# A Multi-Threading Approach to Secure *VERIFYPIN*

Ibraheem Frieslaar
Modelling and Digital Science,
Council for Scientific and Industrial Research,
Pretoria, South Africa.
Email: ifrieslaar@csir.co.za

Barry Irwin
Department of Computer Science,
Rhodes University,
Grahamstown, South Africa
Email: b.irwin@ru.ac.za

***Abstract*—This research investigates the use of a multi-threaded framework as a software countermeasure mechanism to prevent attacks on the *verifypin* process in a pin-acceptance program. The implementation comprises of using various mathematical operations along side a pin-acceptance program in a multi-threaded environment. These threads are inserted randomly on each execution of the program to create confusion for the attacker. Moreover, the research proposes a more improved version of the pin-acceptance program by segmenting the program. The conventional approach is to check each character one at a time. This research takes the verifying process and separates each character check into its individual thread. Furthermore, the order of each verified thread is randomised. This further assists in the obfuscation of the process where the system checks for a correct character. Finally, the research demonstrates it is able to be more secure than the conventional countermeasures of random time delays and insertion of dummy code.**

***Index Terms*—Electromagnetic Analysis, Side Channel, Software Countermeasure, Verify.**

## I. Introduction

The development of sensitive applications such as financial and privacy applications to store vital information has become a critical security concern. Many of these applications are developed in tandem with embedded devices. These embedded devices assist the developer in protecting the individual's sensitive data.

The dilemma presented to software security developers is to implement a software countermeasure to defend against side channel analysis (SCA). The challenging aspect in this field lies with the hardware that executes the algorithm. Although the algorithm's mathematical integrity is kept in place, microcontrollers leak out sensitive information. Kocher *et al.* [7] have demonstrated that there is a correlation between the secret key-dependent intermediate variables and the power consumption of a embedded device. Upon acquiring this information the attacker is able to retrieve the secret information from the device [8].

Electromagnetic Analysis (EMA) attacks is a well known alternative to power consumption [3]. Electromagnetic (EM) emanations are captured from the device and are subsequently used to retrieve the secret information. EM measurements do not require the attacker to have direct contact with the device. Therefore EMA is less intrusive than the conventional power analysis.

There are a limited number of studies performed on SCA attacks on more powerful general purpose devices such as computers and smartphones [4], [5]. This research field is a vital opportunity to discover critical information on the behaviour of these powerful devices. Furthermore, it is projected that by 2017 , there will be 2.2 billion smartphones worldwide [14]. That essentially is 2.2 billion vulnerable devices.

Many of these smartphones and high powered devices contain hardware capable of executing software in a parallel sequence. Multi threading is mainly used as a means to increase performance and runtime execution of an algorithm. Therefore, this research aims to investigate the use of multi-threads as a countermeasure. This investigation is the first of its kind, as there has not been a software countermeasure based on multi-threading to secure *verifypin* on microcontrollers from SCA attacks. This study will evaluate the proposed countermeasure against a real world scenario of a pin-acceptance program that is attacked using SCA.

The remainder of this paper is organized as follows: Section II will discuss the basic fundamentals of threading; Section III will detail the research methodology; followed by the results and analysis in Section IV and V. Finally, the paper is concluded in Section VI

## II. Threading

Multi-threading is the process of executing multiple threads or instructions concurrently. These threads are managed independently by a task scheduler [6]. 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 [12]. 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' utilization of resources to increase the runtime execution of a program [13]. 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 central processing units (CPU)s.

Based on the above procedures, multi-threading can be further broken down into three techniques. These techniques are block, interleaved, and simultaneous multi-threading [6]. The most 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. In effect creating a stall or 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 execution pipeline. As there is a high probability of one thread being independent from 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 preemptive multi-tasking [10].

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 [6].

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 [1]. 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 use many different libraries to develop their code in a multi-threaded framework. The most common framework in the UNIX environment is the POSIX Threads (Pthreads) [9]. Multi threading libraries provide developers access to creating applications with multi-threaded support. Furthermore, these libraries provides developers with the option to create a synchronised environment between the threads by using mutexes, condition variables, semaphores, monitors and other synchronization primitives [15]. This research makes use of the AVR thread library [2] for multi-threading on microcontrollers. This library is based on the preemptive multi-tasking technique.

## III. Methodology

This section discusses the hardware, and the approach used to capture the EM emanations from the devices under test. Furthermore, the experiments used in this research are discussed in this section.

The device under test was the Atmel ATmega328p microcontroller. The Atmel microcontrollers was selected as it was demonstrated to be used under various scientific and industrial research, and applications [9]. In order to complete a successful capture of the EM emanations from the devices, an open source development kit was used. This kit is known as the ChipWhisperer kit [11]. The kit consisted of measuring equipment and capturing software at a low price of $325. The ChipWhisperer's hardware comprised of a field-programmable gate array (FPGA). The FPGA used was the Spartan 6 LX25 FPGA [16]. The ChipWhisperer allows users the ability to upload their own executable code onto the microcontrollers. For a more comprehensive description on all the functionality of the ChipWhisperer, refer to [11].

The ChipWhisperer Lite Board was used to capture the EM emanations from the microcontroller. The printed circuit board (PCB) board which contains the Atmega328p microcontroller, is referred to the device under test. The ChipWhisperer was connected to the device under test. In order to capture Electromagnetic (EM) emanations the ChipWhisperer was connected to a low noise amplifier which in turn was connected to an EM probe. The probe was placed over the Atmega328p processor. Fig. 1 illustrates the hardware setup.

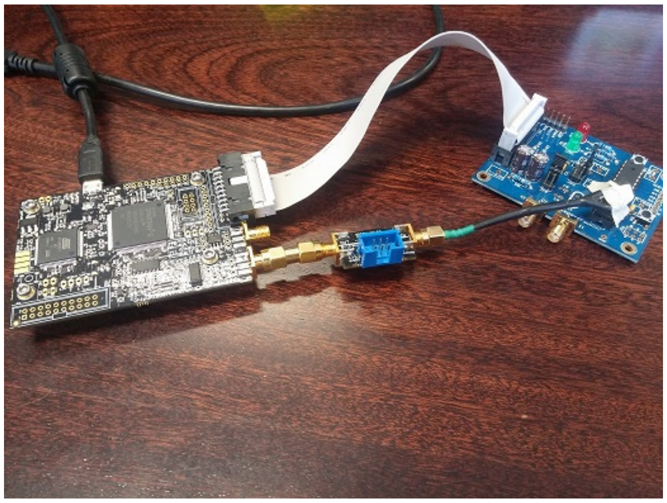
Fig. 1: The hardware setup: On the left, the ChipWhisperer connected to the PCB containing the Atmega328p processor; A low noise amplifier is connected to an EM probe placed on the processor.

The ChipWhisperer Lite was connected to a computer. The Python programming language was used to interface between the computer and the ChipWhisperer. Instructions were sent to the microcontroller to execute the programs. While the execution of the programs were commencing, the

EM emanations was captured by the ChipWhisperer capture software.

The experiments comprised of various programs executed on the microcontroller. All programs was developed and compiled in C and converted to a hex file that was flashed onto the microcontroller.

The research used the AVR Thread Library multi-threading framework [2] as the proposed countermeasure. This framework was designed to work with Atmel family of microcontrollers. The framework implements a simple round-robin style task switcher with basic preemptive multi-tasking. Originally, the framework was designed to work with earlier versions of Atmel microcontrollers. The last revision of this framework was in 2008. Therefore, the framework was modified and recompiled to work for the latest generation of the Atmel microcontrollers.

This research implemented a real world scenario where a pin-acceptance program was executed on the microcontroller. The program instructed the user to enter a five character password. This will form part of experiment one. The EM emanations of the microcontroller was monitored as the correct and incorrect password was entered into the system. It will be demonstrated that using SPA the attacker was able to predict the correct password by monitoring the EM emanations. It should be noted that the pin-acceptance program was designed to have the basic countermeasure of random time delays in the system.

The second set of experiments implemented a known countermeasure of using random dummy instructions within the pin-acceptance program, Followed by the final set of experiments which comprised of programming the same pin-acceptance program within the multi-thread framework. Additionally, execution time of the pin-acceptance program with and without the various software countermeasures was recorded.

## IV. Results

This section examines the results obtained based on the experiments discussed in the previous section. Experiment one is analysed to illustrate that the research is able to successfully perform a simple power analysis attack by monitoring the EM emanations in the time frequency domain.

The EM emanations from the device as the correct and incorrect password are entered is depicted in Fig. 2a and 2b, respectively. It is observed from points 100 – 300 in both traces there is a difference in EM emanations. Therefore, the attacker can clearly see the difference in EM emanations when correct and incorrect passwords are entered. This is further evident in Fig. 3 which represents the EM emanations from points 0 – 300.

Observing Fig. 3a the square block demarcates the EM emanation for the characters that are entered correctly. Comparing Fig. 3a and 3b, there is a distinct difference in the EM emanations from points 100 – 300. Furthermore, Fig. 4 depicts the change in EM emanations as more correct characters are entered into the system.

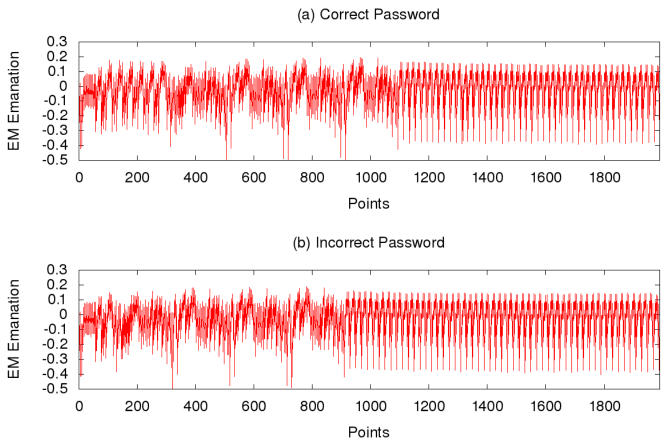

Fig. 2: The EM emanations as the correct (a) and incorrect (b) password are entered

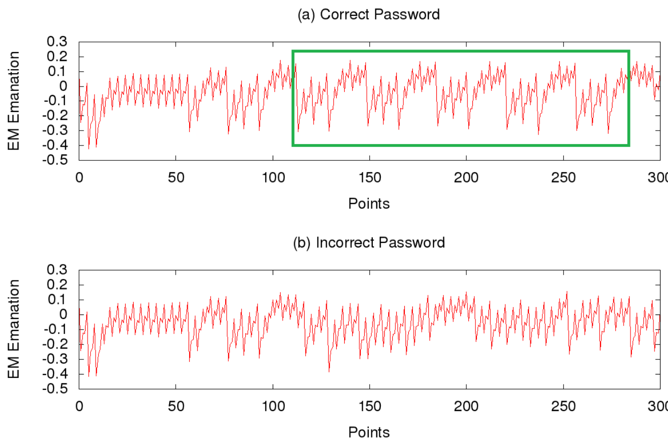

Fig. 3: The EM emanations as the incorrect (a) and correct (b) password are entered.

The rectangular blocks illustrates the change in a byte from 0 to 1 when a correct character is entered. This is seen as EM emanation spikes close to 0.3. Additionally, it indicates that going from one correct character to the next character, the change in bytes appear to shift forward in time by every 36 points. Having this information the attacker is able to create an automatic process to guess the password based on the EM emanations.

The attacker creates a script that loops through all possibilities per character and based on the EM emanations looks for a change in bytes at a specific point of interest. Upon acquiring the first correct character, the next character is attacked by moving forward 36 points. This process repeats until all characters are correct and the attacker has access to the system.

Table I represents the results for attacking a pin-acceptance system with different password types, automatically. These password types are numeric, alphabetical, and alpha-numeric passwords. As shown in the table, it takes under 3 minutes to

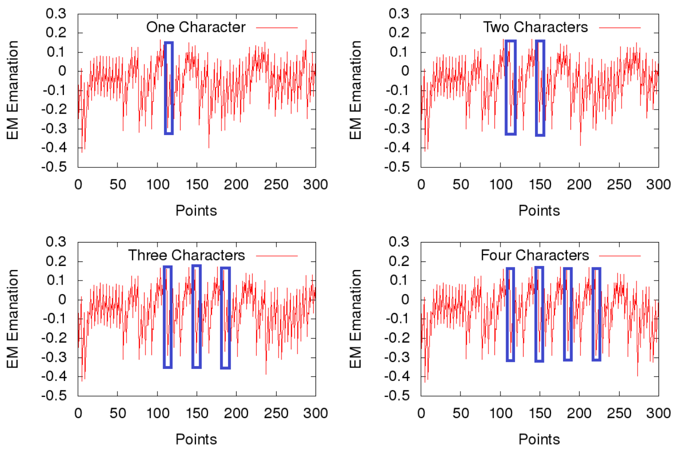

Fig. 4: The change in EM emanations as more correct characters are entered into the system.

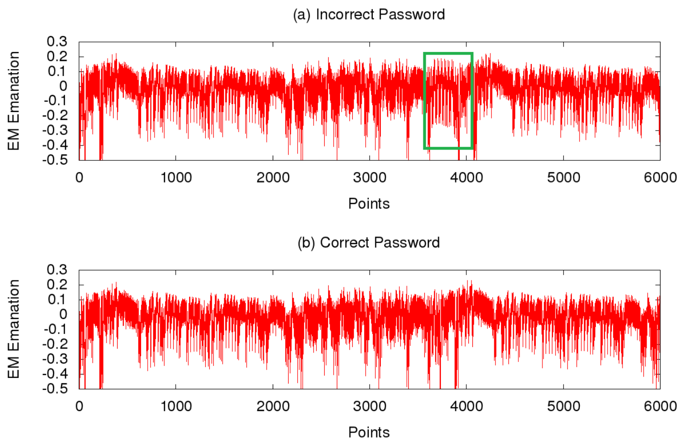

Fig. 5: The EM emanations as the incorrect (a) and correct (b) password are entered with a known countermeasure in place.

predict the entire password if the system is using an alpha-numeric password and, only 48 seconds to predict a numeric password.

Table I: Time taken to predict the correct password

| Password Type | Time (m) |
|---|---|
| Numeric | 0.48 |
| Alphabetical | 1.12 |
| Alpha-numeric | 2:41 |

Fig. 5 depicts the difference between EM emanations as the correct and incorrect password are entered while the known countermeasure of inserting random dummy code is in place. Although, more noise is generated the attacker is still able to see the process of verifying the correct characters. Taking a closer look at the figure, it can be seen at the points close to 4000, there is a difference between the two traces. The added spike in Fig. 5a is the location of the correct characters being accepted.

Analysing the next set of experiments Fig. 6 and Fig. 7 illustrates the EM emanations as the pin-acceptance program is encapsulated into the thread framework. Furthermore, Fig. 6 is when the correct password is entered and Fig. 7 is when the incorrect password is entered.

Observing both figures, the EM emanations are very similar to each other. This is due to the fact that the threaded framework makes use of a 256 byte size stack for each thread. Since each thread has the same stack size, the attacker is unable to visibly see a difference when entering the correct and incorrect password. The attacker is now at a disadvantage and unable to determine the correct region of interest.

The proposed technique is improved by adding dummy threads. These dummy threads consists of various mathematical operations. Fig. 8 depicts the EM emanations as dummy threads are inserted randomly with the pin-acceptance thread. A further improvement is made by randomising each thread on each execution of the code. This enables the system to create

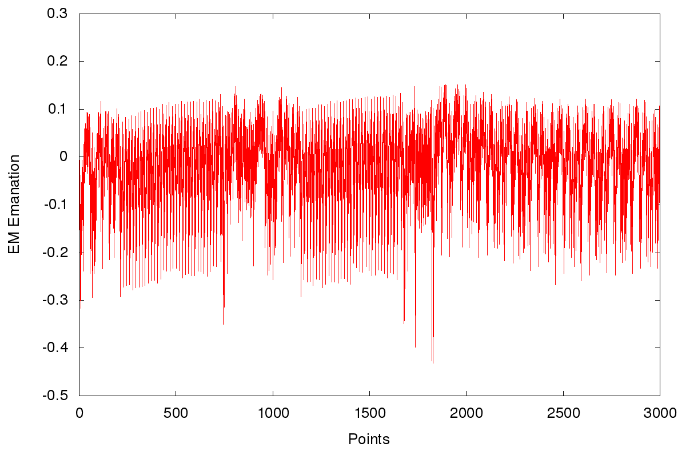

Fig. 6: The EM emanations as the correct password is entered in the threaded framework.

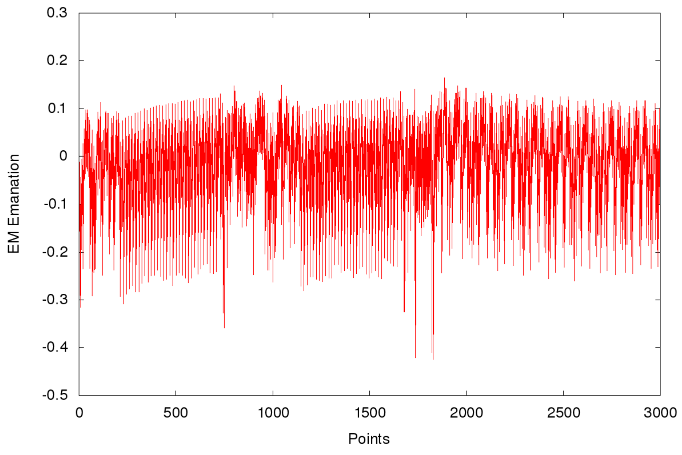

Fig. 7: The EM emanations as the incorrect password is entered in the threaded framework.

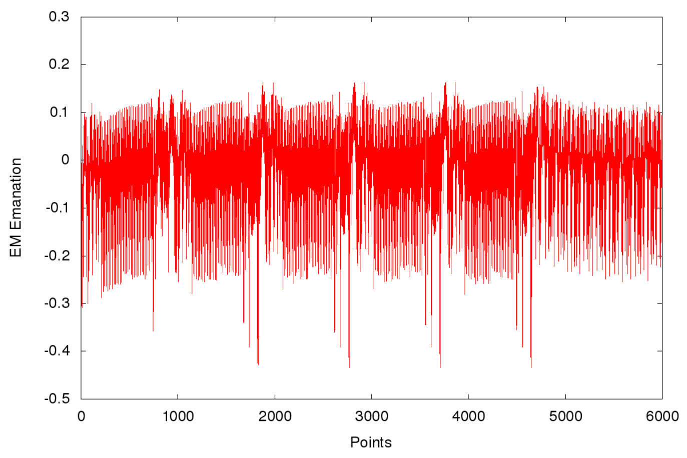

Fig. 8: EM emanations as dummy threads are inserted randomly with the pin-acceptance thread.

dynamic threads that changes location on each execution. This makes it even harder for the attacker to use timing attacks, as the attacker is unable to locate the correct process for verifying the password.

Since each thread is similar to others. This research proposes a more improve version of the pin-acceptance program by segmenting the pin-acceptance program. The conventional method is to check each character one at a time. This research takes the verifying process and separates each character check into its individual thread. Furthermore, the order of each thread that is verified is randomised. This further assists in the obfuscation of the process where the system checks for a correct character. Additionally, noise threads can be added to improve randomisation.

Table II displays the average time it takes to execute one occurrence of the pin-acceptance program, with and without the various countermeasures in place. From the table, it is observed that using the proposed countermeasure as the pin-acceptance is encapsulated into a thread, only results in a small 4ms increase. Furthermore, using the countermeasure where the verification of characters is 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 secured.

Table II: Average execution time of the system with and without the countermeasures

| Programs | Time (s) |
|---|---|
| Pin-Acceptance | 0.113 |
| Pin-Acceptance + Multi Thread, 3 Threads | 0.117 |
| Pin-Acceptance + Verification Threads, 8 Threads | 0.125 |

## V. Analysis

Examining the results in the first set of experiments, this research demonstrates the ability to successfully predict the correct password by monitoring the EM emanations of a device. The process is automated and the correct password is predicted in under 3 minutes for alpha-numeric characters.

The research implements the threaded framework as a software countermeasure to a SPA attack. The results indicate that the proposed technique prevents the attacker from being able to predict the password from the EM emanations. Fig. 9 illustrates a comparison between the implementation of the pin-acceptance program without a software countermeasure, and with the multi-thread countermeasure in place. Based on the figure, it can be visually determine that entering the incorrect password does not allow the attacker to obtain sensitive information.

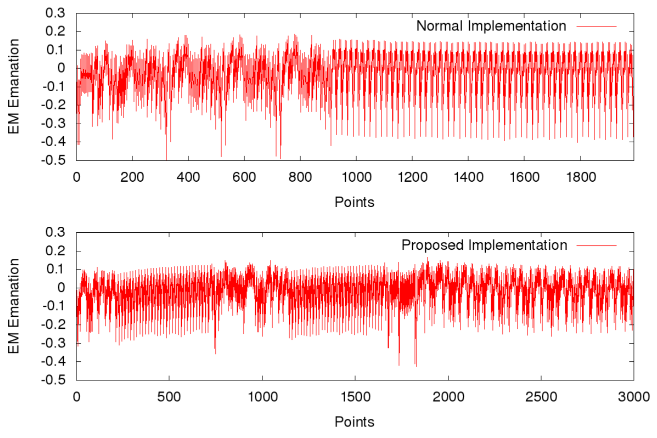

Fig. 9: A comparison between the EM emanations when the incorrect password is entered for the multi-thread framework and no countermeasure.

The results demonstrates that a known countermeasure of inserting dummy instructions is still vulnerable, as the attacker is able to see the difference between a correct and incorrect password. This method prevents an automatic approach as the dummy code could be placed at every character check. However, the attacker would still be able to notice the difference.

Using the multi-thread approach as a software countermeasure allows for flexibility. The developer is able to randomise each thread on each execution of the code. This prevents glitch attacks as the execution of the thread with the pin-acceptance program would execute at a different location on each runtime. Furthermore, a template attack would be extremely difficult since all the threads are similar.

The research further improves the security of the verification process by allowing each character check to have it's own individual thread that is executed in a random order per iteration. Reviewing Fig. 10 it is observed that all eight threads look identical to each other. Five threads for the character check and three random noise threads. This allows the developer a greater security option by adding more threads. This technique demonstrates that it is able to prevent side channel analysis attacks against a pin-acceptance program. Additionally, based on the time results in Table II when 8 threads are used, the extra time overhead is 12ms. Therefore,

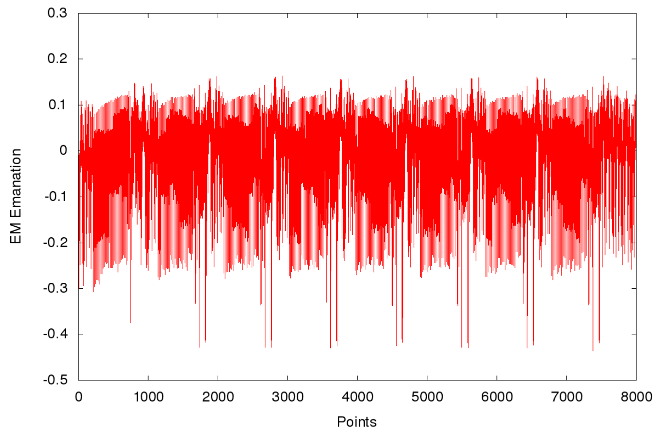

Fig. 10: The power traces as the incorrect and correct password is entered with the known software countermeasure

the proposed countermeasure prevents an attack with minimal overhead.

## VI. Conclusion and Ongoing Research

In this research a novel approach is introduced to protect pin-acceptance programs from side channel attacks. The implementation comprised of a multi-thread framework on a single core microcontroller. These threads consist of dummy mathematical operations to be inserted randomly on each execution of the program, while the pin-acceptance thread is executed. The research demonstrated it was able to outperform known countermeasures of random time delays and insertion of dummy code. Furthermore, the research demonstrates that segmenting the verification process into smaller individual threads leads to a more secure process by randomising each thread check with a minimal execution overhead.

This work sets the basis for the forthcoming research where this approach would be implemented and tested on true multi-threading platforms, such as smartphones and laptops. Moreover, the research aspires to implement a countermeasure capable of defending embedded hardware and high-powered devices against known and future side channel attacks.

## 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] S. Akhter and J. Roberts, *Multi-core programming*. Intel press Hillsboro, 2006, vol. 33.
[2] D. Ferreyra, "Avr development." Nov. 2008. [Online]. Available: http://www.bourbonstreetsoftware.com/AVRDevelopment.html
[3] K. Gandolfi, C. Mourtel, and F. Olivier, "Electromagnetic analysis: Concrete results," in *Cryptographic Hardware and Embedded SystemsCHES 2001*. Springer, 2001, pp. 251–261.
[4] D. Genkin, L. Pachmanov, I. Pipman, and E. Tromer, "Stealing keys from pcs using a radio: Cheap electromagnetic attacks on windowed exponentiation," in *Cryptographic Hardware and Embedded Systems–CHES 2015*. Springer, 2015, pp. 207–228.
[5] ——, "Ecdh key-extraction via low-bandwidth electromagnetic attacks on pcs," in *Topics in Cryptology-CT-RSA 2016*. Springer, 2016, pp. 219–235.
[6] J. L. Hennessy and D. A. Patterson, *Computer architecture: a quantitative approach*. Elsevier, 2011.
[7] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in CryptologyCRYPTO99*. Springer, 1999, pp. 388–397.
[8] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi, "Introduction to differential power analysis," *Journal of Cryptographic Engineering*, vol. 1, no. 1, pp. 5–27, 2011.
[9] 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, 2015.
[10] L. Levinson, R. Männer, M. Sessler, and H. Simmler, "Preemptive multitasking on fpgas," in *fccm*. IEEE, 2000, p. 301.
[11] C. O'Flynn and Z. D. Chen, "Chipwhisperer: An open-source platform for hardware embedded security research," in *Constructive Side-Channel Analysis and Secure Design*. Springer, 2014, pp. 243–260.
[12] D. A. Patterson and J. L. Hennessy, *Computer organization and design: the hardware/software interface*. Newnes, 2013.
[13] B. R. Rau and J. A. Fisher, *Instruction-level parallelism*. John Wiley and Sons Ltd., 2003.
[14] Statista, "Number of smartphone users worldwide from 2014 to 2019." Aug. 2015. [Online]. Available: http://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide/
[15] A. S. Tanenbaum, A. S. Woodhull, A. S. Tanenbaum, and A. S. Tanenbaum, *Operating systems: design and implementation*. Prentice-Hall Englewood Cliffs, NJ, 1987, vol. 2.
[16] ZTEX, "Spartan 6 lx9 to lx25 fpga board." Nov. 2016. [Online]. Available: http://www.ztex.de/usb-fpga-1/usb-fpga-1.11.e.html