An Investigation into the Signals Leakage From a Smartcard based on Different Runtime Code

Ibraheem Frieslaar*†, Barry Irwin †

†Department of Computer Science, Rhodes University, Grahamstown, South Africa.

Abstract—This paper investigates the power leakage of a smartcard. It is intended to answer two vital questions: what information is leaked out when different characters are used as output; and does the length of the output affect the amount of the information leaked. The investigation determines that as the length of the output is increased more bus lines are switched from a precharge state to a high state. This is related to the output array in the code increasing its length. Furthermore, this work shows that the output for different characters generates a different pattern. This is due to the fact that various characters needs different amount of bytes to be executed since they have different binary value. Additionally, the information leaked out can be directly linked to the smartcard's interpreter.

Index Terms—Smartcards, DPA, side channel.

I. INTRODUCTION

A smartcard’s physical features is identical to that of a credit card in shape and size. However, the strength of a smartcard is contained in the embedded microprocessor that is integrated into the card. In most cases the microprocessor is under a gold contact pad [28]. The microprocessor on the chip has built in cryptographic hardware features that are used for security purposes. This allows the smartcard to encrypt important information.

These cryptographic modules which implement cryptographic algorithms, such as Data Encryption Standard (DES), Triple DES (3DES) and Advanced Encryption Standard (AES) play a critical role in safe guarding our private information. However, since the first power analysis attack published by Kocher et al. [10] researchers have begun to exploit the vulnerability of cryptographic modules using side channel power analysis attacks [13], [16], [25].

Side channel power analysis measures the power consumed as each clock cycle is executed on a digital device. The microcontroller unintentionally leaks out information, with the power consumption being directly linked with the processor [1]. Having this information it is possible to deduce what data is being processed by the device. This can be further described as follows: In a digital device a physical charge is needed to the change the state of a bus line, a change from low to high (0 – 1). It would be trivial to capture information if we assume all bus lines are set to low, with the expected outcome, that the current would be proportional to that of the number of bus lines changing to a high state. However, microcontrollers set their bus lines to a precharge state [21]. This state is in the middle of the low and high state. Therefore, it is possible to measure the current flow of every clock cycle at the power supply pin, which is known as the $V_{CC}$ line, with the current being linearly related to the number of bus lines going from a precharge state to a high state. Furthermore, as the current peak grows, the more bus lines have switched to a high state [18].

The main focus of side channel analysis is to attack the cryptographic algorithms. However, in this paper we intend to investigate other possible leakages that can occur when a program is executed on a smartcard. It is intended to answer two vital questions: what information is leaked out when different characters are used as output; and does the length of the output affect the amount of the information leak. To our knowledge this type of investigation has not been carried out before. Quisquater and Samyde [23] stated that it is possible to develop a smartcard app that can be used to reveal various signals based on the code. The rest of the paper is as follows: Section II discusses the research carried out in the field of power analysis attacks; Section III details the equipment and techniques used for the proposed system with Section IV discussing the experimental design; Section V will contain the results, with Section VI analysing the results; and finally Section VII and VIII will conclude the paper with future work.

II. SIDE CHANNEL ANALYSIS

The first side channel analysis attack that was successfully performed was in 1996 by Kotcher [11] where he was able to retrieve private information by simply monitoring the runtime of a crypto algorithm. In 1999 with the aid of Jun and Jaffe they introduced the power analysis attacks [10]. They observed that using the power consumption of cryptographic devices they were able to retrieve the secret key. Since this discovery many researchers have carried out research in the field of side channel analysis [1], [4], [7], [15], [18].

Further research into this field has shown the electromagnetic (EM) emanation can be used as a means of a side channel attack [1], [23]. Up till today power and EM side analysis has been in the forefront of side channel attacks because of its simplicity, low-cost, and efficiency compared to other existing side channel attacks. The other forms of side channel attacks are making use of sound [4], optical emission and using the temperature as a side channel attack [7].
As side channel analysis increases, researchers have attempted to introduce new encryption schemes in the hope of preventing side channel attacks. A few of these schemes are Piccolo [26], LED [6], TWINE [27], and Klein [5]. However, these new cryptographic algorithms have been shown to be vulnerable. Aumasson et al. [2] demonstrated they are able to use differential power analysis (DPA) to attack KLEIN. Mendel et al. [14] demonstrates they are able to successfully attack LED. Isobe and Shibutani shows they are able to use man in the middle attacks against LED and Piccolo [8]. Additionally, there have been numerous attacks on AES-128 [3], [12], [24]. In the latest work researcher are now focusing on AES-256 and have shown they are able to break it using side channel analysis [15], [18], [29].

III. EQUIPMENT AND TECHNIQUES

Since Kocher et al. [10] introduced attacks against smartcards using Differential Power Analysis (DPA) there has been a rapid growth in research focusing on side channel attacks [1], [4], [7], [15], [18]. However, a problem arose when comparing results as researchers were using different equipment and there was no standard in place for side channel attacks. One solution to achieve a standard platform attack is the SASEBO attack board [9]. Additionally, Oswald made the OpenSCA toolbox openly available [20] and Mangard et al. published a book for DPA attacks [13] in the hope of achieving a standardised approach to attack smartcards. Although, these basic resources have been made available there is still much more to be done to achieve a standard form of attack for everyone. The downside of the SASEBO board is that the researcher needs to use an external oscilloscope.

In this research we make use of a complete platform introduced by O’Flynn [19], known as the ChipWhisperer kit. This kit sets up the platform for side channel attacks as it consists of the main elements required for an attack: target device, measurement equipment, capture software, and attack software. The ChipWhisperer kit is extremely beneficial for students working in a low cost lab. Both the hardware and software has been built from open source materials. Therefore, allowing flexibility and growth of the device. The rest of this section is divided into two sub sections describing the hardware and software used in the ChipWhisperer.

A. Hardware

The ChipWhisperer kit has many features and attack possibilities. However, only the essential parts needed for our attack will be discussed. For a detailed explanation of the kit, the reader is referred to [19].

The hardware consists of a field-programmable gate array (FPGA). The ZTEX FPGA Module uses a Spartan 6 LX25 FPGA. Figure 1 illustrates the FPGA with an OpenADC as the analog front-end. The FPGA board provides multiple features specifically designed for side-channel analysis: two points for mounting ADC or DAC boards, an AVR programmer, voltage level translators for the target device, clock inputs, power for a differential probe and Low Noise Amplifier (LNA), external Phase Locked Loop (PLL) for clock recovery. Additionally, there are extension connectors for future improvements such as fault injection hardware. The FPGA consists of input/output blocks which includes a Universal asynchronous receiver/transmitter (UART), a Smart Card interface. This interface can be controlled from the computer.

Fig. 1. ZTEX FPGA Module with a Spartan 6 LX25 FPGA with an OpenADC as the analog front-end.

The hardware allows for the use of a technique known as synchronous sampling. This grants the ability to synchronise the sample clock with the device clock. In [17] it is demonstrated that sampling at 96 MS/s synchronously achieves the same results as sampling at 2 GS/s asynchronously. Once the system clock and the device clock is synchronised it is possible to multiple the digital signal.

There are several triggering options available such as rising and falling edge. For analysing devices that has been programmed by the researcher, the rising edge is used when an event is triggered in the code.

Implementing an attack is carried out by using the provided board known as the multi-target board. The multi-target board is connected to the FPGA module. The multi-target board consists of various areas that can target different devices such as an AVR microprocessor or a smartcard. The multi-target board consists of jumpers that can be changed around, thus allowing to switch between attacking targets. Figure 2 illustrates the multi-target board with the multiple areas labelled and outlined.

B. Software

The implementation of the software interface is designed in Python. Python is used since it is cross platform, thus allowing to be used both on Windows and Linux systems. The GUI interface is created using PySide. This allows for
other languages such as C++ to interface with it\textsuperscript{22}. The functionality consists of cryptographic functions, plotting, numeric computations, low-level IO, and smartcard interfaces. The use of Python makes it easy to develop scripts to carry out an attack.

The software comprises of two programs, one for capturing data and one for analysing data. This separation allows for researchers with existing data to feed their data into the analysis program without having to recapture data again. The capture program is ideal for researchers starting new experiments using various settings. The software also consists of a debug window. This displays the input and output results and can be used to confirm if the correct output has been achieved.

IV. EXPERIMENTAL DESIGN

The experimental setup was as follows: The smartcard was placed into the multi-target board, the target board was connected to the FPGA which was connected to the computer. Using the capture software, Python was used to send commands to the multi-target board to execute the test program on the smartcard. Figure\textsuperscript{3} illustrates the experimental setup.

The test program consists of the English alphabet, henceforth, known as the alphabet app. The commands sent to the smartcard instruct the program which character to output and the total amount of characters. While the program was executed the capture software captured the power trace, the values of the power trace were stored in a comma-separated values (CSV) file.

The raw data from the CSV file was sent to a separate C++ program that analyses the data by calculating: the minimum, maximum, mean, variance, standard deviation and coefficient variance of each trace. Using Gnuplot the raw data was used to plot the traces on one graph and an average trace was also drawn on its own graph. Additionally, for each trace the autocorrelation was calculated and plotted on a graph. Examples of these graphs can be seen in Section V.

For each experiment the following variables remained constant: The same smartcard was used and each program was compiled on the same java compiler. The smartcard was a javacard running the GlobalPlatform 2.1.1 operating system and all code was compiled using the javacard 2.2.1 framework. Each set consisted of 5000 samples per trace. The gain was set to low and 28.0586 dB was used to increase the signal output. The falling edge technique was used when capturing traces. The clock frequency was multiplied by 4 to give a frequency of 14.3 MHz. Each experiment was repeated 10 times.

The first experiment comprised of generating a baseline. To obtain this baseline, a program only outputting “1” was executed on the smartcard and the data was used to plot a graph. This graph was used as the baseline graph and it was compared to experiments two and three.

The second experiment consisted of executing the alphabet app. For this experiment the character “A” was selected as the output with the length changing from 18, 49, 79 to 89. The resulting graphs were compared against each other and the baseline.

The third experiment consisted of executing the alphabet app. For this experiment the character “X” was selected as the output with the length changing from 18, 49, 79 to 89. The resulting graphs was compared against each other and the baseline. Additionally, experiment two and three was compared against each other.

V. RESULTS

This section discusses the results obtained from the experimental setup in Section IV with each experiment being divided into its own subsection.

A. Experiment One

Figure 4 illustrates the power trace acquired in experiment one. Figure 5 is a zoomed in view of the power trace from...
between the various lengths of X’s and the baseline. The output of 49 X’s is outputting more power. Close to the sample point of 40, where 18 X’s power usage is very close to the sample point of 40, where 18 X’s power usage is similar to each until it gets close to the sample point of 40, where 18 X’s power usage is slightly more. Comparing 49 X’s to 18 X’s and the baseline it is clear to see that 49 X’s is outputting more power.

Figure 10 depicts a comparison of the autocorrelation between the various lengths of X’s and the baseline.

B. Experiment Two

Figure 7 illustrates three different power traces. The red wave is for the output of 49 A’s, the green wave is for the output of 79 A’s and the cyan wave represents the baseline. It can be seen that 49 A’s and the baseline are very similar with each other except just before the 40 sample mark, where 49 A’s has a slight increase in spike. Comparing 79 A’s to the baseline and 49 A’s, it is seen that the spikes increase and close to the 30 sample mark it has an extra spike that the 49 A’s and baseline does not have.

Figure 11 illustrates the difference between the autocorrelation of 49 A’s compared to 49 X’s and 79 A’s compared to 79 X’s respectively. In both figures there is a visible difference in the pattern of A’s and X’s. Therefore, different characters can yield a different power trace.

C. Experiment Three

Figure 9 illustrates three different power traces. The red wave is for the output of 18 X’s, the green wave is for the output of 49 X’s and the cyan wave representing the baseline. The baseline and 18 X’s are very similar to each until it gets close to the sample point of 40, where 18 X’s power usage is slightly more. Comparing 49 X’s to 18 X’s and the baseline it is clear to see that 49 X’s is outputting more power.

Figure 12 depicts a comparison of the autocorrelation between the various lengths of A’s and the baseline. The baseline is represented by the cyan wave, with red; green; blue; and purple wave representing 18,49,79 and 89 X’s respectively. It can be seen that 49,79 and 89 X’s have a similar pattern. Although, these three waves are similar, it is very different when compared to the baseline and 18 X’s. This further supports the idea that when the output length is increased, more data bus lines on the smartcard is changed from a precharge state to a high state.

Fig. 5. The power trace for the baseline in Experiment One from sample 0 – 120.

Fig. 6. Autocorrelation of the baseline power trace over a period of 70 lag.

Sample 0 to 120. Figure 5 represents the autocorrelation of the power trace only using the samples from 0 to 120 over a period of 70 lag. Looking at Figure 4 it is clear to see that there is a spike in power and then the power wave begins to normalise. This can be attributed to the bus lines changing from a precharge state to a high state and back to a precharge state. Therefore Figure 5 is depicted with only samples 0 to 120, after 120 samples the smartcard has already completed executing the program and the picoscope is still processing. The autocorrelation is to assist in finding patterns within the wave. The autocorrelation pattern in Figure 6 was used as the baseline.

Fig. 7. Three average wave forms for 49 A’s 79 X’s and the baseline.

Fig. 8. Comparison of the autocorrelation between the various lengths of A’s and the baseline.

Fig. 9. Three average wave forms for 18 X’s 49 X’s and the baseline.

Fig. 10. Comparison of the autocorrelation between the various lengths of X’s and the baseline.

Fig. 11. Comparison of the autocorrelation between the various lengths of X’s and the baseline.

Fig. 12. Comparison of the autocorrelation between the various lengths of X’s and the baseline.

VI. Analysis

Based on the results acquired in Section V, it can be seen that as the output length increases the more power is used, more spikes are present in the graphs. This is related to the alphabet app that were discussed in Section IV, as the length increases the output array size also increases. Having a larger array more bus lines are required to change from a precharge state to a high state. This results in the fact that more power is used when the length of the output is increased.

The results show that there is a different pattern between A’s and X’s. In order to find why there is a different pattern between the two, one needs to know the hex and binary values of A and X. The hex values are 41 and 58 with the binary values being 0100 0001 and 0101 1000 for A and X. Based on the binary values it is determined that X needs more bits to be processed than that of A. Additionally, the change from 0 to 1 is more for X than A, since X has more 1’s and leads to more power being used. This is due to the microcontroller needed to use more memory and thus, in turn open more data gates. Based on the analysis mentioned above, the information leaked can be directly linked to how the interpreter on the smartcard executes the code.

VII. Conclusion

This paper investigates the power leakage of a smartcard. It is intended to answer two vital questions: what information is leaked out when different characters are used as output; and does the length of the output affect the amount of the information leak.

The investigation illustrates that as the length of the output is increased more bus lines are switched from a precharge state to a high state. This is related to the output array in the
alphabet app increasing as the length of the output increases. Furthermore, this work shows that the output for character “A” generates a different pattern than that for the character “X”. This is due to the fact that A’s and X’s need different amount of bytes to be executed since they have a different binary value. Additionally, the information leaked out can be directly linked to the smartcard’s interpreter. 

VIII. FUTURE WORK

To establish a power trace database for all characters. Once the power traces are captured, the electromagnetic (EM) traces will also be captured. Acquiring this information will aid in the investigating on how the different versions of the smartcard’s compiler and interpreter can affect the leakage of information.

ACKNOWLEDGEMENT

The authors would like to thank the department of Modelling and Digital Science at CSIR for providing funding and support.

REFERENCES


Ibraheem Frieslaar is currently perusing his PhD. in Computer Science at Rhodes University. The focus of his research is smartcards and microcontrollers.

Barry Irwin is the founder and head of the Security and Networks Research Group at Rhodes University. His research focuses on passive traffic analysis, Internet background radiation, Web-based malware and national level cyber defence.