could potentially be incorporated in the FRIES noise generator.

To implement and extend the proposed countermeasure into an easy and readily available application that can be utilised in various device such as the Orange Pi and other smart devices that are incorporated and will be apart of our interconnected society.

This research has served as a very remarkable experience to the researcher. It is hoped that this research will benefit others in the field towards developing a software countermeasure library that is able to mitigate known and future SCA attacks across various devices. Therefore, the datasets have been provided as detailed in Appendix F.

References

Aboulkassimi, D., Agoyan, M., Freund, L., Fournier, J., Robisson, B., and Tria, A. Electromagnetic analysis (EMA) of software AES on Java mobile phones. In *2011 IEEE International Workshop on Information Forensics and Security*, pages 1–6. Nov 2011. ISSN 2157-4766. doi:10.1109/WIFS.2011.6123131.

Agrawal, D., Archambeault, B., Rao, J. R., and Rohatgi, P. The EM Side – Channel(s), pages 29–45. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-36400-9. doi:10.1007/3-540-36400-5_4.

Akhter, S. and Roberts, J. Multi-core programming: Increasing Performance Through Software Multi-threading. Intel Press, 1st edition, 2006. ISBN 978-0976483243.

Alam, M. R., Reaz, M. B. I., and Ali, M. A. M. A review of smart homes; past, present, and future. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42:1190–1203, Nov 2012. ISSN 1094-6977. doi: 10.1109/TSMCC.2012.2189204.

Allen, P. E. and Holberg, D. R. CMOS analog circuit design. The Oxford Series in Electrical and Computer Engineering. Oxford Univiversity Press, 3rd edition, 2011. ISBN 978-0199765072.

Amin, S., Litrico, X., Sastry, S., and Bayen, A. M. Cyber security of water SCADA systems; Part I: Analysis and experimentation of stealthy deception attacks. *IEEE Transactions on Control Systems Technology*, 21(5):1963–1970, Sept 2013. ISSN 1063-6536. doi:10.1109/TCST.2012.2211873.

Anderson, R. J. and Kuhn, M. G. Low cost attacks on tamper resistant devices. In *Proceedings of the 5th International Workshop on Security Protocols*, pages 125–136. Springer-Verlag, London, UK, UK, 1998. ISBN 978-3-540-64040-0. doi:10.1007/BFb0028165.

Antoniou, A. Digital signal processing: Signals, Systems, and Filters. McGraw-Hill Education, 1st edition, 2016. ISBN 978-0071454247.

Atmel. X MEGA D manual. 2014. Accessed: 2014-08-02.
URL http://www.atmel.com/images/atmel-8210-8-and-16-bit-avr-microcontrollers-xmega-d_manual.pdf

Atmel. ATmega328P. 2015. Accessed: 2014-08-02.
URL http://www.atmel.com/images/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf

Aziz, S. M. and Pham, D. M. Threat implications of the Internet of Things. In *2013 5th International Conference on Cyber Conflict (CYCON 2013)*, pages 1–12. June 2013. ISBN 978-1-4799-0450-1.

Balakrishnan, N. Continuous multivariate distributions. Wiley Online Library, 2nd edition, 2000. ISBN 978-0471183877.

Balasch, J., Gierlichs, B., Reparaz, O., and Verbauwhede, I. DPA, bitslicing and masking at 1 GHz. In *Cryptographic Hardware and Embedded Systems - CHES 2015 - 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 599–619. 2015. doi:10.1007/978-3-662-48324-4_30.

Bar-El, H., Choukri, H., Naccache, D., Tunstall, M., and Whelan, C. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94(2):370–382, Feb 2006. ISSN 0018-9219. doi:10.1109/JPROC.2005.862424.

Barenghi, A., Pelosi, G., and Teglia, Y. Improving first order differential power attacks through digital signal processing. In *Proceedings of the 3rd International Conference on Security of Information and Networks*, SIN '10, pages 124–133. ACM, New York, NY, USA, 2010. ISBN 978-1-4503-0234-0. doi:10.1145/1854099.1854126.

Barney, B. POSIX threads programming. 2017. Accessed: 2017-03-01.
URL https://computing.llnl.gov/tutorials/pthreads/

Batronix. Antennas and probes. 2017. Accessed: 2017-08-01.
URL http://www.batronix.com/shop/accessory/aaronia.html

Beasley, J. S. and Miller, G. M. Modern Electronic Communication. Prentice Hall, 9th edition, 2007. ISBN 978-0132251136.

Belgarric, P., Fouque, P.-A., Macario-Rat, G., and Tibouchi, M. Side-channel analysis of Weierstrass and Koblitz curve ECDSA on Android smartphones. In *Proceedings of the RSA Conference on Topics in Cryptology – CT-RSA 2016 – Volume 9610*, pages 236–252. Springer-Verlag New York, Inc., New York, NY, USA, 2016. ISBN 978-3-319-29484-1. doi:10.1007/978-3-319-29485-8_14.

Bernstein, D. J. Cache-timing attacks on AES. 2005. Accessed: 2017-06-10. URL https://cr.yp.to/antiforgery/cachetiming-20050414.pdf

Bertino, E. and Islam, N. Botnets and Internet of Things security. *Computer*, 50(2):76–79, Feb 2017. ISSN 0018-9162. doi:10.1109/MC.2017.62.

Biham, E., Chen, R., Joux, A., Carribault, P., Lemuet, C., and Jalby, W. Collisions of SHA–0 and reduced SHA–1. In *Proceedings of the 24th Annual International Conference on Theory and Applications of Cryptographic Techniques*, EUROCRYPT'05, pages 36–57. Springer-Verlag, Berlin, Heidelberg, 2005. ISBN 978-3-540-25910-7. doi:10.1007/11426639_3.

Biham, E. and Shamir, A. Differential cryptanalysis of the Data Encryption Standard. Springer-Verlag, 1st edition, 1993. ISBN 978-1-4613-9314-6.

Bogdanov, A. Improved side-channel collision attacks on AES. In *Selected Areas in Cryptography: 14th International Workshop, SAC 2007, Ottawa, Canada, August 16-17, 2007, Revised Selected Papers*, pages 84–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-77360-3. doi:10.1007/978-3-540-77360-3_6.

Bogdanov, A. Multiple-differential side-channel collision attacks on AES. In *Proceeding Sof the 10th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '08, pages 30–44. Springer-Verlag, Berlin, Heidelberg, 2008. ISBN 978-3-540-85052-6. doi:10.1007/978-3-540-85053-3_3.

Bonneau, J. and Mironov, I. Cache-collision timing attacks against AES. In *Proceedings of the 8th International Conference on Cryptographic Hardware and Embedded Systems*, CHES'06, pages 201–215. Springer-Verlag, Berlin, Heidelberg, 2006. ISBN 978-3-540-46559-1. doi:10.1007/11894063_16.

Bonomi, F., Milito, R., Natarajan, P., and Zhu, J. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1519-7. doi:10.1145/2342509.2342513.

Bouder, H. L., Barry, T., Couroussé, D., Lanet, J., and Lashermes, R. A template attack against VERIFY PIN algorithms. In *Proceedings of the 13th International Joint Conference on e-Business and Telecommunications (ICETE 2016) - Volume 4: SECRYPT, Lisbon, Portugal, July 26-28, 2016.*, pages 231–238. 2016. ISBN 978-989-758-196-0. doi:10.5220/0005955102310238.

Brier, E., Clavier, C., and Olivier, F. Correlation power analysis with a leakage model. In *Cryptographic Hardware and Embedded Systems - CHES 2004: 6th International Workshop Cambridge, MA, USA, August 11-13, 2004. Proceedings*, pages 16–29. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. ISBN 978-3-540-28632-5. doi:10.1007/978-3-540-28632-5_2.

Brouchier, J., Kean, T., Marsh, C., and Naccache, D. Temperature attacks. *IEEE Security & Privacy*, 7(2):79–82, 2009. ISSN 1540-7993. doi:10.1109/MSP.2009.54.

Bruccoleri, F., Klumperink, E. A. M., and Nauta, B. Wideband Low Noise Amplifiers Exploiting Thermal Noise Cancellation. Springer US, 1st edition, 2005. ISBN 978-1-4020-3187-8.

Bruen, A. and Forcinito, M. A. Cryptography, information theory, and error-correction: A handbook for the 21st century. Wiley-Interscience, 1st edition, 2005. ISBN 978-0-4716-5317-2.

Button, R. Dyn DynDNS DDoS attack. 2016. Accessed: 2017-05-01.
URL http://www.red-button.net/blog/dyn-dyndns-ddos-attack/

Carlier, V., Chabanne, H., Dottax, E., and Pelletier, H. Electromagnetic side channels of an FPGA implementation of AES. In *International Association for Cryptologic Research (IACR) Eprint archive*. 2004.
URL http://eprint.iacr.org/2004/145

Carlier, V., Chabanne, H., Dottax, E., and Pelletier, H. Generalizing square attack using side-channels of an AES implementation on an FPGA. In *International Conference on Field Programmable Logic and Applications, 2005.*, pages 433–437. Aug 2005. ISSN 1946-147X. doi:10.1109/FPL.2005.1515760. Accessed: 2017-04-10.

Carter, J. L. and Wegman, M. N. Universal classes of hash functions. In *Proceedings of the Ninth Annual ACM Symposium on Theory of Computing*, STOC '77, pages 106–112. ACM, New York, NY, USA, 1977. ISBN 978-3-662-53008-5. doi:10.1145/800105.803400.

Chari, S., Rao, J. R., and Rohatgi, P. Template attacks. In *Cryptographic Hardware and Embedded Systems - CHES 2002: 4th International Workshop Redwood Shores, CA, USA, August 13–15, 2002 Revised Papers*, pages 13–28. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-36400-9. doi:10.1007/3-540-36400-5_3.

Chen, J., Moon, J., and Bazargan, K. Reconfigurable readback-signal generator based on a Field-Programmable Gate Array. *IEEE Transactions on Magnetics*, 40(3):1744–1750, May 2004. ISSN 0018-9464. doi:10.1109/TMAG.2004.826913.

Chen, Y. Challenges and opportunities of internet of things. In *17th Asia and South Pacific Design Automation Conference*, pages 383–388. Jan 2012. ISSN 2153-6961. doi:10.1109/ASPDAC.2012.6164978.

Chen, Z. Java Card Technology for Smart Cards: Architecture and Programmer's Guide. Addison-Wesley Longman Publishing, 1st edition, 2000. ISBN 978-0201703290.

Churchhouse, R. F. Codes and ciphers : Julius Caesar, the Enigma, and the Internet. Cambridge University Press, 1st edition, 2002. ISBN 978-0521008907.

Cimpanu, C. About 90% of smart tvs vulnerable to remote hacking via rogue tv signals. 2017. Accessed: 2017-07-01.
URL https://www.bleepingcomputer.com/news/security/about-90-percent-of-smart-tvs-vulnerable-to-remote-hacking-via-rogue-tv-signals/

Clavier, C., Coron, J.-S., and Dabbous, N. Differential power analysis in the presence of hardware countermeasures. In *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '00, pages 252–263. Springer-Verlag, London, UK, 2000. ISBN 978-3-540-41455-1. doi:10.1007/3-540-44499-8_20.

Cohen, L. Time-frequency analysis. Prentice Hall PTR Englewood Cliffs, 1st edition, 1995. ISBN 978-0135945322.

Collins, C. M. Fundamentals of signal-to-noise ratio (snr). In *Electromagnetics in Magnetic Resonance Imaging*, 2053-2571, pages 2–1 to 2–9. Morgan & Claypool Publishers, 2016. ISBN 978-1-6817-4083-6. doi:10.1088/978-1-6817-4083-6ch2.

Cooley, J. W. and Tukey, J. W. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965. ISSN 00255718, 10886842. doi:10.2307/2003354.

Coppersmith, D. The data encryption standard (DES) and its strength against attacks. *IBM J. Res. Dev.*, 38(3):243–250, May 1994. ISSN 0018-8646. doi:10.1147/rd.383.0243.

Coron, J.-S. and Goubin, L. On boolean and arithmetic masking against differential power analysis. In *Proceedings of the Second International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '00, pages 231–237. Springer-Verlag, London, UK, 2000. ISBN 978-3-540-41455-1. doi:10.1007/3-540-44499-8_18.

Daemen, J. and Rijmen, V. The design of Rijndael: AES – The Advanced Encryption Standard. Springer-Verlag Berlin Heidelberg, 1st edition, 2002. ISBN 978-3-540-42580-9.

Dagum, L. and Menon, R. OpenMP: an industry standard API for shared-memory

programming. *IEEE computational science and engineering*, 5(1):46–55, 1998. ISSN 1070-9924. doi:10.1109/99.660313.

Dobbins, R. Mirai IoT botnet description and DDoS attack mitigation. *Arbor Threat Intelligence*, 2016.
URL https://www.arbornetworks.com/blog/asert/mirai-iot-botnet-description-ddos-attack-mitigation/

Eastlake, D., 3rd and Jones, P. US Secure Hash Algorithm 1 (SHA1). Technical report, National Security Agency, United States, 2001.
URL https://tools.ietf.org/html/rfc3174

Ellis, M. A. and Stroustrup, B. The Annotated C++ Reference Manual. Addison-Wesley Longman Publishing, 1st edition, 1990. ISBN 978-0201514599.

Ferguson, N. Practical Cryptography. John Wiley & Sons, 1st edition, 2003. ISBN 978-0471223573.

Ferreyra, D. AVR development. 2008. Accessed: 2015-01-02.
URL http://www.bourbonstreetsoftware.com/AVRDevelopment.html

Ferrigno, J. and Hlavac, M. When AES blinks: introducing optical side channel. *IET Information Security*, 2(3):94–98, September 2008. ISSN 1751-8709. doi:10.1049/iet-ifs:20080038.

Finke, T., Gebhardt, M., and Schindler, W. A new side-channel attack on RSA prime generation. In *Proceedings of the 11th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '09, pages 141–155. Springer-Verlag, Berlin, Heidelberg, 2009. ISBN 978-3-642-04137-2. doi:10.1007/978-3-642-04138-9_11.

Fraser, C. W. and Hanson, D. R. A retargetable C compiler: design and implementation. Addison-Wesley Professiona, 1st edition, 1995. ISBN 978-0805316704.

Friedman, J. Tempest: A signal problem. *NSA Cryptologic Spectrum*, 1972. Accessed: 2015-02-13.
URL http://jproc.ca/crypto/tempest.pdf

Frieslaar, I. and Irwin, B. An investigation into the signals leakage from a smartcard based on different runtime code. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, pages 279–284. 2015. ISBN 978-0-620-67151-4.

Frieslaar, I. and Irwin, B. Evaluating the multi-threading countermeasure. *International Journal of Computer Science and Information Security*, 14(12):379–387, 2016a. ISSN 1947 5500.

Frieslaar, I. and Irwin, B. Investigating multi–thread utilization as a software defence mechanism against side channel attacks. In *Proceedings of the 8th International Conference on Signal Processing Systems*, ICSPS 2016, pages 189–193. ACM, New York, NY, USA, 2016b. ISBN 978-1-4503-4790-7. doi:10.1145/3015166.3015176.

Frieslaar, I. and Irwin, B. A multi-threading approach to secure verifypin. In *2016 2nd International Conference on Frontiers of Signal Processing (ICFSP)*, pages 32–37. Oct 2016c. ISBN 978-1-5090-3815-2. doi:10.1109/ICFSP.2016.7802952.

Frieslaar, I. and Irwin, B. Investigating the effects various compilers have on the electromagnetic signature of a cryptographic executable. In *Proceedings of the South African Institute of Computer Scientists and Information Technologists (SAICSIT 2017)*, SACSIT 2017. ACM, New York, NY, USA, 2017a. ISBN 978-145035250-5/17/09. doi:10.1145/3129416.3129436.

Frieslaar, I. and Irwin, B. Investigating the electromagnetic leakage from a Raspberry Pi. In *2017 Information Security South Africa (ISSA)*. August 2017b. ISBN 978-1-5386-0544-8.

Frieslaar, I. and Irwin, B. Investigating the utilization of the secure hash algorithm to generate electromagnetic noise. In *Proceedings of the 9th International Conference on Signal Processing Systems*, ICSPS 2017. ACM, New York, NY, USA, 2017c. ISBN 978-1-4503-5384-7.

Frieslaar, I. and Irwin, B. Recovering AES-128 encryption keys from a Raspberry Pi. In *Southern Africa Telecommunication Networks and Applications Conference (SATNAC)*, pages 228–235. September 2017d. ISBN 978-0-620-76756-9.

Frieslaar, I. and Irwin, B. Developing an electromagnetic noise generator to protect a Raspberry Pi from side channel analysis. *SAIEE Africa Research Journal*, 109:85 – 101, 2018. ISSN 1991-1696.

Gandolfi, K., Mourtel, C., and Olivier, F. Electromagnetic analysis: Concrete results. In *Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '01, pages 251–261. Springer-Verlag, London, UK, UK, 2001. ISBN 978-3-540-42521-2. doi:10.1007/3-540-44709-1_21.

GCC. GCC optimization. 2013. Accessed: 2017-02-01.
URL https://wiki.gentoo.org/wiki/GCC_optimization

Genkin, D., Pachmanov, L., Pipman, I., and Tromer, E. Stealing keys from PCs using a radio: Cheap electromagnetic attacks on windowed exponentiation. In *Cryptographic Hardware and Embedded Systems – CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 207–228. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-48323-7. doi: 10.1007/978-3-662-48324-4_11.

Genkin, D., Pachmanov, L., Pipman, I., Tromer, E., and Yarom, Y. ECDSA key extraction from mobile devices via nonintrusive physical side channels. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 1626–1638. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4139-4. doi:10.1145/2976749.2978353.

Genkin, D., Pipman, I., and Tromer, E. Get your hands off my laptop: Physical side-channel key-extraction attacks on PCs. In *Proceedings of the 16th International Workshop on Cryptographic Hardware and Embedded Systems — CHES 2014 - Volume 8731*, pages 242–260. Springer-Verlag New York, Inc., New York, NY, USA, 2014. ISBN 978-3-662-44708-6. doi:10.1007/978-3-662-44709-3_14.

Genkin, D., Shamir, A., and Tromer, E. Acoustic cryptanalysis. *Journal of Cryptology*, 30:392–443, Apr 2017. ISSN 1432-1378. doi:10.1007/s00145-015-9224-2.

Golic, J. D. and Tymen, C. Multiplicative masking and power analysis of AES. In *Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '02, pages 198–212. Springer-Verlag, London, UK, 2003. ISBN 978-3-540-00409-7. doi:10.1007/3-540-36400-5_16.

Goller, G. and Sigl, G. Side channel attacks on smartphones and embedded devices using standard radio equipment. In *Constructive Side-Channel Analysis and Secure Design: 6th International Workshop, COSADE 2015, Berlin, Germany, April 13-14, 2015. Revised Selected Papers*, pages 255–270. Springer International Publishing, 2015. ISBN 978-3-319-21476-4. doi:10.1007/978-3-319-21476-4_17.

Gornik, A., Moradi, A., Oehm, J., and Paar, C. A hardware-based countermeasure to reduce side-channel leakage: Design, implementation, and evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(8):1308–1319, Aug 2015. ISSN 0278-0070. doi:10.1109/TCAD.2015.2423274.

Goubin, L. and Patarin, J. DES and differential power analysis (the "duplication" method). In *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems*, CHES '99, pages 158–172. Springer-Verlag, London, UK, UK, 1999. ISBN 978-3-540-66646-2. doi:10.1007/3-540-48059-5_15.

Gough, B. and Stallman, R. An Introduction to GCC: For the GNU Compilers Gcc and G++. Network Theory, 1st edition, 2004. ISBN 978-0954161798.

Hamming, R. W. Error detecting and error correcting codes. *The Bell System Technical Journal*, 29(2):147–160, April 1950. ISSN 0005-8580. doi:10.1002/j.1538-7305.1950.tb00463.x.

Hennessy, J. L. and Patterson, D. A. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, 5th edition, 2011. ISBN 978-0123838728.

Heron, S. Advanced Encryption Standard (AES). *Netwwork Security.*, pages 8–12, 2009. ISSN 1353-4858. doi:10.1016/S1353-4858(10)70006-4.

Hörmann, W. and Leydold, J. Continuous random variate generation by fast numerical inversion. *ACM Transactions on Modeling and Computer Simulation*, 13(4):347–362, October 2003. ISSN 1049-3301. doi:10.1145/945511.945517.

Hutter, M., Mangard, S., and Feldhofer, M. Power and EM attacks on passive 13.56 MHz RFID devices. In *Cryptographic Hardware and Embedded Systems - CHES 2007: 9th International Workshop, Vienna, Austria, September 10-13, 2007. Proceedings*, pages 320–333. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-74735-2. doi:10.1007/978-3-540-74735-2_22.

Hutter, M. and Schmidt, J.-M. The temperature side channel and heating fault attacks. *Smart Card Research and Advanced Applications: 12th International Conference, CARDIS 2013, Berlin, Germany, November 27-29, 2013. Revised Selected Papers*, pages 219–235, 2014. doi:10.1007/978-3-319-08302-5_15.

IBM. CryptoCard. 2017. Accessed: 2017-07-01.
URL https://www-03.ibm.com/security/cryptocards/

Johnson, S. C. A tour through the portable C compiler. In *Unix Programmer's Manual, 7th Edition, 2B, Section 33*. 1981.

Kahn, D. The Codebreakers: The story of Secret Writing. Scribner, 1st edition, 1996a. ISBN 978-0684831305.

Kahn, D. The history of steganography. In *Information Hiding: First International Workshop Cambridge, U.K., May 30 – June 1, 1996 Proceedings*, pages 1–5. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996b. ISBN 978-3-540-49589-5. doi:10.1007/3-540-61996-8_27.

Kamoun, N., Bossuet, L., and Ghazel, A. Correlated power noise generator as a low cost DPA countermeasures to secure hardware AES cipher. In *2009 3rd International Conference on Signals, Circuits and Systems (SCS)*, pages 1–6. Nov 2009. ISBN 978-1-4244-4397-0. doi:10.1109/ICSCS.2009.5412604.

Katz, J. and Lindell, Y. Introduction to Modern Cryptography, Second Edition. Chapman & Hall/CRC, 2nd edition, 2014. ISBN 9781466570269.

Kelly, S. D. T., Suryadevara, N. K., and Mukhopadhyay, S. C. Towards the implementation of IoT for environmental condition monitoring in homes. *IEEE Sensors Journal*, 13(10):3846–3853, Oct 2013. ISSN 1530-437X. doi:10.1109/JSEN.2013.2263379.

Kelsey, J., Schneier, B., Wagner, D., and Hall, C. Side channel cryptanalysis of product ciphers. *Computer Security*, 8(2,3):141–158, August 2000. ISSN 0926-227X. doi:10.1007/BFb0055858.

Kim, N. S., Austin, T., Baauw, D., Mudge, T., Flautner, K., Hu, J. S., Irwin, M. J., Kandemir, M., and Narayanan, V. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, Dec 2003. ISSN 0018-9162. doi:10.1109/MC.2003.1250885.

Kioumars, A. H. and Tang, L. ATmega and XBee-based wireless sensing. In *The 5th International Conference on Automation, Robotics and Applications*, pages 351–356. Dec 2011. ISBN 978-1-4577-0330-0. doi:10.1109/ICARA.2011.6144908.

Knudsen, L. R. Block ciphers-analysis, design and applications. Ph.D. thesis, AARHUS University: Departmnt of Computer Science, 1994. doi:10.7146/dpb.v23i485.6978.

Koc, C. K. RSA hardware implementation. *RSA Laboratories, August*, 1995. Accessed: 2017-01-15.
URL https://koclab.cs.ucsb.edu/docs/koc/r02.pdf

Kocher, P., Jaffe, J., and Jun, B. Differential power analysis. In *Advances in Cryptology — CRYPTO' 99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings*, pages 388–397. Springer Berlin Heidelberg, Berlin, Heidelberg, 1999. ISBN 978-3-540-66347-8. doi:10.1007/3-540-48405-1_25.

Kocher, P., Jaffe, J., Jun, B., and Rohatgi, P. Introduction to differential power

analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, Apr 2011. ISSN 2190-8516. doi:10.1007/s13389-011-0006-y.

Kocher, P. C. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology – CRYPTO '96: 16th Annual International Cryptology Conference Santa Barbara, California, USA August 18–22, 1996 Proceedings*, pages 104–113. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996. ISBN 978-3-540-68697-2. doi:10.1007/3-540-68697-5_9.

Köpf, B. and Basin, D. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 286–296. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-703-2. doi:10.1145/1315245.1315282.

Kozaczuk, W. Enigma: how the German machine cipher was broken, and how it was read by the Allies in World War Two. Arms and Armour Press, 1st edition, 1984. ISBN 978-0313270079.

Kruh, L. and Deavours, C. The commercial enigma: Beginnings of machine cryptography. *Cryptologia*, 26(1):1–16, January 2002. ISSN 0161-1194. doi:10.1080/0161-110291890731.

Kumar, N. S., Saravanan, M., and Jeevananthan, S. Microprocessors and Microcontrollers. Oxford University Press, 1st edition, 2011. ISBN 978-0198066477.

Kunikowski, W., Czerwiński, E., Olejnik, P., and Awrejcewicz, J. An overview of ATmega AVR microcontrollers used in scientific research and industrial applications. *Pomiary Automatyka Robotyka*, 19(215):15–20, 1 2015. doi:10.14313/PAR_215/15.

Langner, R. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security and Privacy*, 9(3):49–51, May 2011. ISSN 1540-7993. doi:10.1109/MSP.2011.67.

Lattner, C. LLVM and clang: Next generation compiler technology. 2008. Accessed: 2017-03-10.

URL https://www.academia.edu/7382406/LLVM_and_Clang_Next_Generation_Compiler_Technology

Lattner, C. and Adve, V. LLVM: a compilation framework for lifelong program analysis transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86. March 2004. ISBN 0-7695-2102-9. doi: 10.1109/CGO.2004.1281665.

Le, T.-H., Canovas, C., and Clédière, J. An overview of side channel analysis attacks. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security*, ASIACCS '08, pages 33–43. ACM, 2008. ISBN 978-1-59593-979-1. doi:10.1145/1368310.1368319.

Lee, D.-U., Luk, W., Villasenor, J. D., and Cheung, P. Y. K. A Gaussian noise generator for hardware-based simulations. *IEEE Transactions on Computers*, 53(12):1523–1534, December 2004. ISSN 0018-9340. doi:10.1109/TC.2004.106.

Lee, D.-U., Luk, W., Villasenor, J. D., Zhang, G., and Leong, P. H. W. A hardware Gaussian noise generator using the Wallace method. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 13(8):911–920, August 2005. ISSN 1063-8210. doi:10.1109/TVLSI.2005.853615.

Lee, R. M., Assante, M. J., and Conway, T. German steel mill cyber attack. *Industrial Control Systems*, 2014.
URL https://ics.sans.org/media/ICS-CPPE-case-Study-2-German-Steelworks_Facility.pdf

Levine, J. Flex & Bison: Text Processing Tools. O'Reilly Media, 1st edition, 2009. ISBN 978-0596155971.

Levinson, L., Männer, R., Sessler, M., and Simmler, H. Preemptive multitasking on FPGAs. In *Proceedings of the 2000 IEEE Symposium on Field-Programmable Custom Computing Machines*, FCCM '00, pages 301–. IEEE Computer Society, Washington, DC, USA, 2000. ISBN 0-7695-0871-5. doi:10.1109/FPGA.2000.903426.

LLVM. LLVM's analysis and transform passes. 2013. Accessed: 2017-02-01. URL http://llvm.org/docs/Passes.html

Longo, J., De Mulder, E., Page, D., and Tunstall, M. SoC it to EM: Electromagnetic side-channel attacks on a complex System-on-Chip. In *Cryptographic Hardware and Embedded Systems – CHES 2015: 17th International Workshop, Saint-Malo, France, September 13-16, 2015, Proceedings*, pages 620–640. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-48324-4. doi:10.1007/978-3-662-48324-4_31.

Macq, B. M. and Quisquater, J.-J. Cryptology for digital TV broadcasting. *Proceedings of the IEEE*, 83(6):944–957, Jun 1995. ISSN 0018-9219. doi:10.1109/5.387094.

Mane, S. H. Implementation of SCA-Resistant CPU and an ECDLP Engine on FPGA Platform. Master's thesis, Virginia Polytechnic Institute and State University, 2012.

Mangard, S., Oswald, E., and Popp, T. Power analysis attacks: Revealing the secrets of smart cards. Springer Science & Business Media, 1st edition, 2008. ISBN 978-0387381626.

Martin, D. P., O'Connell, J. F., Oswald, E., and Stam, M. Counting keys in parallel after a side channel attack. In **Iwata, T. and Cheon, J. H.**, editors, *Advances in Cryptology – ASIACRYPT 2015*, pages 313–337. Springer Berlin Heidelberg, Berlin, Heidelberg, 2015. ISBN 978-3-662-48800-3.

Matsui, M. Linear cryptanalysis method for DES cipher. In *Workshop on the Theory and Application of Cryptographic Techniques on Advances in Cryptology*, EUROCRYPT '93, pages 386–397. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1994. ISBN 978-3-540-57600-6. doi:10.1007/3-540-48285-7_33.

Mayes, K. Smart Cards, Tokens, Security and Applications. Springer, 2nd edition, 2017. ISBN 978-3319504988.

McLaughlin, J. NSA looking to exploit internet of things. 2016. Accessed: 2017-04-01. URL https://theintercept.com/2016/06/10/nsa-looking-to-exploit-internet-of-things-including-biomedical-devices-official-says/

Menezes, A. J., Vanstone, S. A., and Oorschot, P. C. V. Handbook of Applied Cryptography. CRC Press, Inc., 1st edition, 1996. ISBN 978-0849385230.

Mesquita, D., Techer, J.-D., Torres, L., Robert, M., Cathebras, G., Sassatelli, G., and Moraes, F. G. Current Mask Generation: an Analog Circuit to Thwart DPA Attacks, pages 317–330. Springer US, 2007. ISBN 978-0-387-73661-7. doi:10.1007/978-0-387-73661-7_20.

Mesquita, D., Techer, J. D., Torres, L., Sassatelli, G., Cambon, G., Robert, M., and Moraes, F. Current mask generation: A transistor level security against DPA attacks. In *2005 18th Symposium on Integrated Circuits and Systems Design*, pages 115–120. Sept 2005. ISBN 1-59593-174-0. doi:10.1109/SBCCI.2005.4286842.

Messerges, T. S. Securing the AES finalists against power analysis attacks. In *Proceedings of the 7th International Workshop on Fast Software Encryption*, FSE '00, pages 150–164. Springer-Verlag, London, UK, 2001. ISBN 978-3-540-41728-6. doi: 10.1007/3-540-44706-7_11.

Mohammad, A., Jararweh, Y., and Tawalbeh, L. A. Aes-512: 512-bit advanced encryption standard algorithm design and evaluation. *2011 7th International Conference on Information Assurance and Security (IAS)*, pages 292–297, 2011. doi: 10.1109/ISIAS.2011.6122835.

Mössenböck, H. A generator for production quality compilers. In *Compiler Compilers: Third International Workshop, CC '90 Schwerin, FRG, October 22–24, 1990 Proceedings*, pages 42–55. Springer Berlin Heidelberg, Berlin, Heidelberg, 1991. ISBN 978-3-540-46953-7. doi:10.1007/3-540-53669-8_73.

Mulder, E. D., Buysschaert, P., Ors, S. B., Delmotte, P., Preneel, B., Vandenbosch, G., and Verbauwhede, I. Electromagnetic analysis attack on an FPGA implementation of an Elliptic Curve Cryptosystem. In *EUROCON 2005 - The International Conference on "Computer as a Tool"*, volume 2, pages 1879–1882. Nov 2005. ISBN 1-4244-0049-X. doi:10.1109/EURCON.2005.1630348.

Nakano, Y., Souissi, Y., Nguyen, R., Sauvage, L., Danger, J.-L., Guilley, S., Kiyomoto, S., and Miyake, Y. A pre-processing composition for secret key recovery on Android smartphone. In *Proceedings of the 8th IFIP WG 11.2 International Workshop on Information Security Theory and Practice. Securing the Internet of Things – Volume 8501*, pages 76–91. Springer-Verlag New York, Inc., New York, NY, USA, 2014. ISBN 978-3-662-43825-1. doi:10.1007/978-3-662-43826-8_6.

Neuvo, Y., Cheng-Yu, D., and Mitra, S. Interpolated finite impulse response filters. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 32:563–570, 1984. ISSN 0096-3518. doi:10.1109/TASSP.1984.1164348.

NewAe. H-field probe. 2017. Accessed: 2017-08-01.
URL http://store.newae.com/h-field-probe-old-version/

NewAE Technology. CW1002 ChipWhisperer Capture Rev2. 2017a. Accessed: 2015-08-01.
URL https://wiki.newae.com/CW1002_ChipWhisperer_Capture_Rev2

NewAE Technology. CW1173 ChipWhisperer-Lite. 2017b. Accessed: 2016-04-01.
URL https://wiki.newae.com/CW1173_ChipWhisperer-Lite

Nie, S., Liu, L., and Du, Y. Free-fall: Hacking Tesla from wireless to CAN bus. Technical report, Keen Security Lab of Tencent, 2017. Accessed: 2017-07-15.
URL https://www.blackhat.com/docs/us-17/thursday/us-17-Nie-Free-Fall-Hacking-Tesla-From-Wireless-To-CAN-Bus-wp.pdf

Novak, R. SPA-based adaptive chosen-ciphertext attack on RSA implementation. In *Proceedings of the 5th International Workshop on Practice and Theory in Public Key Cryptosystems: Public Key Cryptography*, PKC '02, pages 252–262. Springer-Verlag, London, UK, UK, 2002. ISBN 978-3-540-43168-8. doi:10.1007/3-540-45664-3_18.

Novillo, D. GCC – an architectural overview, current status and future. In *Proceedings of the Linux Symposium*, pages 185–200. 2006. Accessed: 2017-05-01.
URL https://www.kernel.org/doc/ols/2006/ols2006v2-pages-193-208.pdf

NXP. BGA2801. 2015. Accessed: 2015-08-02.
URL https://www.nxp.com/docs/en/data-sheet/BGA2801.pdf

O'Flynn, C. and Chen, Z. A case study of side-channel analysis using decoupling capacitor power measurement with the OpenaDC. In *Foundations and Practice of Security: 5th International Symposium, FPS 2012, Montreal, QC, Canada, October 25-26, 2012, Revised Selected Papers*, pages 341–356. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-37119-6. doi:10.1007/978-3-642-37119-6_22.

O'Flynn, C. and Chen, Z. D. Chipwhisperer: An open-source platform for hardware embedded security research. In *Constructive Side-Channel Analysis and Secure Design: 5th International Workshop, COSADE 2014, Paris, France, April 13-15, 2014. Revised Selected Papers*, pages 243–260. Springer International Publishing, 2014. ISBN 978-3-319-10175-0. doi:10.1007/978-3-319-10175-0_17.

O'Flynn, C. and Chen, Z. D. Side channel power analysis of an AES-256 bootloader. In *2015 IEEE 28th Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 750–755. May 2015. ISSN 0840-7789. doi:10.1109/CCECE.2015.7129369.

Oranchak, D. Evolutionary algorithm for decryption of monoalphabetic homophonic substitution ciphers encoded as constraint satisfaction problems. In *Proceedings of the 10th Annual Conference on Genetic and Evolutionary Computation*, GECCO '08, pages 1717–1718. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-130-9. doi:10.1145/1389095.1389425.

Oswald, D. and Paar, C. Breaking Mifare DESFire MF3ICD40: Power analysis and templates in the real world. In *Cryptographic Hardware and Embedded Systems – CHES 2011: 13th International Workshop, Nara, Japan, September 28 – October 1, 2011. Proceedings*, pages 207–222. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011. ISBN 978-3-642-23951-9. doi:10.1007/978-3-642-23951-9_14.

Oswald, E., Mangard, S., Pramstaller, N., and Rijmen, V. A side-channel analysis resistant description of the AES S-Box. In *Fast Software Encryption: 12th International*

Workshop, FSE 2005, Paris, France, February 21-23, 2005, Revised Selected Papers, pages 413–423. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31669-5. doi:10.1007/11502760_28.

Ott, H. W. Noise reduction techniques in electronic systems. Wiley New York, 2nd edition, 1988. ISBN 978-0471850687.

Paganini, P. OVH hosting hit by 1Tbps DDoS attack. 2016. Accessed: 2016-12-01. URL http://securityaffairs.co/wordpress/51640/cyber-crime/tbps-ddos-attack.html

Pan, J. and Tompkins, W. J. A real-time QRS detection algorithm. *IEEE Transactions on Biomedical Engineering*, BME-32(3):230–236, March 1985. ISSN 0018-9294. doi:10.1109/TBME.1985.325532.

Parkinson, S., Ward, P., Wilson, K., and Miller, J. Cyber threats facing autonomous and connected vehicles: Future challenges. *IEEE Transactions on Intelligent Transportation Systems*, PP(99):1–18, 2017. ISSN 1524-9050. doi:10.1109/TITS.2017.2665968.

Parzen, E. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 33(3):1065–1076, 1962. ISSN 00034851. doi:10.1214/aoms/1177704472.

Patterson, D. A. and Hennessy, J. L. Computer organization and design: the hardware/software interface. Morgan Kaufmann Publishers Inc., 2nd edition, 1998. ISBN 978-0124077263.

Plos, T. Susceptibility of UHF RFID tags to electromagnetic analysis. In *Topics in Cryptology – CT-RSA 2008: The Cryptographers' Track at the RSA Conference 2008, San Francisco, CA, USA, April 8-11, 2008. Proceedings*, pages 288–300. Springer, Berlin, Heidelberg, 2008. ISBN 978-3-540-79263-5. doi:10.1007/978-3-540-79263-5_18.

Popp, T. and Mangard, S. Masked dual-rail pre-charge logic: DPA-resistance without routing constraints. In *Proceedings of the 7th International Conference on Cryptographic*

Hardware and Embedded Systems, CHES'05, pages 172–186. Springer-Verlag, Berlin, Heidelberg, 2005. ISBN 978-3-540-28474-1. doi:10.1007/11545262_13.

Power, E. A. Introductory quantum electrodynamics. American Elsevier Publishing Company, 1st edition, 1965. ISBN 978-0582463141.

Preneel, B. Cryptographic hash functions. *European Transactions on Telecommunications*, 5(4):431–448, 1994. ISSN 1541-8251. doi:10.1002/ett.4460050406.

Priestley, M. B. Spectral Analysis and Time Series. Academic Press, 1st edition, 1983. ISBN 978-0125649223.

Prouff, E. and Rivain, M. Masking against side-channel attacks: A formal security proof. In *Advances in Cryptology – EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings*, pages 142–159. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013. ISBN 978-3-642-38348-9. doi:10.1007/978-3-642-38348-9_9.

Quisquater, J.-J. and Samyde, D. Electromagnetic analysis (EMA): Measures and counter-measures for smart cards. In *Proceedings of the International Conference on Research in Smart Cards: Smart Card Programming and Security*, E-SMART '01, pages 200–210. Springer-Verlag, London, UK, UK, 2001. ISBN 978-3-540-42610-3. doi:10.1007/3-540-45418-7_17.

Rabaey, J. M., Chandrakasan, A. P., and Nikolic, B. Digital integrated circuits. Prentice hall Englewood Cliffs, 2nd edition, 2003. ISBN 978-0130909961.

Rabe, D. and Nebel, W. Short circuit power consumption of glitches. In *Proceedings of 1996 International Symposium on Low Power Electronics and Design*, pages 125–128. Aug 1996. ISBN 0-7803-3571-6. doi:10.1109/LPE.1996.547493.

Radcliffe, J. Hacking medical devices for fun and insulin: Breaking the human SCADA system. Technical report, SecureConcern, 2012. Accessed: 2017-07-15.
URL https://media.blackhat.com/bh-us-11/Radcliffe/BH_US_11_Radcliffe_Hacking_Medical_Devices_WP.pdf

Rao, P. B. and Uma, S. Raspberry Pi home automation with wireless sensors using smart phone. *International Journal of Computer Science and Mobile Computing*, 4(5):797–803, 2015. ISSN 2320–088X.

Raspberry Pi. Raspberry Pi 2 model b. 2015. Accessed: 2016-06-01. URL https://www.raspberrypi.org/products/raspberry-pi-2-model-b/

Ratanpal, G. B., Williams, R. D., and Blalock, T. N. An on-chip signal suppression countermeasure to power analysis attacks. *IEEE Transactions on Dependable and Secure Computing*, 1(3):179–189, July 2004. ISSN 1545-5971. doi:10.1109/TDSC.2004.25.

Rau, B. R. and Fisher, J. A. Instruction-level parallelism. John Wiley and Sons Ltd., 1st edition, 2001. ISBN 978-1558601925.

Raymond, S. and Jewett, J. Physics for scientists and engineers. Brooks Cole, 9th edition, 2013. ISBN 978-1133947271.

Reinders, J. Intel threading building blocks: outfitting C++ for multi-core processor parallelism. O'Reilly Media, 1st edition, 2007. ISBN 978-0596514808.

Renauld, M. and Standaert, F.-X. Algebraic side-channel attacks. In **Bao, F., Yung, M., Lin, D., and Jing, J.**, editors, *Information Security and Cryptology*, pages 393–410. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010. ISBN 978-3-642-16342-5.

Renauld, M., Standaert, F.-X., and Veyrat-Charvillon, N. Algebraic side-channel attacks on the aes: Why time also matters in dpa. In **Clavier, C. and Gaj, K.**, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 97–111. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-04138-9.

Ring, T. Connected cars - the next targe tfor hackers. *Network Security*, 2015(11):11–16, November 2015. ISSN 1353-4858. doi:10.1016/S1353-4858(15)30100-8.

Rolf, S. Design of Analog Filters. Oxford University Press, 2nd edition, 2009. ISBN 978-0195373943.

Russell, D. and Gangemi, G. Computer security basics. O'Reilly & Associates, 2nd edition, 2006. ISBN 978-0596006693.

Sadiku, M. N. Elements of electromagnetics. Oxford university press, 6th edition, 2014. ISBN 978-0199321384.

Sakoe, H. and Chiba, S. Dynamic programming algorithm optimization for spoken word recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 26(1):43–49, Feb 1978. ISSN 0096-3518. doi:10.1109/TASSP.1978.1163055.

Salmon, D., Zeller, M., Guzman, A., Mynam, V., and Donolo, M. Mitigating the aurora vulnerability with existing technology. In *proceedings of the 36th Annual Western Protective Relay Conference*. 2009.

Salvador, S. and Chan, P. Toward accurate dynamic time warping in linear time and space. *Intell. Data Anal.*, 11(5):561–580, October 2007. ISSN 1088-467X. doi:10.1002/cav.v17:3/4.

Satorra, A. and Bentler, P. M. A scaled difference chi-square test statistic for moment structure analysis. *Psychometrika*, 66(4):507–514, Dec 2001. ISSN 1860-0980. doi:10.1007/BF02296192.

Savitzky, A. and Golay, M. J. Smoothing and differentiation of data by simplified least squares procedures. *Analytical Chemistry*, 36(8):1627–1639, 1964. doi:10.1021/ac60214a047.

Scahill, J. and Begley, J. The Great Sim Hack. 2015. Accessed: 2017-02-01. URL https://theintercept.com/2015/02/19/great-sim-heist/

Schafer, R. W. What is a savitzky-golay filter? [lecture notes]. *IEEE Signal Processing Magazine*, 28(4):111–117, July 2011. ISSN 1053-5888. doi:10.1109/MSP.2011.941097.

Schindler, W., Lemke, K., and Paar, C. A Stochastic Model for Differential Side Channel Cryptanalysis, pages 30–46. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. ISBN 978-3-540-31940-5. doi:10.1007/11545262_3.

Schlösser, A., Nedospasov, D., Krämer, J., Orlic, S., and Seifert, J.-P. Simple photonic emission analysis of AES. In *Cryptographic Hardware and Embedded Systems – CHES 2012: 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, pages 41–57. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-642-33027-8. doi:10.1007/978-3-642-33027-8_3.

Schneier, B. A self-study course in block-cipher cryptanalysis. *Cryptologia*, 24(1):18–33, January 2000. ISSN 0161-1194. doi:10.1080/0161-110091888754.

Schramm, K., Wollinger, T., and Paar, C. A new class of collision attacks and its application to DES. In *Fast Software Encryption: 10th International Workshop, FSE 2003, Lund, Sweden, February 24-26, 2003. Revised Papers*, pages 206–222. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003. ISBN 978-3-540-39887-5. doi:10.1007/978-3-540-39887-5_16.

Shamir, A. and Tromer, E. Acoustic cryptanalysis. 2004. Accessed: 2017-06-10. URL http://www.wisdom.weizmann.ac.il/~tromer

Simmons, G. J. Authentication theory/coding theory. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 411–431. Springer-Verlag New York, Inc., New York, NY, USA, 1985. ISBN 978-3-540-15658-1. doi:10.1007/3-540-39568-7_32.

Skorobogatov, S. Using optical emission analysis for estimating contribution to power analysis. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 111–119. Sept 2009. ISBN 978-1-4244-4972-9. doi:10.1109/FDTC.2009.39.

Smith, S. W. The Scientist and Engineer's Guide to Digital Signal Processing. California Technical, 1st edition, 1997. ISBN 978-0966017632.

Stallman, R. Using and porting the GNU compiler collection. Iuniverse Inc, 1st edition, 2000. ISBN 978-0595100354.

Standard, Data Encryption and others. Federal information processing standards publication 46. *National Bureau of Standards, US Department of Commerce*, 1977.

Statista. Internet of things (IoT) connected devices installed base worldwide. 2017. Accessed: 2017-02-01.
URL https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

Sun, K. Adaptive motion estimation based on statistical sum of absolute difference. In *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269)*, pages 601–604. Oct 1998. ISBN 0-8186-8821-1. doi:10.1109/ICIP.1998.727335.

Sun Microsystems. The Java card 3 platform. Technical report, Sun Microsystems, 2008. Accessed: 2016-02-01.
URL http://www.oracle.com/technetwork/articles/javase/javacard3-whitepaper-149761.pdf

Talbot, D. B. Frequency Acquisition Techniques for Phase Locked Loops. Wiley-IEEE Press, 1st edition, 2012. ISBN 978-1118168103.

Tanenbaum, A. S., Woodhull, A. S., Tanenbaum, A. S., and Tanenbaum, A. S. Operating systems: design and implementation. Prentice-Hall Englewood Cliffs, NJ, 3rd edition, 1987. ISBN 978-0131429383.

Telandro, V., Kussener, E., Barthélemy, H., and Malherbe, A. A bi-channel voltage regulator protecting smart cards against power analysis attacks. *Analog Integrated Circuits and Signal Processing*, 59(3):275–285, Jun 2009. ISSN 1573-1979. doi:10.1007/s10470-008-9260-z.

Telandro, V., Kussener, E., Malherbe, A., and Barthelemy, H. On-Chip voltage regulator protecting against power analysis attacks. In *2006 49th IEEE International Midwest Symposium on Circuits and Systems*, volume 2, pages 507–511. Aug 2006. ISBN 1-4244-0172-0. doi:10.1109/MWSCAS.2006.381778.

Tillich, S. and Großschädl, J. Power Analysis Resistant AES Implementation with Instruction Set Extensions, pages 303–319. Springer Berlin Heidelberg, 2007. ISBN 978-3-540-74734-5. doi:10.1007/978-3-540-74735-2_21.

Tillich, S., Herbst, C., and Mangard, S. Protecting AES software implementations on 32-bit processors against power analysis. In *Proceedings of the 5th International Conference on Applied Cryptography and Network Security*, ACNS '07, pages 141–157. Springer-Verlag, Berlin, Heidelberg, 2007. ISBN 978-3-540-72737-8. doi:10.1007/978-3-540-72738-5_10.

Tsui, C.-Y., Pedram, M., and Despain, A. M. Efficient estimation of dynamic power consumption under a real delay model. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '93, pages 224–228. IEEE Computer Society Press, Los Alamitos, CA, USA, 1993. ISBN 0-8186-4490-7. doi:10.1109/ICCAD.1993.580061.

van Woudenberg, J. G. J., Witteman, M. F., and Bakker, B. Improving differential power analysis by elastic alignment. In *Proceedings of the 11th International Conference on Topics in Cryptology: CT-RSA 2011*, CT-RSA'11, pages 104–119. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-19073-5. doi:10.1007/978-3-642-19074-2_8.

Vaudenay, S. A Classical Introduction to Cryptography. Springer, 1st edition, 2006. ISBN 978-0-387-25880-5.

Veyrat-Charvillon, N., Medwed, M., Kerckhof, S., and Standaert, F.-X. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In *Proceedings of the 18th International Conference on The Theory and Application of Cryptology and Information Security*, ASIACRYPT'12, pages 740–757. Springer-Verlag, Berlin, Heidelberg, 2012. ISBN 978-3-642-34960-7. doi:10.1007/978-3-642-34961-4_44.

Von Hagen, W. The definitive guide to GCC. Definitive Guides. Apress, 2nd edition, 2006. ISBN 978-1590595855.

Wei, D., Lu, Y., Jafari, M., Skare, P., and Rohde, K. An integrated security system of protecting smart grid against cyber attacks. In *2010 Innovative Smart Grid Technologies (ISGT)*, pages 1–7. Jan 2010. ISBN 978-1-4244-6264-3. doi:10.1109/ISGT.2010.5434767.

Welch, P. The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms. *IEEE Transactions on Audio and Electroacoustics*, 15(2):70–73, Jun 1967. ISSN 0018-9278. doi:10.1109/TAU.1967.1161901.

Weste, N. H. and Eshraghian, K. Principles of CMOS VLSI design. Addison-Wesley New York, 2nd edition, 1985. ISBN 978-0201533767.

Wrixon, F. B. Codes, ciphers & other cryptic & clandestine communication: Making and breaking secret messages from hieroglyphs to the Internet. Black Dog & Leventhal Paperbacks, 1st edition, 2005. ISBN 978-1579124854.

XILINX. Spartan-6 family overview. 2011. Accessed: 2015-08-01. URL https://www.xilinx.com/support/documentation/data_sheets/ds160.pdf

Yamaguchi, I. Phase-shifting digital holography. In *Digital Holography and Three-Dimensional Display: Principles and Applications*, pages 145–171. Springer US, Boston, MA, 2006. ISBN 978-0-387-31340-5. doi:10.1007/0-387-31397-4_5.

Yun, M. and Yuxin, B. Research on the architecture and key technology of Internet of Things (IoT) applied on smart grid. In *2010 International Conference on Advances in Energy Engineering*, pages 69–72. June 2010. ISBN 978-1-4244-7831-6. doi:10.1109/ICAEE.2010.5557611.

Glossary

This section list all the acronyms mentioned in this work. Each acronym is followed by the page number where it has first been mentioned.

3DES Triple Data Encryption Standard. 19

ADC Analog-to-Digital. 63
AES Advance Encryption Standard. 4
ALU Arithmetic Logic Unit. 39
API Application Program Interface. 164
ATM Automated Teller Machine. 104

CA Certificate Authority. 18
CMOS Complementary Metal Oxide Semiconductor. 37
CPA Correlation Power Analysis. 6
CPU Central Processing Unit. 195
CU Control Unit. 39

DAC Digital-to-Analog. 93
DB Decibel. 123
DDoS Distributed Denial-of-Service Attack. 195
DEMA Differential Electromagnetic Analysis. 30
DES Data Encryption Standard. 4, 209
DFT Discrete Fourier Transform. 82
DPA Differential Power Analysis. 6
DTW Dynamic Time Warping. 89

ECDSA Elliptic Curve Digital Signature Algorithm. 36
EEPROM Electrically Erasable Programmable Read-Only Memory. 30
EM Electromagnetic. 4

FastDTW Fast Dynamic Time Warping. 89
FFT Fast Fourier Transform. 82
FIR Finite Impulse Response. 180
FPGA Field-Programmable Gate Array. 4

GB Gigabyte. 100
GCC GNU Compiler Collection. 75
GCD Greatest Common Divisor. 114
GS/s Giga-Samples per Second. 94
GUI Graphical User Interface. 77

IBM International Business Machines. 63
IC Integrated Circuit. 63
IDE Integrated Development Environment. 77
IO Input/Output. 95
IoT Internet of Things. 1
IR Intermediate Representation. 75

JC Java Card. 107

kHz Kilohertz. 100

LALR Look-Ahead Left-to-Right. 76
LLVM Low Level Virtual Machine. 76
LNA Low Noise Amplifier. 93

MATLAB Matrix Laboratory. 98
MHz Megahertz. 123
ms Milliseconds. 122
MS/s Mega-Samples per Second. 93

NIST National Institute of Standards and Technology. 5
NOP No Operation. 113
NSA National Security Agency. 4

OpenMP Open Multi-Processing. 72

PCB Printed Circuit Board. 45
PDF Probability Density Function. 57
PGE Partial Guessing Entropy. 65
PITA Portable Instrument for Trace Acquisition. 35

PKI Public Key Infrastructure. 17, 18

PLC Programmable Logic Controllers. 2

PLL Phase Locked Loop. 93

PM Phase Modulation. 82

PMOS P-channel Metal Oxide Semiconductor. 37

POSIX Portable Operating System Interface. 71, 72

RAM random Access Memory. 100

RC Resistor Capacitor. 84

RFID Radio-frequency identification. 4

RISC Reduced Instruction Set Computer. 97

RL Resistor Inductor. 84

RPIs Random Process Interrupts. 66

RSA Ron Rivest, Adi Shamir and Leonard Adleman. 4

SAD Sum of Absolute Differences. 86

SCA Side Channel Analysis. 3

SDR Software Defined Radio. 34

SEA Simple Electromagnetic Analysis. 36

SG Savitzky-Golay. 85

SHA Secure Hashing Algorithms. 22

SIM Subscriber Identity Module. 104

SMA SubMiniature version A. 96

SNR Signal-to-Noise Ratio. 46

SPA Simple Power Analysis. 36

SRAM Static Random-Access Memory. 30

TBB Threading Building Blocks. 72

TV Television. 2, 197

UART Universal Asynchronous Receiver/Transmitter. 95

UI User Interface. 73

URL Uniform Resource Locator. 9

USB Universal Serial Bus. 100

A

Lab Environment

Figure A.1 illustrate the lab environment where the experiments of this research was conducted. It is important to note that this was a post graduate Computer Science lab where there were multiple computers and other devices that gives off electromagnetic emissions. In addition, it can be seen that the environment is not encapsulated in a Faraday cage.

Figure A.1: General lab environment.

B

Full URLS

This section displays the full URLs that has been shorten within this document. The table consists of the chapter in which the footnote can be located followed by the shorten and full URL, respectively.

Table B.1: List of shortened URLs.

Chapter	Footnote No.	Shorten URL	Long URL
3	7	https://goo.gl/Z5cjs4	http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.den0024a/CJAGFHED.html
5	1	https://goo.gl/aKUJq7	http://www.oracle.com/technetwork/java/javacard/specs-138637.html
5	2	https://goo.gl/Nh12Pb	http://www.oracle.com/technetwork/articles/javase/javacard3-142122.html
6	4	https://goo.gl/4LjwXV	https://ubuntu-pi-flavour-maker.org/blog/ubuntu-pi-flavours-xenial-point-2-released/

C

Frequency bands

Table C.1: Radio spectrum of frequencies. (Beasley and Miller, 2007)

Frequency	**Designation**	**Abbreviation**
3 – 30 Hz	Extremely low frequency	ELF
30 – 300 Hz	Super low frequency	SLF
300 – 3000 Hz	Ultra low frequency	ULF
3 – 30 kHz	Very low frequency	VLF
30 – 300 kHz	Low frequency	LF
300 kHz – 3 MHz	Medium frequency	MF
3 – 30 MHz	High frequency	HF
30 – 300 MHz	Very high frequency	VHF
300 MHz – 3 GHz	Ultra high frequency	UHF
3 – 30 GHz	Super high frequency	SHF
30 – 300 GHz	Extremely high frequency	EHF
300 GHz – 3 THz	Tremendously high frequency	THF

Table C.2: Abbreviations of frequency.

Abbreviation	**Designation**
Hz	Hertz
kHz	kilohertz
MHz	Megahertz
GHz	Gigahertz
THz	Terahertz

D

Code Implementation

The code provided in this appendix is part of ChipWhisperer[1] software which is under the terms of the GNU General Public License as published by the Free Software Foundation. Sections D.1 – D.2 provide snippets of the Python code utilised by the Correlation Power Analysis (CPA) and template attack, respectively. Furthermore, Section D.3 contains general code examples that has been referred to in this document. Full code can be located at the GitHub repository: https://github.com/ebiefrieslaar/Raspberry-Pi

D.1 Correlation Power Analysis

The code snippets from Listings D.1 – D.4 are from the *attack.py* file.

Listing D.1: Performing the Guess.

```
for bnum in range(0, 16):
  cpaoutput = [0]*256
  for kguess in range(0, 256):
    print "Subkey %d, hyp = %02x"%(bnum, kguess)
    for tnum in range(0, numtraces):
      hypint = HW[intermediate(pt[tnum][bnum], kguess)]
```

[1] ChipWhisperer Repository – https://github.com/newaetech/chipwhisperer

While utilising Python, Equation 3.14 can be simplified into three separate equations as follows:

$$sumnum = \sum_{d=1}^{D}[(h_{d,i} - \bar{h_i})(t_{d,j} - \bar{t_j})]$$
$$sumdem1 = \sum_{d=1}^{D}(h_{d,i} - \bar{h_i})^2$$
$$sumdem2 = \sum_{d=1}^{D}(t_{d,j} - \bar{t_j})^2$$

The three above equations can be coded in Python as follows:

Listing D.2: Performing the Check.

```
#For each trace, do the following
for tnum in range(numtraces):
    hdiff = (hyp[tnum] - meanh)
    tdiff = traces[tnum,:] - meant

    sumnum = sumnum + (hdiff*tdiff)
    sumden1 = sumden1 + hdiff*hdiff
    sumden2 = sumden2 + tdiff*tdiff
```

Note each of these is calculated independently, thus it is possible to avoid looping through every point by using the vector notation of *NumPy*. Finally, the single output vector is calculated and saved it as a specific key guess:

Listing D.3: Saving specific key guess.

```
cpaoutput[kguess] = sumnum / np.sqrt( sumden1 * sumden2 )
```

Listing D.4: Calculating the PGE.

```
bestguess[bnum] = np.argmax(maxcpa)
cparefs = np.argsort(maxcpa)[::-1]
#Find PGE
pge[bnum] = list(cparefs).index(knownkey[bnum])
```

D.2 Template Attack

The code snippets from Listings D.5 – D.8 are from the *TempAttack.py* file.

Listing D.5: Determine Hamming Weight to go with each input.

```
# Note - we're only working with the first byte here
tempSbox = [sbox[tempPText[i][0] ^ tempKey[i][0]] for i in range(len(tempPText))]
tempHW   = [hw[s] for s in tempSbox]
#Sort traces by HW
# Make 9 blank lists - one for each Hamming weight
tempTracesHW = [[] for _ in range(9)]

#Fill them up
for i in range(len(tempTraces)):
  HW = tempHW[i]
  tempTracesHW[HW].append(tempTraces[i])
```

Listing D.6: Calculate the mean.

```
tempMeans = np.zeros((9, len(tempTraces[0])))
for i in range(9):
  tempMeans[i] = np.average(tempTracesHW[i], 0)
```

Listing D.7: Locate the POIs

```
POIs = []
numPOIs = 5
POIspacing = 5
for i in range(numPOIs):
  nextPOI = tempSumDiff.argmax()
  POIs.append(nextPOI)

# Make sure we don't pick a nearby value
poiMin = max(0, nextPOI - POIspacing)
poiMax = min(nextPOI + POIspacing, len(tempSumDiff))
for j in range(poiMin, poiMax):
  tempSumDiff[j] = 0
```

Listing D.8: Determine the mean and covariance matrix for each HW.

```
meanMatrix = np.zeros((9, numPOIs))
covMatrix  = np.zeros((9, numPOIs, numPOIs))
  for HW in range(9):
    for i in range(numPOIs):
    meanMatrix[HW][i] = tempMeans[HW][POIs[i]]
    for j in range(numPOIs):
      x = tempTracesHW[HW][:,POIs[i]]
      y = tempTracesHW[HW][:,POIs[j]]
      covMatrix[HW,i,j] = cov(x, y)
```

D.3 General Code

The code snippet in Listing D.9 is taken from the *aes_enc.c* file in the subroutine of *aes_encrypt_core()*.

Listing D.9: Shuffling of *SubByte* routine.

```
int arrayX[15];
for (int ij = 0; ij < 16; ij++) {    // shuffle array
  int tempP = arrayX[ij];
  int randomIndex = rand() % 16;
  arrayX[ij]= arrayX[randomIndex];
  arrayX[randomIndex] = tempP;}

int pkc=0;
while(pkc<16){ //Shuffe S-box order
  i =  arrayX[pkc];
  state->s[i] ^= ks->key[0].ks[i];
  pkc++;}
```

Listing D.10: Python code for the direct appraoch.

```
status = self.hw.sendAPDU(0x00,0xA4,0x04,0x00,[0xD0,0xD1,0xD2,0xD3,
0xD4,0xD5,0x01,0x01])
if status != 0x9000:
raise IOError("Invalid Status: %x" % status)
(resp, pay) = self.hw.sendAPDU(0x80,0x10,0x01,0x02,[0xAE],rxdatalen=6)
if resp != 0x9000:
raise IOError("Invalid Status: %x" % status)
counter1 = (pay[5] << 8) | pay[6]
counter2 = (pay[7] << 8) | pay[8]
counter3 = (pay[9] << 8) | pay[10]
self.resp = (counter1, counter2, counter3)
```

Listing D.11: Python code for the indirect appraoch.

```
SELECT = [0x00,0xA4,0x04,0x00,0x0B,0x01,0x02,0x03,0x04,0x05,
0x06,0x07,0x08,0x09,0x00,0x00]
APP = [0x80,0x02,0x00,0x00,0x24,0x01,0x02,0x03,0x04,0x05,
0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,0x0E,0x0F,0x1F]
cardtype = AnyCardType()
cardrequest = CardRequest( timeout=10, cardType=cardtype )
cardservice = cardrequest.waitforcard()
observer=ConsoleCardConnectionObserver()
cardservice.connection.addObserver( observer )
cardservice.connection.connect()
apdu = SELECT
response, sw1, sw2 = cardservice.connection.transmit( apdu )
apdu = APP
response, sw1, sw2 = cardservice.connection.transmit( apdu )
self.resp = response.
```

The code snippets from Listing D.10 and D.11 are from the file *SmartCard.py*.

Listing D.12: AES-256 decrypt method.

```
void aes256_decrypt_ecb(aes256_context *ctx, uint8_t *buf){
  .
  .
  .
for (i = 14, rcon = 0x80; --i;){
  makeNoise(); \\Leakage can be seen.
  makeNoise2(); \\Leakage can not be seen.
  if( ( i & 1 ) ){
    aes_expandDecKey(ctx->key, &rcon);
    aes_addRoundKey(buf, &ctx->key[16]);}
  else aes_addRoundKey(buf, ctx->key);

  aes_mixColumns_inv(buf);
  aes_shiftRows_inv(buf);
  aes_subBytes_inv(buf);}

aes_addRoundKey( buf, ctx->key); }
```

The Listing D.12 is part of the ChipWhisperer bundle and is located in the *aes256.c* in the firmware folder.

Listings D.13 and D.14 can be located in *noise.c*.

Listing D.13: Leakage that can be seen.

```
int armMul(int A, int B){
  int x;
  x = A*B;
  return x;
}
printf("%d\n",armMul(10,20));
```

Listing D.14: Leakage that can not be seen.

```
int armMul(int A, int B){
  int x;
  x = A*B;
  return x;
}
armMul(10,20);
```

E

List of Publications

During the course of this research, the following publications were produced.

1. **Frieslaar, I. and Irwin, B**. An investigation into the signals leakage from a smartcard based on different runtime code. *In Southern Africa Telecommunication Networks and Applications Conference (SATNAC), pages 279284, 2015. ISBN 279-284978-0-620-67151-4.*

2. **Frieslaar, I. and Irwin, B.** Towards a software approach to mitigate correlation power analysis. *In Proceedings of the 13th International Joint Conference on e-Business and Telecommunications - Volume 4: SECRYPT, (ICETE 2016), pages 403410. INSTICC, SciTePress, 2016a. ISBN 978-989-758-196-0. doi:10.5220/0005955604030410.*

3. **Frieslaar, I. and Irwin, B.** Evaluating the multi-threading countermeasure. *International Journal of Computer Science and Information Security, 14(12):379–387, 2016b. ISSN 1947 5500.*

4. **Frieslaar, I. and Irwin, B.** Investigating multithread utilization as a software defence mechanism against side channel attacks. *In Proceedings of the 8th International Conference on Signal Processing Systems, ICSPS 2016, pages 189–193, New York, NY, USA, 2016c. ACM. ISBN 978-1-4503-4790-7. doi:10.1145/3015166.3015176.*

5. **Frieslaar, I. and Irwin, B.** A multi-threading approach to secure VERIFYPIN. *In 2016 2nd International Conference on Frontiers of Signal Processing (ICFSP), pages 3237, Oct 2016d. ISBN 978-1-5090-3815-2. doi: 10.1109/ICFSP.2016.7802952.*

6. **Frieslaar, I. and Irwin, B.** Investigating the electromagnetic leakage from a Raspberry Pi. *In 2017 Information Security South Africa (ISSA), August, 2017, IEEE, ISBN 978-1-5386-0544-8*

7. **Frieslaar, I. and Irwin, B.** Recovering AES-128 encryption keys from a Raspberry Pi. *In Southern Africa Telecommunication Networks and Applications Conference (SATNAC), pages 228235, September 2017, ISBN 978-0-620-76756-9*

8. **Frieslaar, I. and Irwin, B.** Investigating the effects various compilers have on the electromagnetic signature of a cryptographic executable. *In Proceedings of the South African Institute of Computer Scientists and Information Technologists (SAICSIT 2017), SACSIT 2017, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5250-5/17/09. doi: 10.1145/3129416.3129436.*

9. **Frieslaar, I. and Irwin, B.** Investigating the utilization of the secure hash algorithm to generate electromagnetic noise. *In Proceedings of the 9th International Conference on Signal Processing Systems, ICSPS 2017, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-5384-7.*

10. **Frieslaar, I. and Irwin, B.** Developing an Electromagnetic Noise Generator to Protect a Rasberry Pi from Side Channel Analysis *SAIEE Africa Research Journal, Volume: 109, pages:85–101, 2018. ISSN 1991-1696*

F

Datasets

The data sets for this research is stored by the Open Science Framework https://osf.io/mte5q. The data set is free to use and can be modified without consent of the author. The *bibtex* reference is given below:

```
@misc{Frieslaar_2017,
    title={Electromagnetic Data from a Raspberry Pi 2},
    url={osf.io/mte5q},
    publisher={Open Science Framework},
    author={Frieslaar, I and Irwin, B.},
    year={2017},
    month={Aug},
    doi ={10.17605/OSF.IO/MTE5Q}
}
```