# DEVELOPING AN ELECTROMAGNETIC NOISE GENERATOR TO PROTECT A RASPBERRY PI FROM SIDE CHANNEL ANALYSIS

I. Frieslaar and B. Irwin*

* Rhodes University, Department of Computer Science, Grahamstown, South Africa E-mail: ebie099@gmail.com and b.irwin@ru.ac.za

Abstract: This research investigates the Electromagnetic (EM) side channel leakage of a Raspberry Pi 2 B+. An evaluation is performed on the EM leakage as the device executes the AES-128 cryptographic algorithm contained in the libcrypto++ library in a threaded environment. Four multi-threaded implementations are evaluated. These implementations are Portable Operating System Interface Threads, C++11 threads, Threading Building Blocks, and OpenMP threads. It is demonstrated that the various thread techniques have distinct variations in frequency and shape as EM emanations are leaked from the Raspberry Pi. It is demonstrated that the AES-128 cryptographic implementation within the libcrypto++ library on a Raspberry Pi is vulnerable to Side Channel Analysis (SCA) attacks. The cryptographic process was seen visibly within the EM spectrum and the data for this process was extracted where digital filtering techniques was applied to the signal. The resultant data was utilised in the Differential Electromagnetic Analysis (DEMA) attack and the results revealed 16 sub-keys that are required to recover the full AES-128 secret key. Based on this discovery, this research introduced a multi-threading approach with the utilisation of Secure Hash Algorithm (SHA) to serve as a software based countermeasure to mitigate SCA attacks. The proposed countermeasure known as the FRIES noise generator executed as a Daemon and generated EM noise that was able to hide the cryptographic implementations and prevent the DEMA attack and other statistical analysis.

Key words: AES-128, Electromagnetic, Noise Generator, Raspberry Pi, Side Channel Analysis, SHA, Software Countermeasure.

## 1. INTRODUCTION

The volume of data that is produced daily is growing exponentially. Much of this is potentially sensitive, and the challenging aspect relating to this is the security of this information both at the time of use and longer term. The growth of the Internet of Things (IoT) in parallel has further contributed to the amount of data produced. These IoT devices are not only situated in the homes, but in businesses, manufacturing and as part of smart cities. Examples of these interconnected systems are smart grids, where the distribution of electricity [1], natural gas [2] and water are controlled by these IoT devices in real time. In addition, our homes have many sensors that are increasingly being utilised to monitor regular domestic conditions [3] and, recently, companies have introduced digital living assistants that have access to an individuals smartphone and other interconnected devices to assist the user. Although these devices contribute to the ease of daily activities, they pose a potential security threat.

The vulnerabilities of such IoT devices have been exploited. An example is the 2016 incident where IoT units were used to carry out a Distributed Denial of Service (DDoS) attacks on DynDNS which resulted in major sites and services being inaccessible to billions of people worldwide [4]. These IoT devices are however, not the only devices at risks. Industrial Programmable Logic Controllers (PLCs) are among other connected devices at risk [5, 6]. 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 (such as in the case of Stuxnet [7]), becomes a target and are potentially vulnerable if not correctly secured.

To ensure information is secured, cryptographic algorithms such as the Advance Encryption Standard (AES) have been adopted and are widely used [8]. These algorithms conceal vital information from potential eavesdroppers and are in theory mathematically secured. However, it has been demonstrated extensively that the implementation of AES is vulnerable to Side Channel Analysis (SCA) attacks [9, 10]. SCA exploits the power consumption or electromagnetic (EM) emissions to retrieve secret information by determining a correlation between the intermediate values and the power/electromagnetic consumption [11], which enables an adversary to recover encryption keys.

This research aims to investigate the susceptibility of the AES-128 cryptographic algorithm implementation on a Raspberry Pi, as EM radiation is leaked from the device during operation. The device under test is a Raspberry Pi 2 B+ model. This

Based on: "Investigating the Electromagnetic Leakage from a Raspberry Pi", by I. Frieslaar and B. Irwin which appeared in the Proceedings of Information Security South African (ISSA) 2017, Johannesburg, 16 & 17 August 2017. 

device was selected, as it these boards are readily available, widely used and on occasions used as PLCs [12]. This research implements and investigates the effect of multi-threads to purposely leak out obfuscated information at critical points alongside the AES-128 algorithm. Furthermore, there are still many unanswered questions as to the EM leakage from a Raspberry Pi and this study would contribute to the research field by answering the following questions:

1. How exposed is the libcrypto++ [13] implementation of AES on a Raspberry Pi to side channel attacks ?
2. Would various multi-thread libraries behave differently in terms of EM emanations ?
3. Can multi-threading be utilized as a software based countermeasure to SCA attacks ?
4. What effect would a multi-threaded software based countermeasure have on the EM emanations ?

In addition to the above questions, this research will investigate the implementation of an EM noise generator to obfuscate SCA analysis and mitigate the recovery of secret information from EM emissions.

The implementation of an EM noise generator is 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 redesigning hardware is expensive.

The remainder of this paper is organized as follows: Section 2. discusses the impact and dangers of stolen encryption keys; Section 3. details the research carried out in the field of SCA attacks against high frequency devices; Sections 4. and 5. details potential techniques and approaches that are not commonly utilised as a software based countermeasure, but which can potentially serve as a countermeasure; Section 6. discusses how this research aims to capture, transform and utilise EM emissions from a Raspberry Pi to obtain useful information; Section 7. elaborates on the initial experiment and results with regard to the EM leakage from the various multi-threading libraries and a proposed software countermeasure; Section 8. demonstrates the recovery of the AES-128 secret key. This is followed by Section 9. that introduces a novel EM noise generator to prevent the recovery of the secret key. Attacks are performed against the novel EM noise generator in Section 10. Finally the paper is concluded with a discussion in Section 11.

## 2. IMPACT OF STOLEN ENCRYPTION KEYS

The impact of obtaining symmetric cryptographic or private keys could lead to a major catastrophe for industrial control hardware and federal systems. 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 evident when the Iranian nuclear program had been sabotaged by stolen private keys in the Stuxnet attack [7]. The 2014 cyberattack on a German steel-mill [14] where critical infrastructure was destroyed is another example of a comprised system. Once the adversaries gained access to the system, they proceeded by taking control of the production management software of the steel mill and caused serious damage to the infrastructure.

The use of cyberweapons to gain an advantage on the battlefield or against foreign entities is nothing new [15, 16]. However, using these weapons on civilian targets such as power grids, traffic lights, hospitals and train stations is troubling and a compromise would lead to financial and socio-economic disaster. The possibilities of a compromised system are endless. A few scenarios would be that the attacker takes control over an insulin pump, where it is possible to distribute incorrect insulin doses, and produce a slow and painful death to an individual. Additionally, attackers have the ability to compromise and take control of a motor vehicle [17] by either disrupting or intercept electromagnetic signals to allow for malicious actions against the electronic control unit of the vehicle, which can lead to serious fatalities [18].

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 [19]. The decrypted signal can then be used and distributed to others without a subscription fee, thus resulting in industry losing revenue. Furthermore, cellphone calls could also be exposed and leaked to various third parties if the encryption keys were stolen.

In 2015 the hack into Gemalto [20], in which billions of mobile SIM cards cryptographic keys were stolen, demonstrated both the importance and impact of stolen keys. As a result of 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.

Consumer entertainment services such as pay-per-view and video streaming would be susceptible as the attacker could pretend to be the streamer and make millions of dollars. 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 (and other cryptocurrency) transactions on a regular basis. Consequently it is therefore fundamentally important to ensure that all devices are protected.

The deployment and development of cyberweapons have increased drastically in the last decade. This has lead to cyber threats and attacks becoming more common, sophisticated and damaging. Yearly there are discussions on the security downsides and risks of an Internet–connected world [21, 22]. Recently Symantec [23] published a report detailing that the power grids around the world have been infiltrated by adversaries to firstly steal critical information such as technical diagrams, reports, passwords, cryptographic keys. and secondly causing mass destruction by destroying infrastructure via code. It is becoming extremely important and critical that all infrastructure be protected. Therefore, it is imperative to secure all avenues that could possibly lead to the recovery of cryptographic keys.

## 3. ELECTROMAGNETIC ATTACKS

Hitherto the research community has focused on SCA attacks based on smartcards, RFID tags, FPGAs and microcontrollers, particularly by analysing electromagnetic (EM) emissions [11, 24, 25]. In more recent years the research community began to target the vulnerabilities of high powered devices such as laptops and smartphones to SCA attacks [26–28]. The focus of this research is on the Raspberry Pi 2B+, which is a higher frequency device as mentioned in Section 1., that has been selected as it is readily available, widely used and on occasions used in the role of a PLCs [12].

EM emanations are captured from devices and subsequently used to retrieve secret information. EM measurements do not require the attacker to have direct contact with the device and are able to extract information without the user having knowledge of the attack. EM Attacks are therefore less intrusive than the conventional attacks via power analysis [11]. The remainder of this section will focus only on the attacks against high powered devices, such as smartphones and systems typically utilizing ARM family processors.

Aboulkassimi et al. [29] 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. Furthermore, Goller and Sigl [30] implemented attacks on a public key cryptography algorithm against an Android smartphone executing RSA. Additionally, the smartphone's shielding plate was removed.

Nakano et al. [31] attacked an Android smartphone using low frequency attacks. The smartphone ran at 832 MHz. Their attacks focused on the RSA and Elliptic curve cryptography (ECC) encryption implementations contained in the Java Cryptography Extension (JCE). Additionally, they removed the battery and metal covers in order to recover EM data from the smartphone. However, this was not mentioned in their work. Upon further analysis, the graphic provided in their paper, demonstrates visibly that modification had been made to the rear of the smartphone. Furthermore, no mention is made of the device specifications.

Belgarric et al. [27] and Genkin et al. [28] concurrently demonstrated successful attacks on the Elliptic Curve Digital Signature Algorithm (ECDSA) implementation of Android's BouncyCastle library. A minor difference found in the two studies was that the work of Belgarric et al. was intrusive as they placed the magnetic probe inside the smartphone where as Genkin et al. was non-intrusive and placed the magnetic probe in close proximity of the device. Genkin et al. demonstrated a successful secret key recovery of the ECDSA signing keys from OpenSSL executing on an iOS devices and partial keys from an Android device. Furthermore, they were able to recover secret keys from Corebitcoin off an iOS device.

Balasch et al. [32] performed a successful Differential Power Analysis (DPA) attack against the bitsliced AES encryption algorithm executing on a BeagleBone Black ARM development board. An ARM Cortex-A8 processor running at 1 GHz was used. A similar attack against the device was performed by Galea et al. [33]. The results obtained demonstrate the vulnerabilities of symmetric key encryption executing on high powered devices to SCA attacks. It is noted that in both studies the EM probe was physically glued onto the area of leakage and focused on specialised hardware with hardware countermeasures proposed.

As discussed in this section, high powered devices are vulnerable to SCA attacks. This is a fairly new research avenue which is evident by the minimal related work in the field. In addition, the research in [34] recently demonstrated the ability to recover partial (12 out of the 16 subkeys) AES-128 cryptographic keys from a Raspberry Pi. As it is a new field the research [27–34] has followed

attacks introduced by targeting smartcards, RFID tags, FPGAs and microcontrollers. However, each attack against these higher frequency devices had different approaches with regards to recovery EM information and applying digital filtering techniques to recover sensitive information.

Many cryptographic algorithms and high frequency devices have not been put under scrutiny against SCA attacks. This distinctly ties into the questions posed in Section 1. This novel research will therefore be the first of its kind to investigate the EM effects of a symmetric encryption algorithm as it is executed on a Raspberry Pi in a multi-threaded environment and utilising multi-threads as a countermeasure on high frequency devices.

## 4. MULTI–THREADING AS A SOFTWARE COUNTERMEASURE

This section will discuss multi-threading techniques and approaches that this research has identified, but which are commonly utilised to increase the run-time execution of a program as a potential solution for a software based countermeasures to mitigate SCA attacks. This research has selected a multi-threading approach as it has been demonstrated in previous work that a multi-threading solution outperforms known software based countermeasures [25, 35, 36].

In terms of EM emissions and data leakage, it is unknown what effects each multi-threading approach could potentially have. Therefore, this research considers it a requirement to investigate the effects that each multi-threading approach would potentially have on the EM field. The experiments and results can be found in Section 7.

Multi-threading has been widely utilised to speed up run time operations of an executable. However, this research explores the use of multi-threading as a possible software based countermeasure. Four multi-threading Application Program Interface (API)s will be used on the Raspberry Pi. The four implementations are Portable Operating System Interface (POSIX) Threads or more formally referred to as pThreads[*], C++11 multi-threads[**], Thread Building Blocks (TBB)[***] and Open Multi-Processing (OpenMP)[†].

Multi-threading is the process of executing multiple threads or instructions concurrently. These threads are managed independently by a task scheduler [37]. Normally, one process can consists 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 [38]. 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 system via improved utilisation of CPU cores, decreasing the runtime execution of a program [39]. 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.

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.

Developers utilise various libraries to develop their code in a multi-threaded framework. The most common framework in the UNIX environment is the pThreads [40]. Multi threading libraries provide developers access to creating applications with multi-threaded support. These libraries aslo provide developers with the option to create a synchronized environment between the threads by using mutexes, condition variables, semaphores, monitors and other synchronization primitives [41].

### 4.1 POSIX

pThreads, 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 [42].

### 4.2 C++ Threads

The C++11 thread library is more of a memory model approach that supports multi-threading [43]. C++11 threads are fully incorporated into C++ as a language, so that there is no need to allocate arguments in a form of a struct. In addition, C++11 multi-threaded library allows for numerous arguments to be passed to a function that has a thread.

### 4.3 Threading Building Blocks

The Intel TBB [44] 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

[*] pThreads – https://goo.gl/EmNDc6
[**] C++11 – https://goo.gl/wdev41
[***] TBB – https://goo.gl/xpNLFE
[†] OpenMP – www.openmp.org

the developer to create portable code that can execute on different OS architectures. Furthermore, TBB has been built upon pThreads as the underlying threading API.

### 4.4 OpenMp

OpenMP is a directive based extension to C/C++ and Fortran as it supports data and task parallelism on shared memory architectures [45]. As opposed to TBB, OpenMP is a language extension that requires an OpenMP-enabled compiler. OpenMP has support from various compilers except Clang. In addition, OpenMP is build on pThreads.

## 5. HASH FUNCTIONS

This research has selected cryptographic hash functions as a candidate for the software based countermeasure to mitigate SCA. It will be empirically demonstrated in Sections 7. – 10. that the hash function outperforms the other candidates thus, this section will discusses the basic concepts of a hash functions and the intended hash algorithms to be utilized within this research as a software based countermeasure. This section will discusses the basic concepts of a hash functions and the intended hash algorithms to be evaluated within this research as a software based countermeasure.

A cryptographic hash function takes a string of various length as input and converts it to a fixed output of bytes. The hash function is regarded to be extremely flexible as it can be utilised in many schemes such as encryption, authentication and even as a digital signature [46]. Hash function are generally utilised in cohesion with encryption algorithms to increase the security of a system.

A cryptographically secure hash function is regarded as a one-way function as the output of the hash cannot be reversed to determine the original input message [47]. In addition, hash functions should be resilient to collision attacks, where two different messages cannot have the same hash output.

This research focusess on the Secure Hash Algorithm (SHA) family, more specifically the SHA–1 and SHA–2 algorithms. The SHA family of cryptographic functions are published by the National Institute of Standards and Technology (NIST) [48]. SHA–3 also forms part of the published family. The specific algorithm used in this is known as Keccak.[††] However, this research will not consider SHA–3 as it differs substantially from SHA–1 and SHA–2 in its internal operation.

SHA converts plain text into secure information by utilising a cryptographic hash function. A cryptographic hash function is a mathematical algorithm consisting of bitwise operations, modular additions, and compression functions which transforms the input message to a bit string of a fixed size [49].

Each iteration of the SHA–1 and SHA–2 algorithms was designed to increase the security of the algorithm and prevent attacks that the previous version was vulnerable to. The core difference between these algorithms of interest are depicted in Table 1. 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.

Table 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 | | |

Until now, this research has been the first to design and develop a software based countermeasure to mitigate SCA by utilising the SHA family hashes to generate EM noise. 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.

## 6. METHODOLOGY

This section elaborates on the equipment utilised in this research to capture the EM emanations and the process of collecting and analysing the EM data into meaningful information.

### 6.1 Equipment

This research utilised two Raspberry Pi's. The first device served as the victim, while the secondary device served as the adversary. The Lubuntu 14.04 [‡] Operating System, with the Linux 3.18.0-20-rpi2 kernel[‡‡] was utilised. It is to be noted that no services in the stock Operating System were disabled. In addition, 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. This research followed the approach by [32] and [28] to limit the CPU frequency. It is noted has built in time delays and interrupts as a countermeasure. Furthermore, no adjustments were made to the secondary device. The FUNcube dongle was inserted into a USB port and GNURadio was used to interface with the device. Figure 1 illustrates the experimental setup.

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

[‡] Lubuntu 14.04 – https://goo.gl/sUutNh

[‡‡] Linux 3.18.0-20-rpi2 kernel – https://goo.gl/5azpV5

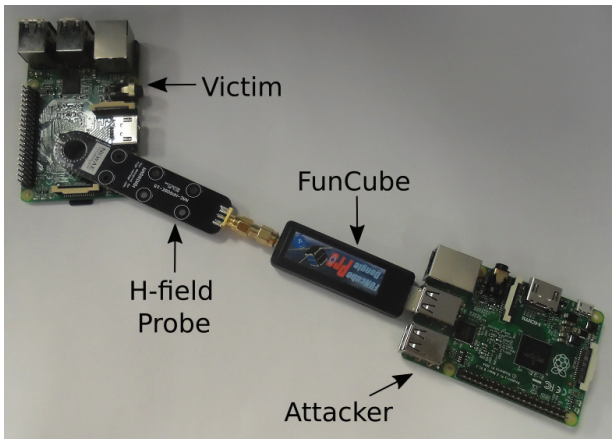



Figure 1: The experimental setup.

In order to interface with the SDR and perform digital signal processing, GNURadio 3.7.9[§]and Baudline 1.0.8[¶] were utilised.

GNU Radio is a free software development toolkit that provides signal processing blocks to implement software defined radios and signal processing systems. In addition, 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.

### 6.2 Data Analysis

There are three stages at which the EM data can be analysed [31]. Stage one is to compute 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 2 illustrates the raw signal after it had been processed through a FFT within GNURadio. The signal is obtained by monitoring the desired 600 MHz frequency in real time via the SDR.

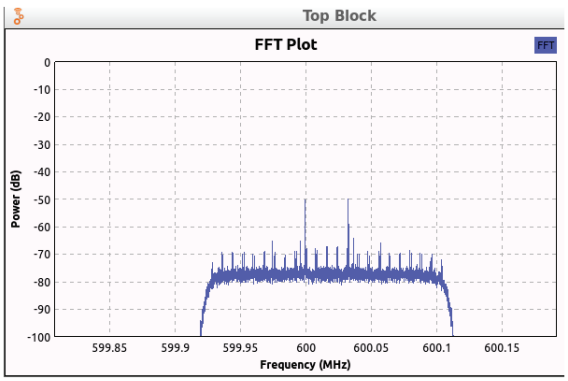



Figure 2: The signal as it is displayed by the FFT.

§ GNURadio – https://www.gnuradio.org/
¶ Baudline – http://www.baudline.com/

Stage two consists of selecting the region of interest at the point of EM leakage, i.e: when a specific operation executes and can be seen in the radio spectrum. This is performed by visual analysis and can be seen in the amplitude domain in Figure 3 by a rectangle encapsulating the peak signal. This informs the adversary of the location of execution in time and the type of signature that is produced.

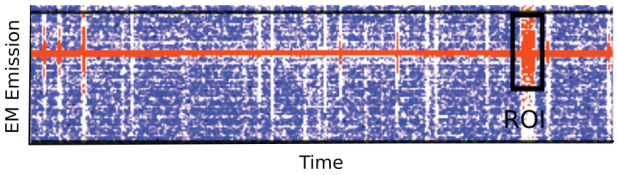
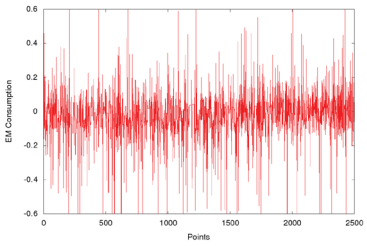
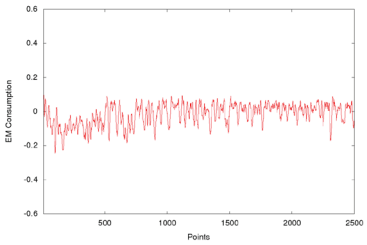



Figure 3: The region of interest.

Finally, the third stage is to apply digital filtering and noise removal techniques (de-noising) to filter unwanted information in the signal. Figure 4a illustrates the signal after digital filtering has been applied and Figure 4b depicts the signal after a de-noising method has been applied to the signal.




(a) Original signal after filtering.




(b) Signal after digital processing

Figure 4: The original (a) and (b) the signal after processing.

While the victim executes various test algorithms, the adversary uses GNURadio to capture the EM emissions from the device at 384 kHz. The FUNcube dongle has a range of features with regard to the input bandwidth, ranging from 44.1 kHz – 384 kHz. Sampling at a higher frequency allows for more data to be obtained. Therefore, each recording is captured at 384kHz. Digital filtering was applied to keep all signal information between 0 – 50 kHz, while the rest of the signal frequency was discarded. The resultant signal was sent to Baudline where the region of interest was extracted and sent back to GNUradio, where

Quadratic demodulating [50] was applied. This was followed by an additional low pass filter.

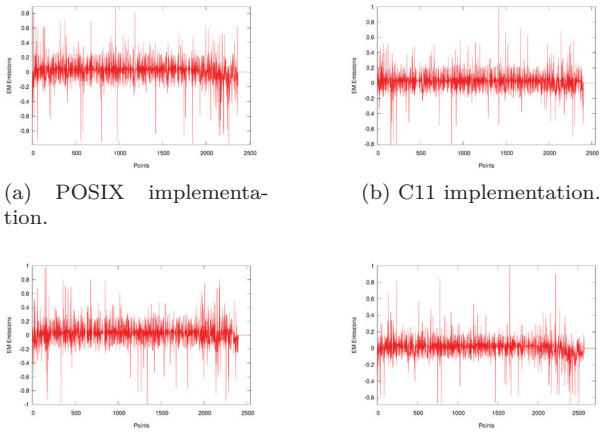
Since the research is dealing with digital signals and it is known that information is carried within the carrier signal [50], demodulation is applied to recover the information within the signal, specifically, quadratic demodulating.

## 7. EXPERIMENTS, RESULTS, AND ANALYSIS

This section discusses the software countermeasure investigated to mitigate the SCA attacks against the Raspberry Pi. The proposed approach uses multi-threads to purposely leak out obfuscated information while the AES-128 algorithm executes. These APIs are pThreads; C++11 multi-threads, henceforth referred to as C11 threads; TBB and OpenMP threads, as discussed in Section 4. In addition, Section 7.1 discusses the findings with regard to the EM leakage as the AES-128 encryption was executed, followed by Section 7.2 elaborating on the experiments as noise was introduced.

### 7.1 EM Leakage

The first set of experiments consisted of the AES-128 algorithm encapsulated into the four multi-threaded APIs. One thread executed the cryptographic algorithm on the four different platforms. No additional threads were utilised. While the programs were running, the EM emanations were captured alongside the time per execution.



(a) POSIX implementation. (b) C11 implementation.

(c) OpenMP implementation. (d) TBB implementation.

Figure 5: The EM leakage of the four cryptographic threaded implementations.

Figure 5 illustrates the EM leakage of the cryptographic threaded implementations as one thread was used for the various multi-threading APIs. It is observed that all four outputs have a similar EM signature. A better visual description can be seen in Figure 6 as the four implementations are overlaid onto one figure. The results are as expected, as only one thread was executing the cryptographic algorithm.

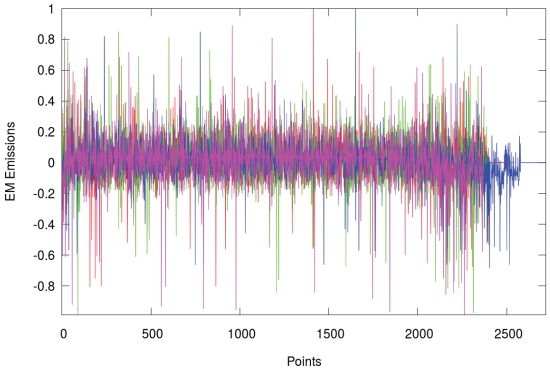



Figure 6: A combination of the EM leakage of the four cryptographic threaded implementations as one thread was used

It is seen in Figures 5d and 6 that the TBB implementation ends at a later stage. Even though the other EM signatures have are similar EM profile to that of the TBB implementation. The TBB has a larger output, i.e. TBB ends above 2500 data points where as the other implementations ends before 2500 data points. This is associated with the fact that TBB takes longer to execute the code. Additional data will be provided later on in this section to demonstrate the findings.

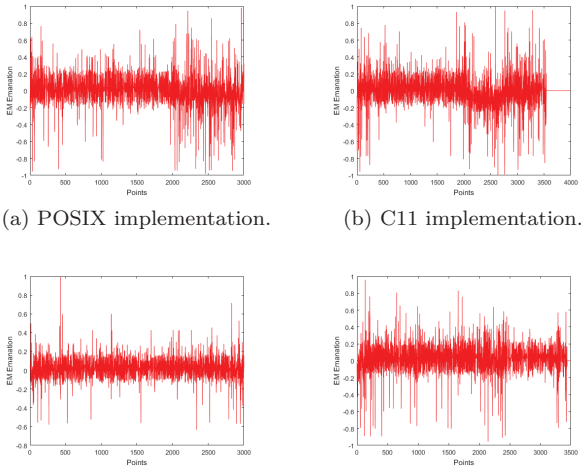
The second set of experiments consisted of the AES-128 algorithm encapsulated into the four threaded APIs with a secondary thread calculating the first 1000 Prime numbers and is depicted in Figure 7. The two threads executed in parallel with no synchronization and no thread had priority over the other. The EM emissions of the experiment is depicted in Figure 7.

The first notable finding is that the data points have increased which is due to the additional thread. In addition, the TBB emissions in Figure 7d have increased by 1000 data points. Each implementation has a noticeably different EM signature.

The third set of experiments consisted of the AES-128 algorithm encapsulated into the four threaded APIs with a secondary thread calculating the first 1000 Fibonacci numbers and is displayed in Figure 8.

Observing Figures 8a–8d, it is shown that each implementation of the four thread techniques have different power patterns. The pThread implantation in Figure 8a illustrates that it takes the fewest number of points as compared to the rest of Figures 8b – 8d.

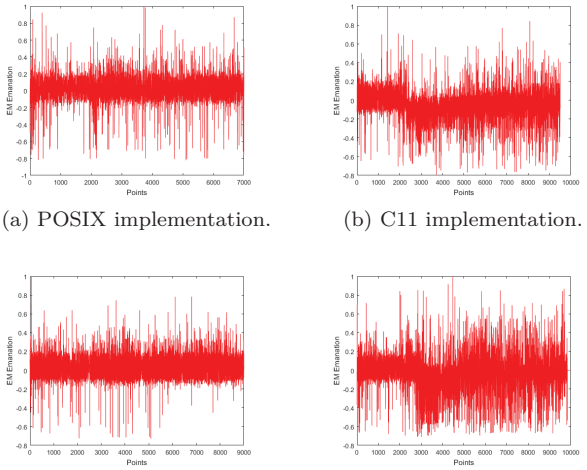
Additional analysis was performed. However, for this occasion only one multi-threaded API was




(a) POSIX implementation. (b) C11 implementation.

(c) OpenMP implementation. (d) TBB implementation.

Figure 7: The EM leakage with an additional thread calculating prime numbers.




(a) POSIX implementation. (b) C11 implementation.

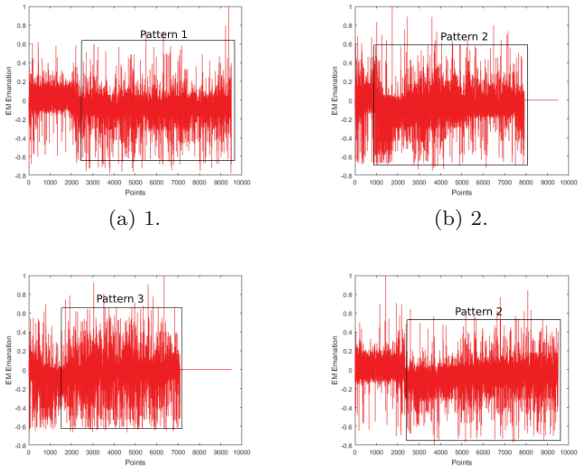
(c) OpenMP implementation. (d) TBB implementation.

Figure 8: The EM leakage with an additional Fibonacci thread.

selected. Multiple instances of calculating the Fibonacci sequence analogised the AES-128 execution was performed and the EM data was captured. Figures 9a – 9d depict that there are three distinct patterns and only Figure 9b and 9d have similar patterns. Figures 9a and 9c have a different pattern and end at a different location in time.

Although, the same sequence of operations was performed, different EM patterns were being recorded. The research noted that this was due to the Raspberry Pi's CPU and power management setup [51]. More information relating to this will be discussed later in this section.

The data presented in the results continue by illustrating the average recorded execution times for




(a) 1. (b) 2.

(c) 3. (d) 4.

Figure 9: Various EM patterns.

the four thread techniques using the various programs in Table 2.

Table 2: The average run time using the four different multi-thread techniques.

| | Threaded-Crypto | Crypto+ Prime Threaded | Crypto+ Fibonacci Threaded |
|---|---|---|---|
| C11 | 2,779 | 16,075 | 152,025 |
| OpenMP | 2,449 | 15,586 | 129,32 |
| pThreads | 0,593 | 13,826 | 117,252 |
| TBB | 6,377 | 31,602 | 151,316 |

Time in milliseconds.

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. The time is displayed in milliseconds.

The data in Table 2 displays that the pThreads implementation was the fastest in all scenarios. This is further evident when comparing the EM emissions of the Fibonacci implementations in Figure 8 as it can be seen that the pThreads implementation has fewer data points over time. The TBB implementation is by far the slowest. Additionally, when computing the Fibonacci sequence the C11 is slower than the TBB.

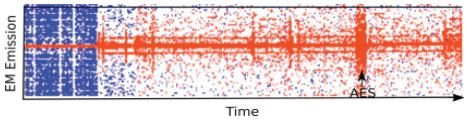
Based on the data over all the graphs, in certain scenarios there are specific patterns. The CPU cores were monitored and the analysis indicated that the built in power management uses different cores on each execution of a program. The multi-core processors distribute resources to other existing cores in order to complete the task effectively. During the cryptographic operation, the process was moved

from one core to another due to built in system interruptions and power management. It can also be executed in parallel by several cores for performance.

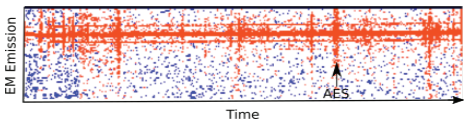
It is noted that all programs were compiled with the GCC compiler and the -O0 command. With -O, the compiler attempts to reduce code size and execution time, without performing any optimizations that reduce compilation time. The research in [52] details work done regarding to EM leakage as different compiler optimizations are utilised.

### 7.2 Generating Noise

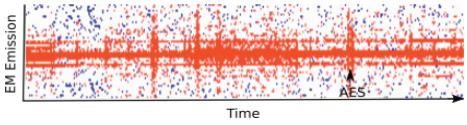
The upcoming experiments will focus on the implementation of a noise generator program to execute in the background on a different thread as a daemon while the cryptographic algorithm executes. This daemon generated noise will be referred to as FRIES noise. The FRIES noise consisted of executing various arithmetic instructions in an infinite loop. These operations consisted of calculating the Fibonacci sequence, prime numbers, and other arithmetic instructions [36]. The Daemon approach has been selected as it is this research desire to develop a countermeasure that does not modify the existing cryptographic algorithm, thus it can be implemented with other application such as banking and various other financial applications.



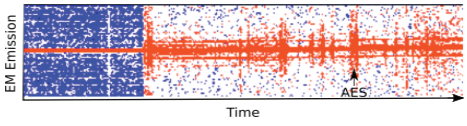

(a) POSIX implementation.




(b) C11 implementation.




(c) OpenMP implementation.




(d) TBB implementation.

Figure 10: The EM leakage with a noise generator.

Figure 10 illustrates the time domain as the four multi-threaded implementations of the cryptographic algorithm are executed while the noise program is running in the background.

The annotated arrow in the Figures 10a–10d illustrates at which point the cryptographic algorithm took place. This indicates it is still possible to detect the location of execution of the cryptographic algorithm in the second stage as discussed in Section 6. although, on 32 occasions out of 50 runs, the cryptographic execution was not detected.

Further analysis was carried out and it was determined as the prime numbers was executing at the same time of the cryptographic thread, the cryptographic execution was hidden. A secondary set of experiments was carried out. This entailed calculating N number of prime numbers randomly in an infinite loop while the cryptographic code executed. Figure 11 indicates the sequence as the prime numbers were randomly calculated alongside the cryptographic algorithm.

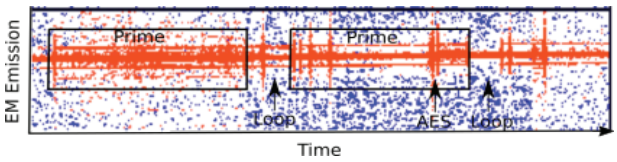



Figure 11: EM leakage as FRIES was executing.

The first rectangle represents the EM emanation as the prime numbers were calculated. In addition, the AES algorithm was executed within that time frame and it cannot be explicitly observed in the plot. 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 is visible. The analysis indicated that when a low number of primes is calculated there is a window in the noise, which results in the where the cryptographic algorithm and for-loops can be seen visibly as depicted by the annotations in Figure 11.

Having established a range to execute the prime numbers while the cryptographic algorithm is executing, the third experiment involved calculating prime numbers between 100 and 1000 indefinitely. Figure 12 displays the EM emanations after demodulation has been applied. The waveform in the figure is repeated over time and it becomes extremely difficult to pin-point the exact location of the execution of the AES-128 algorithm.

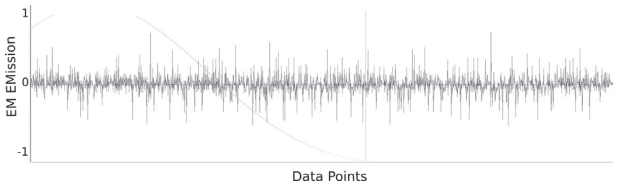
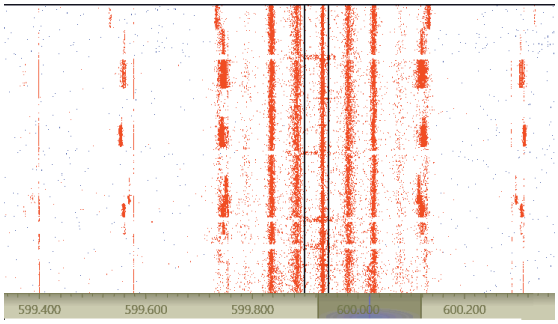



Figure 12: The EM leakage depicting a repeatable waveform.

Additional data has been identified by monitoring

the EM emanations. As more power is used by the Raspberry Pi, more leakages appear across the spectrum. This is due to voltage harmonics. It is further evident in Figure 13a that as more power and/or cores are used by the CPU more data is available across the spectrum as illustrated in Figure 13b.




(a) Amplitude domain.


(b) Htop displaying core usage.

Figure 13: The leakage (a) across the spectrum and (b) the core usage from the device.

## 8. RECOVERING SECRET INFORMATION

Research from Frieslaar and Irwin [34] has recently demonstrated the capabilities of recovering partial AES-128 cryptographic subkeys from a Raspberry Pi. The research was able to retrieve 12 of the 16 subkeys. However, this research aims to build on the previous research by recovering the full encryption key, i.e: all 16 subkeys.

The same capturing procedure will be utilised as mentioned in [34]. Therefore, in order to acquire the EM data from the Raspberry Pi, the same software configuration was used and the procedure is as follows:

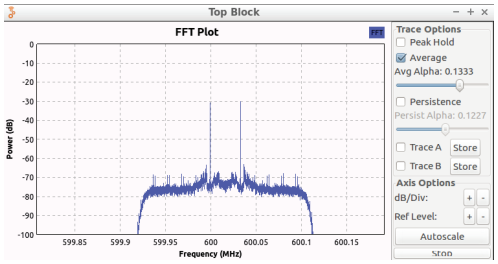
1. While the victim's device executes the cryptographic algorithm, GNURadio is utilised to capture raw signals.
2. The raw signal is sent to a Fast Fourier Transform (FFT) process.
3. Digital filtering techniques such as low and high pass filters are applied to the signal.
4. The region of interest is extracted from the filtered signal with Baudline.
5. Quadratic demodulation is applied to the region of interest.

The resultant signal will be passed to a prepossessing procedure where various alignment techniques will be applied to the signal. This research follows the process mentioned by [34] to align the data. The process is as follows:

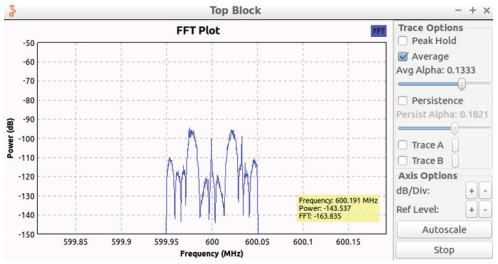
1. The signal is segmented into three partitions.
2. Elastic alignment [53] is applied to each partition.
3. The aligned segmented partitions are rejoined.
4. Resyncing techniques, such as peak detection and sum of absolute difference are applied.
5. The Savitzky-Golay [54] filter is used to increase the signal to noise ratio.

Once the data has been aligned, the resultant data will be sent to the Correlation Power Analysis (CPA) [55] attack. The CPA attack is utilised to recover the secret key used by the AES-128 algorithm with the EM emissions as input data.

The original research had kept all signals ranging from 0–50 kHz within the captured signal. This research proposes the removal of certain frequencies. The frequencies between 0–15 kHz will be removed and the resultant signal will be sent to the attack procedure as input for the CPA attack. Figure 14 illustrates a comparison between the original signal and the signal with the 0 – 15 kHz frequency removed.




(a) Original Signal




(b) Omitted 0–15 kHz.

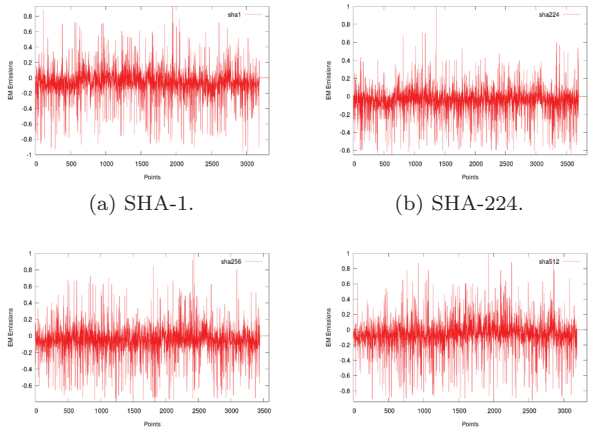
Figure 14: The FFT with the (a) original and (b) modified signal.

Following the removal of the frequencies between 0 –15 kHz, this research was successfully able to recover the entire secret key used for the AES-128 cryptographic algorithm. This is a significant improvement over the previous work where only 12 subkeys were recovered. In addition, only 50 traces were required as input data. The results of the CPA attack is depicted in Table 3.

The results indicate that utilising 10 traces, five subkeys were recovered, eventually as 50 traces were utilised the entire subkey was recovered. It is further observed that subkeys 2, 4, 6, 8, and 12 were recovered in all the subsets and only subkeys 4, 10, 15 and 16 were recovered when 50 traces were used.

## 9. IMPROVING THE COUNTERMEASURE

The prime number sequence within the FRIES noise generator was replaced with the SHA from the libcrypto++ library. Firstly, an EM signature profile was created by capturing the various implementations off the SHA as they executed on a Raspberry Pi. These implementations are SHA-1, SHA-224, SHA-256, and SHA512 functions as discussed in Section 5. The capturing of the EM emissions remained the same as discussed in Section 6.




(a) SHA-1. (b) SHA-224.

(c) SHA-256. (d) SHA-512.

Figure 15: The EM leakage for the various SHA functions.

Figure 15 depicts the EM leakage from the Raspberry Pi as the various SHA functions were executed within libcrypto++. The EM signature patterns changed as different hash function were utilised, more specifically, that of the SHA-512 function as seen in Figure 15d.

The SHA functions are based on strict cryptographic principles as opposed to the calculation of prime numbers as discussed in Section 5. Therefore, the FRIES noise generator was modified to execute the SHA-512 hash function in an infinite loop and run in the background as a Daemon. The new process of the FRIES noise generator is depicted in Figure 16.

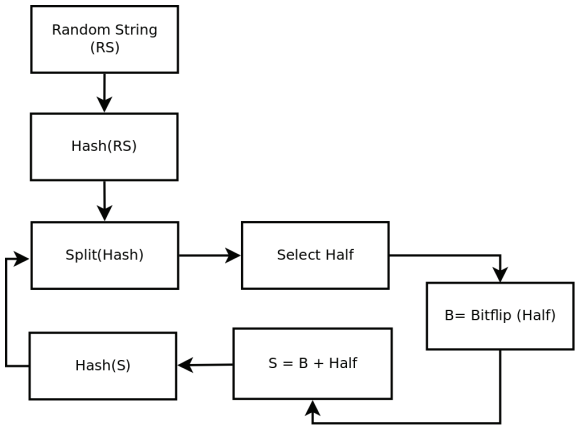



Figure 16: Genearting random hashes for the FRIES noise generator.

The process commences by generating a random alphanumeric string as input for the SHA-512 hash function. The resultant hash value was segmented into two and within each loop a decision was determined randomly whether to take either the upper or lower half of the segmented hash value. A bit-flip was applied to the segmented hash value and appended to the previous half that was not selected in the previous step. The resultant string was utilised as input for the SHA-512 hash function. This process was repeated indefinitely, thus increasing the entropy and generating random secure hashes.

While the FRIES noise generator was running in the background, the AES-128 algorithm was executed over time. After the noise generator had calculated 40 hashes, the CPU of the Raspberry Pi had reached 100% usage on all four cores. This resulted in the device executing the FRIES noise generator in a slow down state. While the device was in this state the AES-128 algorithm could be clearly observed as indicated in Figure 17 by the annotations.

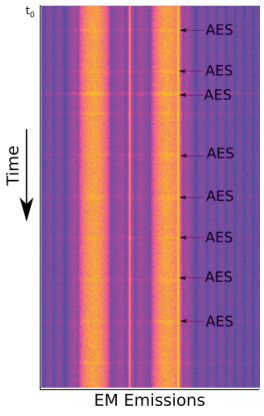


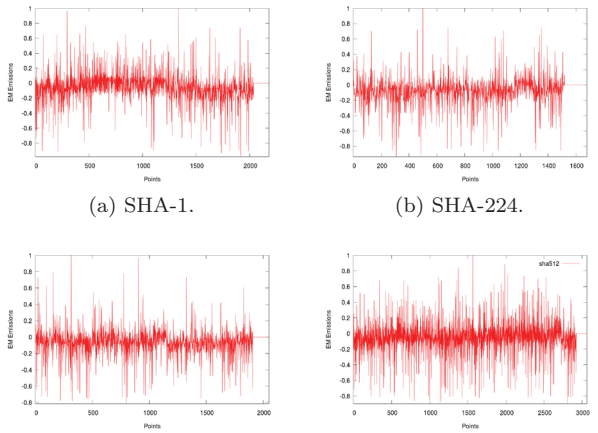

Figure 17: The runtime of the AES-126 algorithm while the FRIES noise was executing.

Instead of utilising the libcrypto++ library, the OpenSSL library that includes the SHA were

Table 3: The subkeys recovered as various traces were utilised.

| | Subkey | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Traces | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | Total |
| 10 | - | - | Y | - | Y | - | Y | - | Y | - | - | - | Y | - | - | - | 5 |
| 20 | Y | Y | Y | - | Y | - | Y | - | Y | - | - | - | Y | - | - | - | 7 |
| 30 | Y | Y | Y | - | Y | Y | Y | Y | Y | - | Y | Y | Y | - | - | - | 10 |
| 40 | Y | Y | Y | - | Y | Y | Y | Y | Y | - | Y | Y | Y | Y | - | - | 12 |
| 50 | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y | 16 |

investigated with regards to the EM leakage and the potential candidate as a countermeasure. As the various hash functions of the OpenSSL library executed the EM leakage was captured, with the EM signatures depicted in Figure 18.



(a) SHA-1. (b) SHA-224.

(c) SHA-256. (d) SHA-512.

Figure 18: The EM leakage for the various SHA functions from the OpenSSL library.

Based on Figure 18, the data points i.e: length of the EM signature increased in length, as seen in Figure 18a (1500 points) to Figure 18d (2600 points). This can be related to the output size of each hash function as seen in Table 1. Furthermore, the block size of the SHA-512 function is 1024 bits where as the rest of the hash function has a block size of 512.

The SHA implementations within the OpenSSL library were incorporated into the FRIES noise generator. A stress test was firstly performed against the Raspberry Pi as the FRIES noise generator was executed to run in the background indefinitely. The results revealed that the CPU usage was 100% on only two cores as opposed to all four cores when the libcrypto++ implementation was in place. In addition, there was no lag from the device. This process demonstrated the SHA functions within the OpenSSL library was well-suited to be implemented as a noise generator as the device had not suffered from the additional overhead and the AES-128 cryptographic algorithm remained hidden.

As the FRIES noise generator was executing in the background the AES-128 program was executed with random intervals on multiple occasions over the monitored time period. The EM signature of this experiment is depicted in Figure 19.

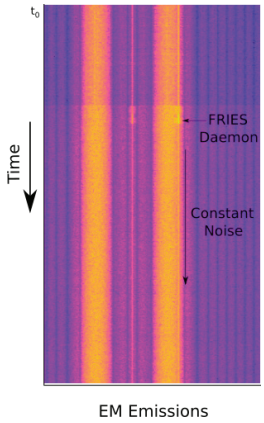



Figure 19: EM emissions depicting the startup point for the FRIES noise generator.

The annotated Figure 19 depicts an initial EM spike which is the start-up process of the FRIES noise generator. After the initialisation process has been completed, the FRIES noise generator produced a constant EM leakage footprint. Within this constant leakage, the execution of the AES-128 algorithm could not be identified visually.

A secondary experiment was conducted by alternating between the FRIES noise generator being on and off. While the FRIES noise generator was being toggled between on and off states, the AES-128 cryptographic program was executed. Within this period the EM emissions were captured and are depicted in Figure 20.

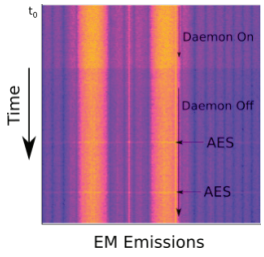



Figure 20: The EM frequency has the FRIES noise is switched on and off.

While the FRIES noise generator was executing in the background, there were no additional scan lines

as seen in Figure 20. In addition, it is demonstrated that while the Daemon was in an off state the AES-128 cryptographic execution could be seen visually as annotated by the "AES" in Figure 20 and the arrow pointing to the spike in the EM spectrum.

## 10. ATTACKING THE IMPROVED COUNTERMEASURE

Although, the results from the previous section demonstrated that the encryption process could not be identified visually, the data was further analysed. The data demonstrated that out of 80 samples, there were ten traces that gave off slight harmonic noise. These traces were extracted and analysed as depicted in Figure 21.

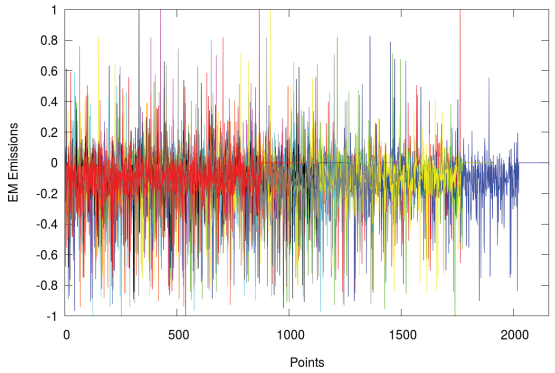



Figure 21: The EM emissions acquired from harmonic leakage.

It can be seen that each trace has a different EM signature. The data was sent to the alignment process as discussed in Section 8. The resultant aligned data was utilised as input for the CPA attack and the results indicated no secret information was revealed.

Statistical analysis was performed on the captured EM data. 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.

The Cross-correlation between AES-128 cryptographic template and the signal produced by the noise generator is illustrated in Figure 22. The x-axis is a representation of time at the point of AES-128 cryptographic signature being similar to that of the EM signature from the noise generator. The figure reveals that cross-correlation test could 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.

It is noted that there is no one evaluation process with regards to evaluating a software countermeasure, the countermeasures are generally evaluated by the number of subkeys recovered. This research felt that

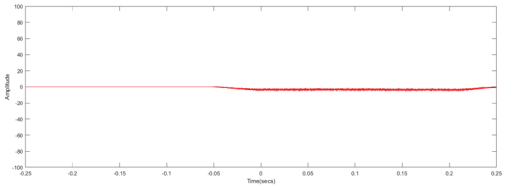


Figure 22: Cross-correlation results.

further evaluation was required, thus the Chi-square test [56] was applied. This approach is utilised to evaluate hardware countermeasures, especially with regard to hardware noise generators [57]. The purpose of this test is to evaluate how likely the EM emissions generated as the AES-128 algorithm executed can be detected within the noise. If it can still be detected that means the AES-128 cryptographic keys could be recovered. The EM signature of the iterations of the FRIES noise generator was compared to the original EM signature of the AES-128 cryptographic within a multi-thread. The results are depicted in Table 4.

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 99% probability that the cryptographic signature is within that signal. The prime number generator, when the signal was visible, depicts the same results as that of the multi-threading.

Interestingly, as the AES-128 cryptographic thread was not visible, the statistical test indicated that there was a 50% probability that the AES-128 cryptographic signal was within the noise of the prime numbers. Furthermore, the final iteration of the FRIES noise generator with the SHA functions from the OpenSSL library revealed that there was only a 1% probability that the AES-128 cryptographic EM signature was located within the noise generated.

## 11. CONCLUSION

The EM leakage of the Raspberry Pi as the device executed the AES-128 algorithm of the Crypto++ library in a threaded environment was investigated. The multi-threaded libraries used were the pThreads; C11 threads; TBB; and OpenMP. The research illustrates that different patterns in the EM emanation occurs. This was determined by monitoring the CPU usage and core intensity. The Raspberry Pi's power management system used different cores on each execution of the code, for example: only three cores were used in the first execution; all the threads would be used in the second execution. The four different thread techniques were demonstrated to have variations in frequency and shape of EM emanations leaked. The pThread thread implementation was found to exhibit the fastest runtime execution.

Table 4: 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 | – | – | – | – | – | – | – | – |

A Daemon noise generator known as the FRIES noise generator was introduced in Section 7. The results demonstrated that the FRIES noise had different effects on the radio spectrum. It was still possible to visually detect the location of the execution of the cryptographic algorithm. However, the visual detection could not been seen on some occasions. It was identified that the calculation of prime numbers would hide the cryptographic algorithm. The analysis revealed that as high prime numbers were calculated there was a window where the cryptographic algorithm could not be seen visibly.

This research improved on previous work [34] by removing the low frequencies from 0–15 kHz within the captured signal i.e: only frequencies between 15 – 50 KHz were kept. This data was utilised as input data for the CPA attack. Following this approach, this research was successfully able to recover the entire secret key used for the AES-128 cryptographic algorithm.

The prime number sequence within the FRIES noise generator was replaced by the libcrytpo++ implementation of the Secure Hash Algorithm (SHA) in Section 9. 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. Furthermore, more hashes were being calculated per second as apposed to libcrypto++ implementation. While the hash function was introducing EM noise, the cryptographic implementation of AES-128 algorithm could not be visibly seen.

Statistical analysis was performed on the FRIES noise generator in Section 10., more importantly the cross-correlation between the FRIES noise and a non-protected AES-128 cryptographic implementation and the Chi-square test 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. The OpenSSL library is therefore deemed to be well suited as a base in order to obfuscate EM SCA attacks.

This research has successfully answered the questions posed in Section 1. The AES-128 cryptographic implementation within the libcrypto++ library on a Raspberry Pi is vulnerable to SCA attacks. The cryptographic process was seen visibly within the EM spectrum. The EM data for this process were extracted and additional digital filtering techniques was applied to the signal, more specifically the removal of the 0-15 kHz frequency. The resultant data was utilised in the CPA attack and the results revealed that the entire AES-128 key was recovered. This research introduced a multi-threaded approach with the utilisation of SHA to serve as a software countermeasure to mitigate SCA attacks. The proposed countermeasure executed as a Daemon and generated EM noise that was able to hide the cryptographic implementations and prevent a CPA attack and other statistical attacks.

## 12. FUTURE WORK

This research lays the foundation for future research towards building software based Electromagnetic noise generators not only for cryptographic processes, but for other programs and applications with sensitive information. Applying cryptography on dummy executions. The datasets utilised have been made available [58].

## ACKNOWLEDGEMENT

This work was undertaken as part of the Distributed Multimedia CoE at Rhodes University, with financial support from the Information Security Competency Area within Modelling and Digital Science at the CSIR, Telkom SA, Tellabs/ CORIANT, Easttel, Bright Ideas 39, THRIP and NRF SA (UID 90243). The authors acknowledge that opinions, findings and conclusions or recommendations expressed here are those of the author(s) and that none of the above mentioned sponsors accept liability whatsoever in this regard.

## REFERENCES

[1] M. Yun and B. Yuxin, “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, June 2010, pp. 69–72.

[2] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing and its role in the internet of things," in Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing, ser. MCC '12. New York, NY, USA: ACM, 2012, pp. 13–16.

[3] S. D. T. Kelly, N. K. Suryadevara, and S. C. Mukhopadhyay, "Towards the implementation of IoT for environmental condition monitoring in homes," IEEE Sensors Journal, vol. 13, no. 10, pp. 3846–3853, Oct 2013.

[4] R. Button, "Dyn DynDNS DDoS attack," 2016, accessed: 2017-05-01. [Online]. Available: http://www.red-button.net/blog/dyn-dyndns-ddos-attack/

[5] D. Wei, Y. Lu, M. Jafari, P. Skare, and K. Rohde, "An integrated security system of protecting smart grid against cyber attacks," in 2010 Innovative Smart Grid Technologies (ISGT), Jan 2010, pp. 1–7.

[6] S. Amin, X. Litrico, S. Sastry, and A. M. Bayen, "Cyber security of water SCADA systems; Part I: Analysis and experimentation of stealthy deception attacks," IEEE Transactions on Control Systems Technology, vol. 21, no. 5, pp. 1963–1970, Sept 2013.

[7] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," IEEE Security and Privacy, vol. 9, no. 3, pp. 49–51, May 2011.

[8] U.S. Depatment of Commerce, "Advanced encryption standard (AES)," National Institute of Standards and Technology (NIST), Tech. Rep., 2001.

[9] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in Advances in Cryptology — CRYPTO' 99: 19th Annual International Cryptology Conference Santa Barbara, California, USA, August 15–19, 1999 Proceedings. Berlin, Heidelberg: Springer Berlin Heidelberg, 1999, pp. 388–397.

[10] I. Frieslaar and B. Irwin, "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), INSTICC. SciTePress, 2016, pp. 403–410.

[11] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems, ser. CHES '01. London, UK, UK: Springer-Verlag, 2001, pp. 251–261.

[12] P. B. Rao and S. Uma, "Raspberry Pi home automation with wireless sensors using smart phone," International Journal of Computer Science and Mobile Computing, vol. 4, no. 5, pp. 797–803, 2015.

[13] Crypto++, "Crypto++ library." [Online]. Available: https://www.cryptopp.com/

[14] R. M. Lee, M. J. Assante, and T. Conway, "German steel mill cyber attack," Industrial Control Systems, 2014. [Online]. Available: https://goo.gl/M7qkAN

[15] J. P. Farwell and R. Rohozinski, "Stuxnet and the future of cyber war," Survival, vol. 53, no. 1, pp. 23–40, 2011.

[16] R. A. Clarke and R. K. Knake, Cyber war: The Next Threat to National Security and What to Do About It, 1st ed. HarperCollins, 2011.

[17] T. Ring, "Connected cars - the next targe tfor hackers," Network Security, vol. 2015, no. 11, pp. 11–16, Nov. 2015.

[18] S. Parkinson, P. Ward, K. Wilson, and J. Miller, "Cyber threats facing autonomous and connected vehicles: Future challenges," IEEE Transactions on Intelligent Transportation Systems, vol. PP, no. 99, pp. 1–18, 2017.

[19] B. M. Macq and J.-J. Quisquater, "Cryptology for digital TV broadcasting," Proceedings of the IEEE, vol. 83, no. 6, pp. 944–957, Jun 1995.

[20] The Intercept, "The Great Sim Hack," 2015, accessed: 2017-02-01. [Online]. Available: https://theintercept.com/2015/02/19/great-sim-heist/

[21] M. J. Covington and R. Carskadden, "Threat implications of the internet of things," in 2013 5th International Conference on Cyber Conflict (CYCON 2013), June 2013, pp. 1–12.

[22] M. O'Neill, "The internet of things: do more devices mean more risks?" Computer Fraud & Security, vol. 2014, no. 1, pp. 16 – 17, 2014.

[23] Symantec, "Dragonfly: Western energy sector targeted by sophisticated attack group," 2017, accessed: 2017-10-01. [Online]. Available: https://www.symantec.com/connect/blogs/dragonfly-western-energy-sector-targeted-sophisticated-attack-group

[24] T. Plos, M. Hutter, and M. Feldhofer, "Evaluation of side-channel preprocessing techniques on cryptographic-enabled HF and UHF RFID-tag prototypes," in Workshop on RFID Security – RFIDSEC 2008, 2008, pp. 114–127.

[25] I. Frieslaar and B. Irwin, "Investigating multi–thread utilization as a software defence mechanism against side channel attacks," in Proceedings of the 8th International Conference on Signal Processing Systems, ser. ICSPS 2016. New York, NY, USA: ACM, 2016, pp. 189–193.

[26] D. Genkin, I. Pipman, and E. Tromer, "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. New York, NY, USA: Springer-Verlag New York, Inc., 2014, pp. 242–260.

[27] P. Belgarric, P.-A. Fouque, G. Macario-Rat, and M. Tibouchi, "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. New York, NY, USA: Springer-Verlag New York, Inc., 2016, pp. 236–252.

[28] D. Genkin, L. Pachmanov, I. Pipman, E. Tromer, and Y. Yarom, "ECDSA key extraction from mobile devices via nonintrusive physical side channels," in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, ser. CCS '16. New York, NY, USA: ACM, 2016, pp. 1626–1638.

[29] D. Aboulkassimi, M. Agoyan, L. Freund, J. Fournier, B. Robisson, and A. Tria, "Electromagnetic analysis (EMA) of software AES on Java mobile phones," in 2011 IEEE International Workshop on Information Forensics and Security, Nov 2011, pp. 1–6.

[30] N. Golyandina and A. Zhigljavsky, Singular Spectrum Analysis for time series, 2013th ed. Springer Science & Business Media, 2013.

[31] Y. Nakano, Y. Souissi, R. Nguyen, L. Sauvage, J.-L. Danger, S. Guilley, S. Kiyomoto, and Y. Miyake, "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. New York, NY, USA: Springer-Verlag New York, Inc., 2014, pp. 76–91.

[32] J. Balasch, B. Gierlichs, O. Reparaz, and I. Verbauwhede, "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, 2015, pp. 599–619.

[33] J. Longo, E. De Mulder, D. Page, and M. Tunstall, "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. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 620–640.

[34] I. Frieslaar and B. Irwin, "Recovering AES-128 encryption keys from a Raspberry Pi," in Southern Africa Telecommunication Networks and Applications Conference (SATNAC), September 2017, pp. 228–235.

[35] ——, "A multi-threading approach to secure verifypin," in 2016 2nd International Conference on Frontiers of Signal Processing (ICFSP), Oct 2016, pp. 32–37.

[36] ——, "Evaluating the multi-threading countermeasure," International Journal of Computer Science and Information Security, vol. 14, no. 12, pp. 379–387, 2016.

[37] J. L. Hennessy and D. A. Patterson, Computer Architecture: A Quantitative Approach, 5th ed. Morgan Kaufmann, 2011.

[38] D. A. Patterson and J. L. Hennessy, Computer organization and design: the hardware/software interface, 2nd ed. Morgan Kaufmann Publishers Inc., 1998.

[39] B. R. Rau and J. A. Fisher, Instruction-level parallelism, 1st ed. John Wiley and Sons Ltd., 2001.

[40] W. Kunikowski, E. Czerwiński, P. Olejnik, and J. Awrejcewicz, "An overview of ATmega AVR microcontrollers used in scientific research and industrial applications," Pomiary Automatyka Robotyka, vol. 19, no. 215, pp. 15–20, 1 2015.

[41] A. S. Tanenbaum, A. S. Woodhull, A. S. Tanenbaum, and A. S. Tanenbaum, Operating systems: design and implementation, 3rd ed. Prentice-Hall Englewood Cliffs, NJ, 1987.

[42] B. Barney, "POSIX threads programming," 2017, accessed: 2017-03-01. [Online]. Available: https://computing.llnl.gov/tutorials/pthreads/

[43] M. A. Ellis and B. Stroustrup, The Annotated C++ Reference Manual, 1st ed. Addison-Wesley Longman Publishing, 1990.

[44] J. Reinders, Intel threading building blocks: outfitting C++ for multi-core processor parallelism, 1st ed. O'Reilly Media, 2007.

[45] L. Dagum and R. Menon, "OpenMP: an industry standard API for shared-memory programming," IEEE computational science and engineering, vol. 5, no. 1, pp. 46–55, 1998.

[46] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," in Proceedings of the Ninth Annual ACM Symposium on Theory of Computing, ser. STOC '77. New York, NY, USA: ACM, 1977, pp. 106–112.

[47] M. Bellare, R. Canetti, and H. Krawczyk, "Keying hash functions for message authentication," in Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology, ser. CRYPTO '96. London, UK, UK: Springer-Verlag, 1996, pp. 1–15.

[48] D. Eastlake, 3rd and P. Jones, "US Secure Hash Algorithm 1 (SHA1)," National Security Agency, United States, Tech. Rep., 2001. [Online]. Available: https://tools.ietf.org/html/rfc3174

[49] B. Preneel, "Cryptographic hash functions," European Transactions on Telecommunications, vol. 5, no. 4, pp. 431–448, 1994.

[50] B. Sklar, Digital communications, 2nd ed. Prentice Hall, 2001.

[51] I. Frieslaar and B. Irwin, "Investigating the electromagnetic leakage from a Raspberry Pi," in 2017 Information Security South Africa (ISSA), August 2017.

[52] ——, "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), ser. SACSIT 2017. New York, NY, USA: ACM, 2017.

[53] J. G. J. van Woudenberg, M. F. Witteman, and B. Bakker, "Improving differential power analysis by elastic alignment," in Proceedings of the 11th International Conference on Topics in Cryptology: CT-RSA 2011, ser. CT-RSA'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 104–119.

[54] R. W. Schafer, "What is a savitzky-golay filter? [lecture notes]," IEEE Signal Processing Magazine, vol. 28, no. 4, pp. 111–117, July 2011.

[55] E. Brier, C. Clavier, and F. Olivier, "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. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 16–29.

[56] A. Satorra and P. M. Bentler, "A scaled difference chi-square test statistic for moment structure analysis," Psychometrika, vol. 66, no. 4, pp. 507–514, Dec 2001.

[57] A. Gornik, A. Moradi, J. Oehm, and C. Paar, "A hardware-based countermeasure to reduce side-channel leakage: Design, implementation, and evaluation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 34, no. 8, pp. 1308–1319, Aug 2015.

[58] I. Frieslaar and B. Irwin, "Electromagnetic data from a Raspberry Pi 2 – dataset," Aug 2017, open Science Framework. Accessed: 2017-08-01. [Online]. Available: https://osf.io/mte5q