A New Frequency-Based Side Channel Attack for Embedded Systems

by

Chin Chi Tiu

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2005

I hereby declare that I am the sole author of this thesis.

I authorize the University of Waterloo to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the University of Waterloo to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Abstract

Mobile devices such as personal digital assistants (PDA's), cell phones and pagers are becoming increasingly popular. Services provided by these Internet-enabled devices include sending emails and shopping online. These mobile devices also contain user's confidential personal information such as phonebook and credit card information. As a result, the security of these wireless embedded systems operating in hostile environments is becoming more challenging. Although confidential data can be protected using cryptographic algorithms, there have been increased concerns of the vulnerabilities of cryptographic algorithms to side channel attacks. Power analysis and EM analysis have been shown in previous research to be able to break conventional symmetric key algorithms implemented on smart cards. However, no conclusive experiments have been reported so far on the security of PDA's. This thesis investigates the threat of EM analysis on a PDA running AES encryption in order to better protect these systems from adversaries in future research.

This thesis presents for the first time conclusive EM analysis results of AES implementation on a PDA. This thesis is also the first one to propose a frequency-based side channel attack that is efficient even when traces are misaligned in experiments, whereas the previously researched DEMA fails in such condition. This thesis makes progress in side channel attacks and is important for future wireless embedded systems, which will increasingly demand higher levels of data security measures.

Results from this thesis show that the secret key can be retrieved successfully using the new frequency-based differential EM analysis. In addition, the proposed first-order frequency attack is capable of defeating the desynchronization countermeasure that randomly inserts delays.

Acknowledgements

I would like to thank my supervisor, Professor Cathy Gebotys, for all her advice, guidance and encouragement. I would also like to my family and friends for their love and support.

I greatly appreciate the generous financial support provided by Professor Gebotys through a Research Assistantship. I am also grateful for the scholarship awarded to me by the University of Waterloo.

Table of Contents

List of Figures

List of Tables

1 Introduction

Mobile devices such as personal digital assistants (PDA's), cell phones and pagers are becoming increasingly popular. Being Internet-enabled, these devices allow mobile users to send emails and even shop online. Confidential data are being exchanged wirelessly under hostile environments. As a result, the data security of wireless embedded systems is becoming more challenging. Although confidential data can be protected using cryptographic algorithms, there have been increased concerns of the vulnerabilities of cryptographic algorithms to side channel attacks.

Power analysis first introduced by Kocher *et al.* [1] is one of the powerful side channel attacks that exploit information leaked from a cryptographic device. Another powerful side channel attack is electromagnetic analysis. Power analysis and EM analysis have been shown to be able to break conventional symmetric key algorithms implemented on smart cards. However, there is a lack of research in the security of PDA's. This thesis aims to explore the feasibility of extracting the secret key of conventional symmetric key algorithms by analyzing EM signals from PDA's.

1.1 Motivation

Side channel attacks are very powerful cryptanalysis techniques because they break a cryptosystem at the implementation level. Hence, they require less computational power comparing to conventional cryptanalysis which defeats a cryptosystem at the algorithmic level. Side channel attacks allow adversaries to pull extremely small signals from noisy data, often without even knowing the design of the target system. As a result, these attacks are of particular concern for mobile devices that must protect secret keys while operating in hostile environments. However, research in the past focuses mainly on smart cards' security. There is a lack of conclusive experiments in the subject of the security of PDA's. Comparing to smart card, a PDA has a much more complex architecture. Its processor operates at a higher clock frequency. With a more complex operating system, some processes are executed in a parallel fashion. It also consists of other components such as LCD screen, radio antenna and receiver, infrared port, non-volatile memory, etc.

These components are closely located on the PDA. As a result, performing side channel attacks on PDA's is a more difficult problem, since there could be interference caused by these neighboring components when measuring EM radiation from the processor. The motivation of this thesis is to examine the threat of side channel attacks on PDA's running symmetric key algorithms in order to better protect these systems from adversaries in future research.

1.2 Problem Description

In this thesis, problems of performing side channel attack to extract the secret key of symmetric key algorithms implemented on wireless embedded systems are addressed. The first problem being addressed is the lack of conclusive experiments on the security of AES implementation on PDA's. This thesis presents results of power analysis and EM analysis on an ARM Integrator/C7TDMI core module and a PDA running the Rijndael encryption algorithm [2].

The second problem being addressed is the lack of methodologies to overcome experimental issues. There are lots of problems encountered in experiments such as environmental noise, equipment noise, etc. Among all the experimental issues, misalignment of traces is the most severe problem encountered while measuring EM signals from a PDA. If spikes are slightly out of alignment in time, they will cancel out rather than reinforced when averaging. Misaligned traces could cause large spurious peaks in a differential trace causing the previously researched DEMA to fail. This thesis addresses this severe issue by proposing a new side channel attack technique called the Differential Frequency Analysis (DFA), which does not require perfect alignment of EM traces, thus supporting attacks on complex wireless embedded systems.

The third problem being addressed is the difficulty of measuring power consumption from real embedded devices. Measuring the power drained by a processor requires tampering the device. Due to the lack of public information about the PDA under test, it is difficult to locate and access the power pins on the complex motherboard of the device to measure the power consumption of the processor. Therefore, EM analysis is the

preferred attack for the PDA since it is a non-invasive attack as it consists in measuring the near field without any modification to the PDA. This thesis solves this problem by alternatively analyzing EM radiation from the PDA to determine whether it leaks any sensitive information.

The last problem being addressed in this thesis is to examine whether some of the existing countermeasures for first order differential analysis are effective against the proposed frequency-based attack and a previously researched attack called FFT 2DPA proposed by Waddle *et al.* [13]. The desynchronization countermeasure [21] and the Split Mask countermeasure [9] are implemented for the Rijndael algorithm.

1.3 Thesis Overview

The first chapter of the thesis consists of a brief introduction, the motivation of the thesis, and the problem description. The second chapter provides some background information and presents some previous research on the subjects of power analysis, EM analysis, and Rijndael algorithm. Also, it summarizes the contribution of this thesis based on what is missing in previous research. Chapter 3 introduces the new side channel attack technique called the Differential Frequency Analysis (DFA), which is the first attack that analyzes signals in the frequency domain. Chapter 4 introduces the experimental setup and presents the experimental results from both the ARM Integrator/C7TDMI core module and the PDA. Chapter 5 discusses the experimental results, compares the DFA attack with previously researched side channel attacks, discusses the effectiveness of the DFA attack, and presents work to be done in the future. Chapter 6 concludes with the findings of this thesis.

2 Background Information and Previous Research

This chapter first provides background information on symmetric key cryptography and side channel attacks followed by previous research on the Rijndael algorithm, power analysis, and EM analysis. The chapter is concluded by outlining the contribution of this thesis to the field of side channel attacks.

2.1 Introduction to Symmetric Key Algorithms

Since 1977, the Data Encryption Standard (DES) [24] has been the most widely used symmetric key algorithm. Due to the fast development and increasing computation power of computers, DES was broken using a brute force attack in 1997. It is not secure anymore nowadays. Therefore, the National Institute of Standards and Technology (NIST) of the United States initiated the development of Advanced Encryption Standard (AES) [2], also known as Rijndael, to replace DES. This new encryption standard proposed by 2 Belgian researchers, Vincent Rijmen and Joan Daemen, has improved security over DES and fast computation performance. It is widely implemented on mobile devices. Hence, this thesis focuses primarily on the security of the Rijndael algorithm implemented on wireless devices.

Advanced Encryption Standard (AES) is a symmetric block cipher that can process data blocks of 128 bits, using cipher keys with lengths of 128, 192, and 256 bits. The basic unit for processing in AES is a byte. The algorithm's operations are performed on a two-dimensional array of bytes called the State. The State consists of four rows of 32-bit word. The number of rounds to be performed during the execution of AES is dependent of the key size. The number of rounds is 10, 12, and 14 for key sizes of 128, 192, and 256 bits of key length respectively. Each round is composed of four different byte-oriented transformations: byte substitution using a substitution table (S-Box), shifting rows of the State array by different offsets, mixing the data within each column of the State array, and adding a Round Key to the State. Figure 1 below illustrates one complete AES encryption. The transformations within rectangular box are repeated

according to the number of rounds, i.e. the cipher key size. For more details of the algorithm, see [2].

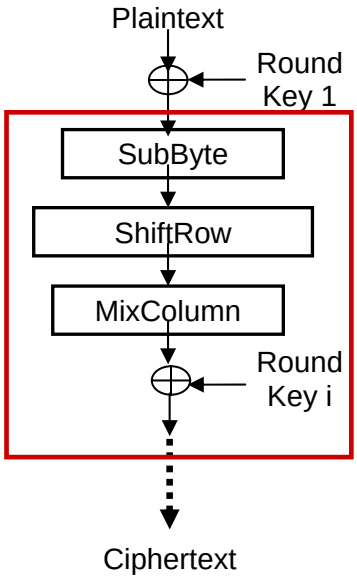

Figure 1: AES Encryption

Gladman suggest that Rijndael can be implemented very efficiently on processors with 32-bit words using tables [20]. In his implementation, the SubByte, ShiftRow, MixColumn and AddRoundKey are combined as only 1 single transformation. Five tables each of 256 32-bit words are defined to replace the original AES 8-bit S-Box. For details about how the four different byte-oriented transformations are combined as only 1 single transformation, refer to [20]. All AES test programs in this thesis are written with the optimized software implementation by Gladman.

2.2 Introduction to Side Channel Attacks

2.2.1 Information Leakages

Secret information about cryptographic devices can be revealed from analyzing power consumption and EM emanation of these devices. Most modern cryptographic devices are implemented using semiconductor logic gates, which are constructed out of transistors. Electrons flow across the silicon substrate when charge is applied to or removed from a transistor's gate, and therefore, consuming power. According to

Messerges' assumption [4], the processor will leak information about the Hamming weight of the data being processed. Processing data with higher Hamming weight will consume more power than processing data with lower Hamming weight and that this relationship is roughly linear. To measure a circuit's power consumption, a small resistor is inserted in series with the power or the ground input. The voltage difference across the resistor divided by the resistance yields the current. Therefore, the power consumption can be determined [1].

Similarly, EM emanations arise as a consequence of current flows. In CMOS devices, current only flows when there is a change in the logic state of a device. As a result, EM emanations can track number of bit transitions, revealing the Hamming weight of data being manipulated [5].

2.2.2 Power Analysis and Electromagnetic Analysis

Simple Power Analysis (SPA), Differential Power Analysis (DPA), Simple Electromagnetic Analysis (SEMA), and Differential Electromagnetic Analysis (DEMA) are side channel attacks that enable extraction of a secret key stored in cryptographic devices. The attacker monitors the power consumption or the EM emanation from such cryptographic devices, and then analyzes the collected data to extract the key. These side channel attacks aim at vulnerabilities of implementations rather than algorithms which make them so powerful since adversaries are not required to know the design of the target system.

Simple power analysis (SPA) [1] is a technique that involves directly interpreting power consumption measurements collected during cryptographic operations. No statistical analysis is required in such attack. SPA can yield information about a device's operation as well as key material. It can be used to break cryptographic implementations in which the execution path depends on the data being processed.

Similarly, in a SEMA attack [3], an adversary is able to extract compromising information from a single EM sample. If a computation makes use of conditional

branches based on secret information, then on a compromising EM signal, this can be observed as relative shifts in the distances between major computational structures. In some cases, these shifts may be sufficient to reveal the branch taken, which in turn confirms the value of the secret information. This is analogous to what has already been demonstrated for simple power analysis. Thus, conditional statements in the code could provide valuable opportunities for both SPA and SEMA.

In a differential power analysis (DPA) attack [1], the adversary monitors the power consumed by the cryptographic devices, and then statistically analyzes the collected data to extract the key in contrary to SPA. In a first order DPA attack, the attacker monitors power consumption signals and calculate the individual statistical properties of the signals at each sample time. More specifically, the attacker identifies some intermediate value in the cryptographic computation that is correlated with the power consumption and dependent only on the plaintext and some small part of the key. A collection of power traces are then gathered throughout a series of encryptions of different plaintexts. Next, the attacker will divide the traces into groups according to the intermediate value predicted by current guess at the key and the trace's corresponding plaintext. If the averaged power trace of each group differs noticeably from the other, it is likely that the current key guess is correct. Incorrect key guesses should result in all groups having very similar averaged power traces, since incorrectly predicted intermediate value will not be correlated with the measured power traces. Figure 2 demonstrates steps involved in a DPA attack.

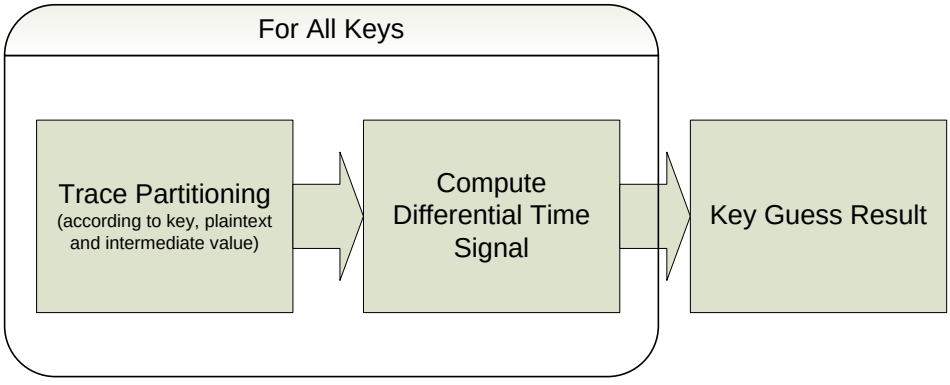

Figure 2: Differential Power Analysis (DPA) Overview

In a higher order DPA attack [1], the attacker calculates joint statistical properties of the power consumption at multiple sample times. One drawback to high-order DPA is increased memory and processor requirements because of the need to store multiple samples for a single DPA computation. Knowledge of the encryption algorithm and specific implementation is more critical in high-order DPA than first order. In most cases, the attacker needs to know specific points of execution where joint statistics can be meaningfully computed.

DEMA is the analogy for DPA [3]. In this attack, instead of monitoring the power consumption, the attacker monitors the electromagnetic emanations from the cryptographic devices, and then same statistical analysis as DPA is performed on the collected EM data to extract secret parameters.

2.3 Previous Research on Attacks on Embedded Systems

This purpose of this section is to present some previous research related to the subjects discussed in this thesis.

2.3.1 Previous Research on SPA and DPA

Power analysis, including SPA and DPA, was introduced by Kocher at al. in [1]. This paper describes specific methods for analyzing power consumption measurements to find secret keys from tamper resistant devices. They had successfully measured the power consumption of a DES operation on a smart card. Their experimental results showed that SPA can reveal the sequence of instructions executed. The 16 DES rounds are clearly visible from a SPA trace that they have captured. In addition, the authors presented experimental results on DPA of DES implementation. They had successfully extracted the secret key used in the DES encryption algorithm. Kocher *et al.* illustrated that DPA allows adversaries to extract secret information without even knowing the design of the target system. Goubin *et al.* later presented a more detailed DPA attack methodology in [7]. The authors also showed a SPA trace where one can see distinctively 16 rounds of DES computation. They also published experimental results on DPA of DES on a typical smart card. However, as one can see, the security of the newly developed encryption

standard, AES, against power analysis is never discussed in both literatures ([1] and [7]). In [10], Golić described a DPA attack methodology on AES. However, no real power measurements were presented in his work.

2.3.2 Previous Research on SEMA and DEMA

The past research has been mainly on power analysis. EM analysis also needs to be understood. Quisquater *et al.* first discussed EM analysis on smart cards in [5]. They proposed that a processor can leak information by different ways; not only by power consumption but also by electromagnetic radiation. The authors developed the continuation of Kocher's ideas by measuring the field radiated by the processor. They proposed that for a non-intrusive attack, EM analysis can be more precise than power analysis. The authors also suggested that EM analysis is strongly dependent on the architecture of the chip, and the knowledge of the internal circuitry of the processor facilitates the work. To measure the EM radiation, they used a simple flat coil so the variations of the electromagnetic field induce a current at the bounds. The sensor is placed under the smart card in the very close field. Again, no real experiments of SEMA or DEMA were put into practice, so no results were presented in their work.

Gandolfi *et al.* later reported conclusive EM analysis results in [11]. They used tiny hand-made probes, solenoids made of a coiled copper wire of outer diameters varying between 150 and 500 microns, for their EM measurements. From their experimental findings, they had pinpointed that the CPU radiates the most informative signal, in other words, the CPU is the most data-dependent component. The authors also stressed the importance to perform measurement as closely as possible to the chip. They suggested to decapsulate a chip since decapsulation offers 2 important advantages. First of all, the probe's coil can be lowered so as to touch the passivation layer and thereby capture the highest possible field once the chip is bare. Secondly, the chip becomes optically visible and its specific blocks can be pinpointed more accurately. In this work, DEMA results of DES from an 8-bit CMOS microcontroller were presented. Note that they did not perform decapsulation to the chip of the CMOS controller in this particular experiment. However, they were still able to retrieve the secret key of the DES

encryption algorithm. In addition, Gandolfi *et al.* compared DEMA results with DPA results in their paper. According to their experimental findings, although more noisy, EM measurements yield better differentials than power signals. DEMA's signal-to-noise ratio was higher than that of DPA. The correct guess identification was easier, as there were no false alerts due to erroneous peaks.

Agrawal *et al.* presented results illustrating various types of EM emanations in [3]. According to them, there are two categories of EM emanations: direct and unintentional emanations. Direct emanations result from intentional current flows, whereas unintentional emanations are caused by coupling effects between components in close proximity. Nonlinear coupling between a carrier signal and a data signal results in the generation and emanation of an amplitude modulated (AM) signals and also angle modulated (FM) signals. The authors suggested that exploiting unintentional emanations can be easier and more effective than trying to work with direct emanations. They also suggested that AM demodulated signals contain much more information. A useful rule of thumb is to expect strong carriers at odd harmonics of the clock. According to Agrawal *et al.*, EM signals propagate via radiation and conduction. All EM emanations are measured either in the near field or in the far field, both away from the smart card unlike [5] and [11]. Radiated signals are best captured by placing near field probes or antennas made of small plate of a highly conducting metal like silver or copper as close as possible or at least in the "near field" to the device, i.e. no more than a wavelength away. Capturing conductive EM emanations requires current probes similar to those used for power analysis and subsequent signal processing to extract them from the stronger signals. They had discovered that apart from the relatively low frequency, high amplitude power consumption signal, there are faint higher frequency AM modulated carriers representing conductive emanations as well. In this work, unlike [5] and [11], they had successfully demonstrated DEMA attacks of DES on smart cards by AM demodulating the raw EM signal at different intermediate carrier frequencies (harmonics of the clock frequency).

Most research on side channel attacks so far focus mainly on DES implementation. Experimental results on DEMA on AES were finally presented by

Carlier *et al.* in [12]. They also launched the attack on FPGA's instead of smart cards. No decapsulation was performed on the FPGA. They used solenoid wires of copper consisting of a dozen of spires with a diameter of approximately 1 mm for their EM measurements. They placed the probe as close as possible to the FPGA to increase the magnetic flux collected by the probe. It is also interesting to note that all bytes are processed in parallel in FPGA's. According to the findings of Carlier *et al.*, for a specific probe position, only specific bits leakage can be detected. So, the bias spike in the DEMA signal can disappear if they modify the specific bit attacked in their partition function. This phenomenon can explain why EM analysis is not disturbed by the parallel computation effect, unlike with power measurements. Their results showed the effectiveness of EM analysis against AES on FPGA.

As one can see from all previous research discussed so far, the main focus was in the vulnerability of DES implementation on smart cards, 8-bit processors, and FPGA's. Until now, the security of AES implementation on 32-bit processors and PDA's are never studied.

2.3.3 Previous Research on Side Channel Attack Countermeasures

Many countermeasures have been proposed in the past to protect symmetric key algorithms implementation against power analysis and EM analysis attacks. Such countermeasures fall into 2 categories: signal strength reduction and signal information reduction. First of all, an example of a signal strength reduction countermeasure can be the use of shielding [5] to reduce the strength of compromising signals available to an attacker.

Secondly, examples of signal information reduction countermeasures use mainly randomization techniques in computation in order to substantially reduce the effectiveness of statistical attacks using the available signals. Many randomization techniques have been proposed for securing conventional symmetric key algorithms, such as DES and AES, against side channel attacks. Daemen *et al.* proposed the Desynchronization countermeasure [21] for Rijndael. The main idea behind this

countermeasure is if the sequence of instruction is not fixed but changes from cipher execution to cipher execution, e.g. by inserting dummy instructions based on some modifying parameter, the DPA attack no longer works. Chari *et al.* proposed a Secret Splitting technique in which data is divided into k shares [6]. A similar Duplication Method was proposed as a particular case by Goubin and Patarin [7]. Furthermore, Messerges introduced the Masking Method which involves masking the secret key by XORing with a random mask [4]. Itoh *et al.* proposed the Fixed Value Masking Method [8]. This is an improved countermeasure of Messerges' Masking Method. With this method, the encryption process is faster and less RAM size is required. This is achieved by randomly choosing one mask value from a fixed set of mask values previously prepared and stored in the ROM. Gebotys *et al.* presented a low energy masking countermeasure for symmetric key in [9] which avoids large overheads of table regeneration or excessive storage unlike [4] and [8]. Although there exist a large number of protection mechanisms, most of the above countermeasures have never tested against side channel attacks experimentally.

2.3.4 Previous Research on High Order Side Channel Attacks

Countermeasures that prevent first order attacks may not be effective against higher order attacks. The effectiveness of high order attacks also needs to be put in practice. In [4], Messerges first presents a second order attack on the masking countermeasure. He launched the attack on an ST16 smart card. His findings draw attention to the powerfulness of higher order DPA. In high order DPA, knowledge of the encryption algorithm and specific implementation is more critical than first order. The attacker needs to know specific points of execution where joint statistics can be meaningfully computed. One drawback to high-order DPA is increased memory and processor requirements because of the need to store multiple samples for a single DPA computation.

To overcome the increased memory and processor requirements, Waddle *et al.* propose a more efficient second-order power analysis in [13]. Two attacks, Zero-Offset 2DPA and FFT 2DPA, are presented in their work. Their work is able to defect the masking countermeasure while minimizing computation resource requirements in terms

of space and time. There is no need to obtain power measurement at multiple sample times. Once again, no real measurements are presented by the authors.

2.3.5 Previous Research on PDA's

Kingpin *et al.* reported security analysis on PDA's in [33]. In their work, the threat of malicious code and virus attack was investigated. The authors provided a summary of the various types of malicious code: viruses, Trojan horses, and worms. They also detailed the risks of weak system password storage and backdoor debug modes inherent in Palm OS. However, attacks presented in [33] are specific to the Palm operating system (OS) software and hardware platform. PDA's security against side channel attacks is not reported in this literature.

2.4 Contribution of Thesis

From all the previous research presented in this chapter, one can see that research in the past focuses primarily on the security of smart cards', 8-bit processors, and FPGA's. No research has been done to study the threat of side channel attacks on 32-bit processors and PDA's. As mobile devices are becoming more and more popular, attacks on these PDA's are big concerns and certainly need to be addressed. Furthermore, even though confidential data on these embedded systems are protected using conventional symmetric key algorithms, numerous researches have already broken the DES implementation using both power analysis and EM analysis. Again, most conclusive experimental results presented so far are attacks on DES implementation. As AES becomes more widely implemented, its security also needs to be addressed. Therefore, the contribution of this thesis is to investigate the threat of EM analysis on PDA's in order to better protect these systems from adversaries in future research. This thesis presents for the first time EM analysis results of AES implementation on an ARM Integrator/C7TDMI core module a PDA both with 32-bit processors.

It is also important to note that experimental issues such as noise and equipment limitation are never discussed in all previous research. No methodology has been proposed to overcome these experimental issues. This thesis is also the first one to

address experimental issues encountered in PDA experiments. Since PDA has a more complex circuitry and operating system than smart cards, there is a problem when capturing EM traces on the oscilloscope where most EM traces measured are temporally misaligned. The contribution of this thesis is to address the severe issues of trace misalignment on PDA experiments. This thesis is the first one to propose a side channel attack called the Differential Frequency Analysis (DFA), which does not require perfect alignment of EM traces, thus supporting attacks on real embedded systems. This thesis also compares the characteristics of EM emanation with power consumption using the ARM Integrator/C7TDMI core module. The Differential Frequency Analysis (DFA) can be applied to both power analysis and EM analysis.

As one can see that most countermeasures proposed in the past are rarely applied to the AES implementation, the effectiveness of these countermeasures needs to be put in practice. Thus, this thesis implements the desynchronization countermeasure [21] and the Split Mask countermeasure [9] on the Rijndael algorithm. The effectiveness of these 2 countermeasures against the proposed frequency-based attack (DFA) and a previously researched high order attack called FFT 2DPA proposed by Waddle *et al.* [13] is examined.

3 Differential Frequency Analysis

The purpose of this chapter is to introduce a new side channel attack on Rijndael encryption called Differential Frequency Analysis (DFA). This chapter first introduces why this attack is being pursued. Subsequently, attack methodology and theory are presented. The last section of this chapter compares the attack methodology of DFA with previously researched side channel attacks: DEMA, DSA, and FFT 2DPA.

3.1 Introduction

This thesis proposes a new side channel attack called Differential Frequency Analysis (DFA). This technique is a modified version of Kocher's differential power analysis [1]. Instead of computing the differential signals in the time domain, this technique is performed in the frequency domain by calculating the differential power spectral density (PSD) signal. The reasoning of analyzing signals in the frequency domain is that sometimes EM or power traces captured are temporally misaligned. As a result, differential electromagnetic analysis (DEMA) or differential power analysis (DPA) fails. When spikes are slightly out of alignment in time, they will cancel out rather than reinforced when averaging. On the other hand, the proposed DFA is efficient in retrieving the secret key successfully even the problem of trace misalignment is present. In addition, it is shown experimentally by Agrawal *et al.* in [3] that the Fast Fourier Transform (FFT) of EM signals contain useful signal information suggesting the validity of analyzing signals in the frequency domain. The side channel attack presented in this thesis is intended to resolve the difficulty of performing first order differential analysis on real embedded systems where uncorrelated temporal misalignment of traces is a big concern. Essentially, the Differential Frequency Analysis can be applied on both EM and power traces. For EM analysis, the attack is called the differential EM frequency analysis (DEMFA). As for power analysis, the attack is called the differential Power frequency analysis (DPFA).

3.2 EM and Power Side Channels

The security of an ARM Integrator/C7TDMI core module and a wireless PDA are examined in this thesis. Before presenting the DFA attack methodology, this section discusses the side channels available from these 2 devices under test.

Both power and EM side channels are available from the ARM Integrator/C7TDMI core module. The power consumption of the ARM processor core is measured in the form of current in this thesis. Since the amount of voltage drawn on the ARM processor core is 3.3V, the current consumption can be easily converted to power consumption by multiplying the current with the supply voltage. It is easy to measure the power consumption of the ARM core processor since the evaluation board provides a number of easily accessible test points that can measure the current drawn by the ARM7TDMI processor core. EM signals can also be obtained from the processor by simply placing an EM probe on top of the processor for best signal quality.

Comparing to the ARM Integrator/C7TDMI core module, the PDA has a much more complex architecture. Its processor operates at a higher clock frequency. It also consists of other components such as LCD screen, radio antenna and receiver, non-volatile memory, etc. Due to the lack of public information about the PDA under test, it is difficult to locate the power pins on the complex motherboard of the device to measure the power consumption of the processor. In addition, measuring the power consumed by the processor requires modifying the device under test. Therefore, EM analysis is the preferred attack for the PDA since it does not require any modification to the PDA. Therefore, only the EM side channel is available from the PDA.

3.3 Methodology

This section introduces the attack methodology of Differential Frequency Analysis (DFA). Once the power consumption or EM emanation are captured from the device under test, they are processed and analyzed to extract secret information. To perform Differential Frequency Analysis (DFA), an attacker first observes n AES encryptions and capture $T_{1...n}[1...m]$ EM or power traces containing m sample points each. In addition, the

adversary keeps track of the plaintexts $P_{1...n}$. No knowledge of the ciphertext is required; this is a known-plaintext attack. Figure 3 gives an overview of the DFA attack. Unlike Figure 2, a pre-processing step is needed in this new frequency-based attack.

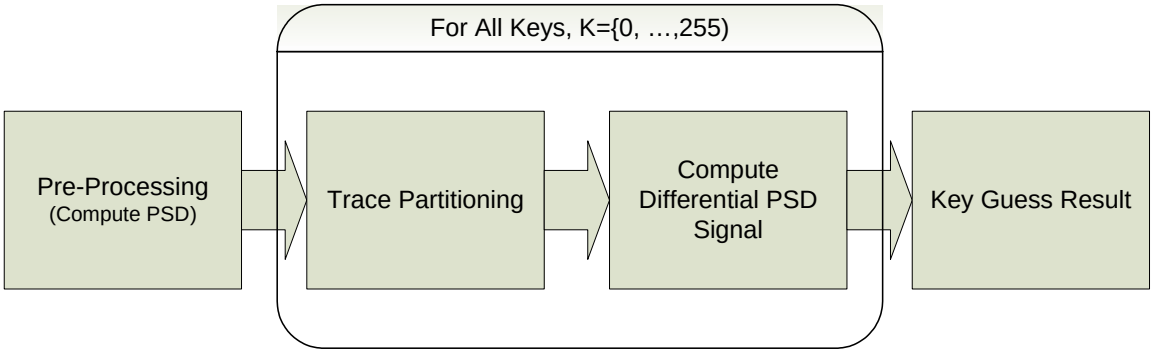

Figure 3: Differential Frequency Analysis (DFA) Overview

3.3.1 Pre-Processing Stage

The DFA attack requires a pre-processing stage to transform signals from time domain to frequency domain. In this stage, it involves calculating the power spectral density (PSD) for all traces. To calculate the power spectral density of a trace, first perform Fast Fourier Transform (FFT) on the trace *T*. Then the squared £$_2$–norm or the complex conjugate of the complex number, FFT(*T*), is computed.

3.3.2 Trace Partitioning

After the pre-processing stage, partition the PSD of traces $T_{1...n}[1...m]$ according to a selected bit, *b*, at the output of one of the S-Boxes in the first round of AES encryption. For each trace, group the trace that has bit *b* equal to 0 to subset 0 and vice-versa to subset 1 according to its plaintext and the key guess. The DFA partition function *D(P, b, K)* is defined as computing the value of bit $0 \leq b < 8$ of the S-Box output in the first round of AES encryption for plaintext *P*, where the 8-bit key entering the S-Box corresponding to bit b are represented by $0 \leq K < 2^8$. Figure 4 illustrates how traces are grouped in the attack.

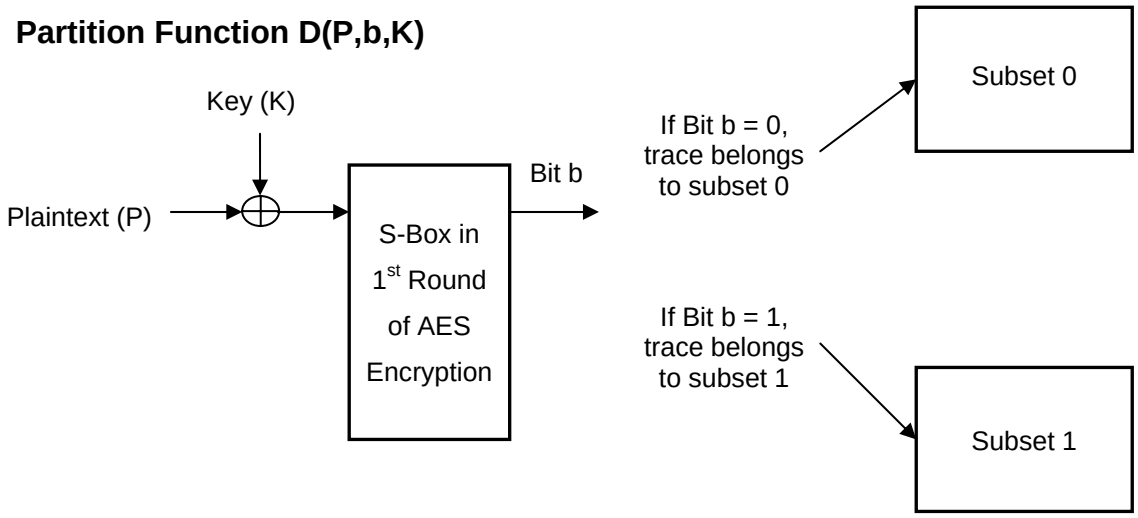

Figure 4: Trace Partitioning in 1st Round of AES Encryption

3.3.3 Computing Differential Power Spectral Density Signal

Now, the main component of this new frequency-based differential analysis begins. In this step, first compute the average of PSD for both subsets. Then, calculate the differential PSD signal by subtracting the averaged PSD of subset 0 from the averaged PSD of subset 1. Note that this differential PSD signal is computed in a similar way as the normal differential time signal found. The only difference is that the frequency domain signals are used instead of the raw time domain signals. Then for each frequency, *f*, of the differential PSD signal, sum up all the spikes that are bigger than 2 times of the standard deviation threshold signal (2*STD_R).

To quantitatively decide whether the differential PSD signal is significant, it is important to pick a threshold signal. A good threshold signal would be a constant multiple *k* of the standard deviation of means of power spectral density of subset 0 and subset 1 (STD_R). This threshold signal is served to minimize the impact of false spikes and to reduce the probability of error. The STD_R is a measure of dispersion of a set of traces from its mean. If the STD_R is high at some point time, it means that there is possibly more noise in the traces at this particular time. Therefore, by comparing the differential PSD signal with 2 times of the STD_R threshold signal, one can evaluate if a spike in the differential PSD signal is significant while minimizing the impact of false

peaks. Figure 5 below illustrates the total area outside the 2*STD_R region for a correct and a wrong key guess.

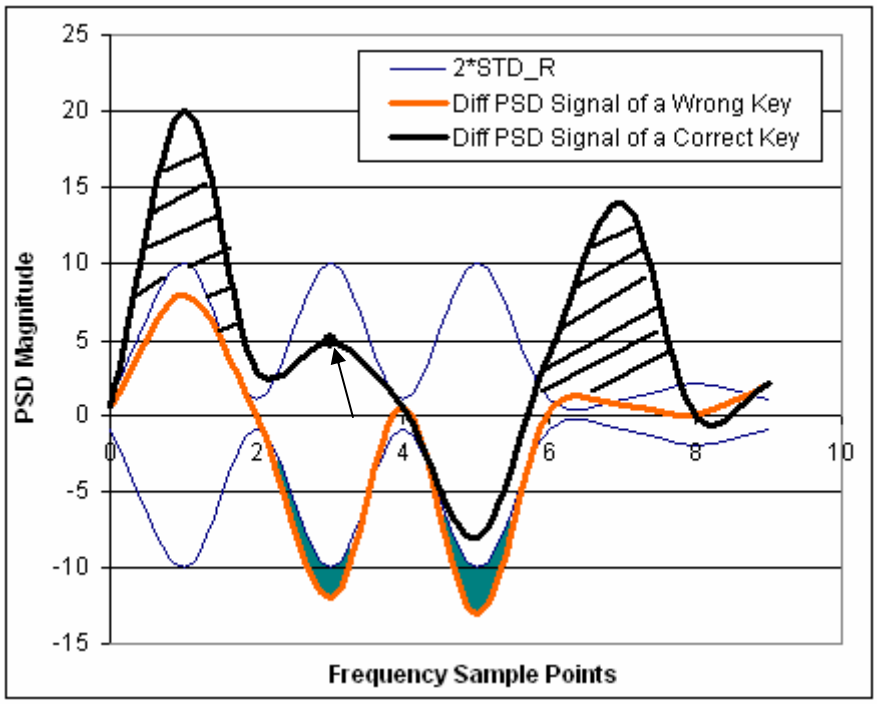

Figure 5: Comparing Differential PSD Signal with 2*STD_R

In Figure 5, the area shaded with lines represents the amount that the differential signal of a correct key guess exceeds the threshold signal (2*STD_R). And the solid shaded area represents that for an incorrect key guess. Although both differential signals have significant peaks, the area of spikes outside the range of 2*STD_R for the correct key guess is much greater than that of the wrong key. To illustrate how the threshold signal can minimize the impact of false peaks, let us take a closer look at a false peak pointed by an arrow in the diagram. It is interesting to note that this peak of the correct differential signal at point 3 is not significant because it is smaller than the standard deviation threshold signal. The peak in the threshold signal at point 3 implies that the EM traces are noisy at this particular point. As a result, by comparing the differential signal with the STD_R threshold signal, one can minimize the impact of false peaks. The pseudo-code of the DFA attack methodology is presented next.

i = trace number, $i \in \{0,\ldots,n-1\}$

b = set number, $b \in \{0,1\}$

T_i^b = EM signal of set b and trace i

STD_R = standard deviation of means of PSD (threshold signal)

K = key of AES encryption

$sumPeak$ = vector of spikes outside $2 * STD\_R$ for all 256 keys

t = time, $t \in \{0,\ldots,m-1\}$

f = frequency points in power spectral density, $f \in \{0,\ldots,\frac{m}{2}-1\}$

PowerSpectralDensity*(T)*

1 : for each $i, i \in \{0,\ldots,n-1\}$:

2 : $F_i \leftarrow FFT(T_i)$

3 : $P_i \leftarrow \left|F_i\right|^2$

4 : return P

STD_DOM(P^0, P^1)

1 : $STD^0 \leftarrow STD(P^0)$

2 : $STD^1 \leftarrow STD(P^1)$

3 : $u^0 \leftarrow Size(P^0)$

4 : $u^1 \leftarrow Size(P^1)$

5 : return $STD\_R \leftarrow \sqrt{\frac{\left|STD^0\right|^2}{u^0} + \frac{\left|STD^1\right|^2}{u^1}}$

DFA(T)

1 : $P \leftarrow PowerSpectralDensity(T)$

2 : for each $K, K \in \{0, \cdots, 255\}$

3 : $\{P^0, P^1\} \leftarrow partitionTraces(P)$

4 : $STD\_R \leftarrow STD\_DOM(P^0, P^1)$

5 : $Diff \leftarrow Mean(P^0) - Mean(P^1)$

6 : $sumPeak(key) \leftarrow 0$

7 : for each $f, f \in \{0,\ldots,\frac{m}{2}-1\}$:

8 : if $(abs(Diff(f)) > 2 * STD\_R(f))$

$sumPeak(K) \leftarrow sumPeak(K) + (abs(Diff(f)) - 2 * STD\_R(f))$

9 : return $sumPeak$

3.3.4 Key Guess

If the key guess *K* is incorrect, the bit computed using the partition function *D* will differ from the target bit for about half of the plaintexts P_i. The partition function is thus effectively uncorrelated to what was actually computed by the target device. If a random function is used to divide into 2 subsets, the difference in the averages of the subsets should be negligible and should be smaller than the range of 2 times of the standard deviation threshold region.

On the other hand, if the key guess *K* is correct, the computed value for *D* will equal to the actual value of the target bit b with probability 1. The partition function is thus correlated to the value of the bit manipulated in the first round of AES encryption. There should be significant differences between the average of PSD for the subset 0 and subset 1. One would expect to see more spikes outside the range of 2*STD_R.

There are only 2^8 or 256 possible keys for an 8-bit S-Box in AES. By comparing the differential PSD signal of all 256 possible keys, the correct key should have the most significant differential PSD signal among all. The correct key *K* can thus be identified since it should have the largest *sumPeak(K)* value calculated in *DFA()* among all 256 key values.

Another advantage of DFA is the possibility of reducing the key search space. For instance, a brute-force attack is impossible for an AES 128-bit key since there are 2^{128} possible key searches. On the other hand, the Differential Frequency Analysis is only performed on each of the 8-bit S-Boxes. For AES 128-bit, there are 16 S-Boxes in total. Therefore, the key search space is reduced to 16 * 256 = 4096.

3.4 Theory

This section presents the theory that justifies the proposed Differential Frequency Analysis attack. The causes and effects of temporal misalignment of traces are also discussed.

3.4.1 Assumptions

There are a few important assumptions made in the theory of DFA. The first assumption is that all AES encryption algorithm executes in constant time, regardless of the values of the plaintexts and the master key. If this is not the case, then the attack would be very difficult to accomplish. The second assumption is that the attacker has knowledge and control of the input plaintexts for the Rijndael encryption. The third assumption is that there is no distortion in the shape of all waveforms in experimental measurements. All EM traces captured in this thesis are only temporally shifted due to reasons to be discussed in the following section. The forth assumption is that averaging also helps to reduce noise from the chip, environment and measurement equipments.

3.4.2 Temporal Misalignment of Traces in Experimental Results

Performing first order differential analysis on real embedded systems is always difficult where uncorrelated temporal misalignment of traces is a big concern. This section describes the causes and effects of misalignment of traces in experiments. Note the misalignment problem is specific to the PDA used in this thesis. Other embedded systems might suffer from different experimental problems.

3.4.2.1 Causes of Misalignment of Traces in Experiments

The architecture of the device under test can cause various experimental problems. It is observed that the experiments on the ARM Integrator/C7TDMI core module have negligible amount of trace misalignment. Averaging a large number of traces could eliminate such problems. However, the experiments on the PDA suffer seriously from temporal misalignment of traces where averaging a large number of traces is no longer effective. The Java architecture, the trigger mechanism for the oscilloscope, and noise are the three main sources causing temporal misalignment in experiments.

First of all, there is a restriction that all the programs on the PDA must be written in Java, a high-level programming language. The final program is not as efficient as that written in a low-level programming language such as the assembly language used for the ARM Integrator/C7TDMI core module processor. Since a single low-level language instruction translates into a single machine-language instruction, whereas a high-level language instruction typically translates into a series of machine-language instructions. As a result, Java is much slower than the assembly language.

Java consists of three components: the Java language, the Java Virtual Machine (JVM), and the Java API (Application Programming Interface). The Java Virtual Machine can be seen as an abstract computer. It is implemented in software on top of the hardware platform and operating system. Java programs are compiled for the JVM instead of the system. Programmer writes a Java program and the Java compiler translates that into codes that the JVM implements. These codes are called bytecodes. Bytecodes can be thought of as the machine language for the JVM. The JVM interprets a stream of bytecodes as a sequence of instructions. These instructions are then executed by the hardware and OS to generate the desired output [14]. Figure 6 shows the steps involved in executing a Java program.

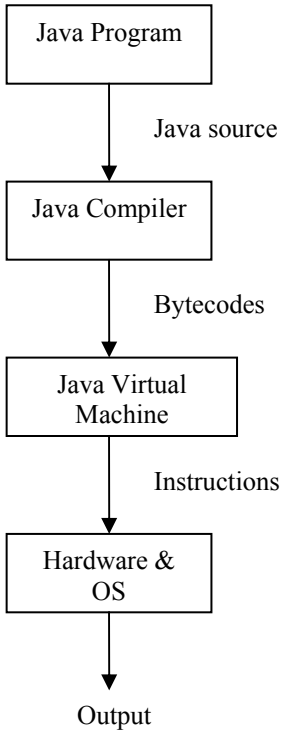

Figure 6: Java Program Execution

As illustrated in Figure 6, it takes several steps to interpret a Java program to execute and then generate the desired output. Due to the complex and unpredictable nature of the Java run-time environment, hardware and software interrupts may occur while the AES encryption test program is running on the PDA. Process switching may also occur while the encryption test program is running. Therefore, EM emanations due to these background operations of the PDA would occur at different times in each run of the Rijndael encryption causing delay when executing a Java program. As a result, a Java program may be started after an unknown period or after a very long delay causing traces to misalign in one acquisition. Also, a Java program may be randomly interrupted by another higher priority process, such as garbage collection, creating unwanted information in the EM or power traces captured. To remove the distortions created by garbage collection, all the experiments for this thesis force the PDA to perform garbage collection before all AES encryption to ensure that garbage collection is not needed during the computations. This is done by calling the System.gc() function supported by the java.lang package which the Java Virtual Machine recycles unused objects to free up memory for quick reuse before running the AES test program.

Secondly, the problem of misalignment of traces may also be caused by the trigger signal to the oscilloscope. The trigger signal is generated by turning the light emitting diode (LED) of the PDA ON and OFF. The LED is controlled by the Java API supported by the PDA vendor. The Java API is the set of classes included with the Java Development Environment. These classes are written using the Java language and run on the JVM. Once again due to the complex and unpredictable nature of the Java run-time environment, the Java API could also be delayed occasionally causing jitters in the trigger signal for each frame. Other trigger mechanisms such as: activating the vibration mode of the PDA and writing data to the USB port of the PDA, are also exploited. However, they give no improvement over the alignment of the traces. As a result, the LED of the PDA is chosen as the trigger signal to the oscilloscope.

Thirdly, noise from the device under test, the testing environment and measurement equipments may also cause traces to misalign. For instance, due to the compact size of the PDA, EM signals captured from the processor might contain noise from other chips near by. On the other hand, the limited resolution of the oscilloscope also introduces quantization noise. One method to eliminate noise is to have as many traces as possible. However, the number of traces measured is limited by the scope memory. Also, since the AES program in Java runs at least hundred times slower than the AES assembly program, a longer frame would thus sacrifice the number of traces acquired due to the fixed scope memory size. For detailed explanations about how the scope memory affect the number of traces acquired, see Section 4.1.3.

3.4.2.2 Effects of Trace Misalignment in Experiments

As discussed in the previous section, temporal misalignment is a difficult experimental problem for DEMA. A high-level programming language and a temporally shifted trigger mechanism and noise could cause EM curves to misalign within a same acquisition set affecting DEMA to fail.

Figure 7 below shows two aligned EM traces acquired from the ARM Integrator/C7TDMI core module. For further details of the device under test, see Section

4.1.1. The purpose of this figure is to show the importance of data alignment in a DEMA attack. To illustrate that they have good alignment, 2 traces performing the same operation are acquired. Figure 7a and Figure 7b are both a single EM trace captured while running the Rijndael encryption algorithm with the same key and the same plaintext. Figure 7c is obtained by subtracting Trace 1 in Figure 7a from Trace 2 in Figure 7b. Since these 2 traces are performing exactly the same operation, it is expected that the EM emanation of both traces are approximately the same. The subtraction of these 2 traces is expected to be zero. As shown in Figure 7c, the plot of their subtraction is close to zero at every point. Note that it will not be exactly zero due to minor variations in the traces caused by noise. Therefore, it is demonstrated that these 2 EM traces have good alignment.

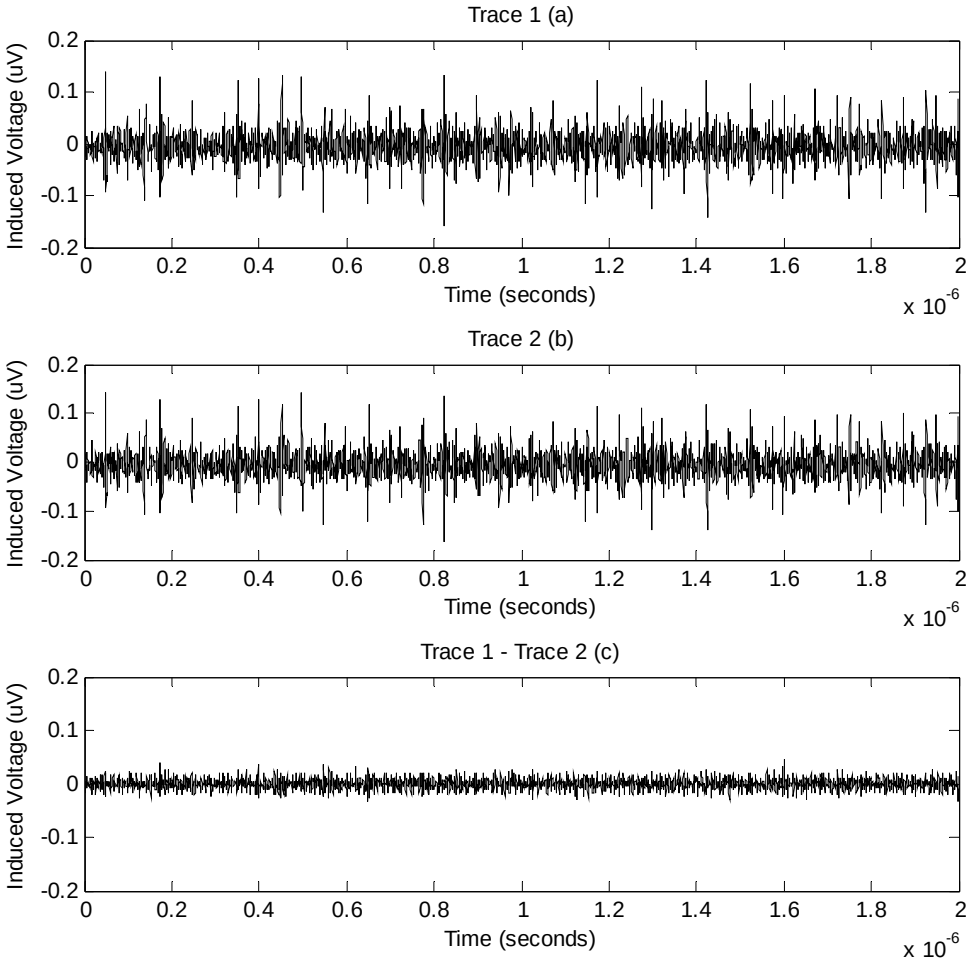

Figure 7: Two Perfectly Aligned EM Traces (a & b), Trace 1 Minus Trace 2 (c)

Figure 8 below shows the acquired DEMA signals of 2976 EM traces measured from the ARM Integrator/C7TDMI core module. Since all the traces captured in this acquisition set are perfectly aligned, there is a clear spike in the differential signal at time 0.8 us. The secret key is retrieved successfully using this set of EM traces.

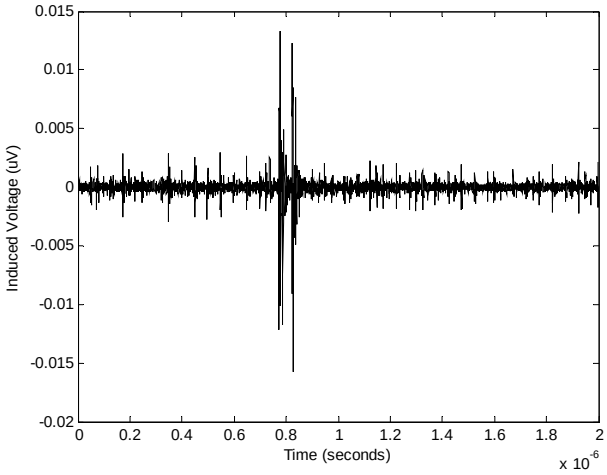

Figure 8: Differential Time Signal of Perfectly Aligned EM Traces

Now let us examine the negative effect of data misalignment. Figure 9 shows the same test as Figure 7 but executed on the PDA. For further details of the PDA, see Section 4.3.1. Although the shape of both traces is similar, one can notice that they are shifted in time. To better illustrate that they are not aligned, 2 traces performing the same operation are acquired. Figure 9a and Figure 9b are both a single EM trace captured while running the Rijndael encryption algorithm with the same key and the same plaintext. Figure 9c is obtained by subtracting Trace 1 in Figure 9a from Trace 2 in Figure 9b. Since these 2 traces are performing exactly the same operation, it is expected that the EM emanation of both traces are approximately the same. However, it is not the case in this particular test since these 2 traces are severely misaligned. As indicated in Figure 9c, the plot of their subtraction is not zero anymore.

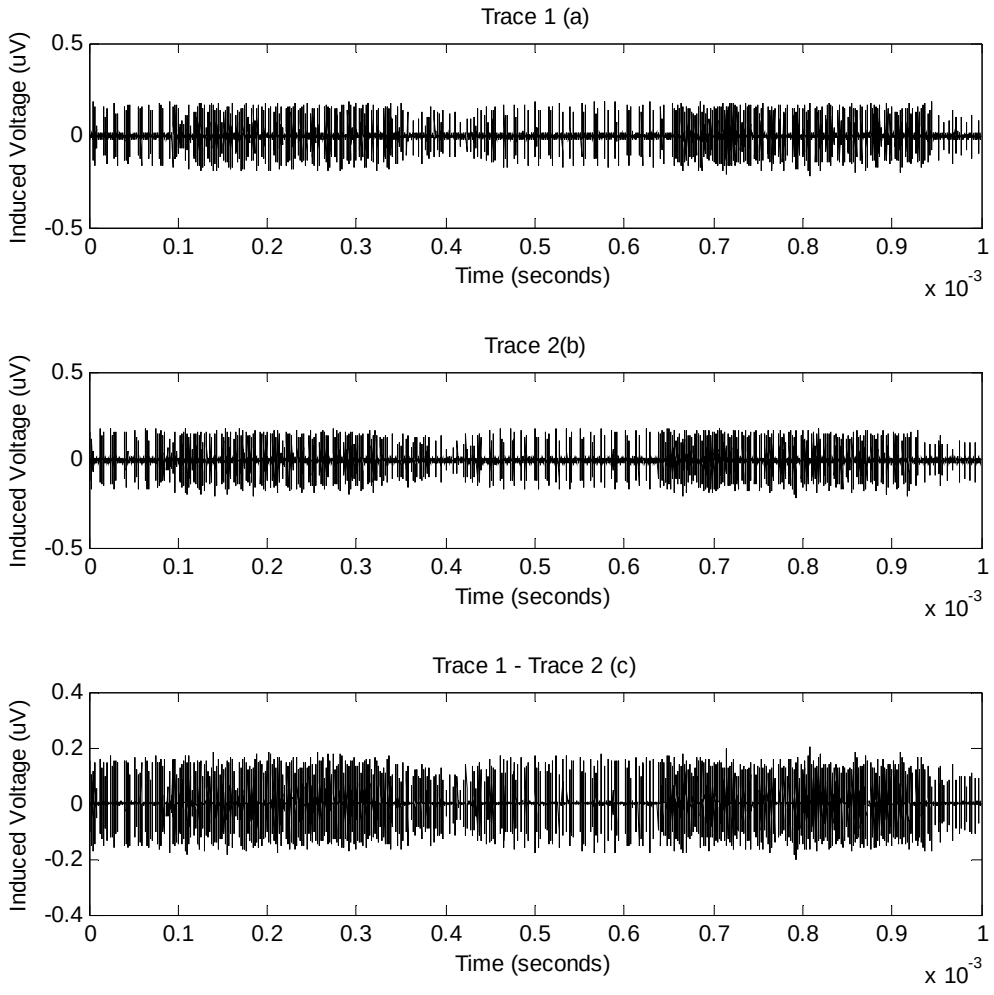

Figure 9: Two Misaligned EM Traces (a & b), Trace 1 Minus Trace 2 (c)

It is clear that if spikes are slightly out of alignment in time, they will cancel out rather than reinforced when averaging. It is observed that the amplitude of the subtraction in Figure 9c is much bigger than that in Figure 7c. Note the spurious spikes in the differential signal shown in Figure 10 below. It is not possible to determine the key information from the misaligned EM traces. Unlike Figure 8, no significant spike is found in the differential signal. As a matter of fact, the power or EM spikes analyzed in DPA or DEMA can be as small as 5 sample points wide, so a misalignment of 1 or 2 sample points can already cause significant loss of information when traces are averaged together.

As a result, DPA or DEMA could fail because of the negative effect of trace misalignment problem.

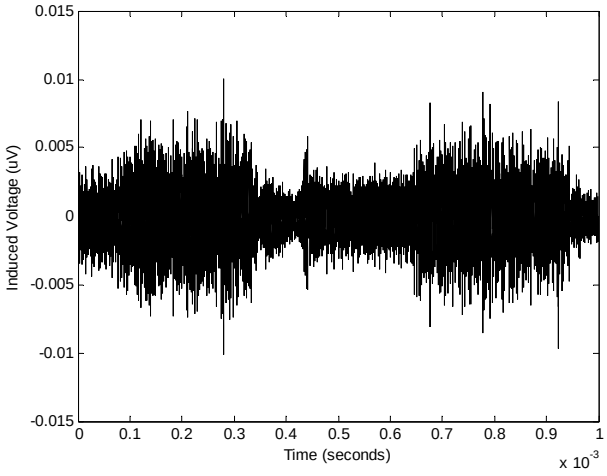

Figure 10: Differential Time Signal of Misaligned EM Traces

3.4.3 Eliminating Trace Misalignment Using Power Spectral Density

The previous sections have discussed about the causes and effects of temporal misalignment. In practice, temporal misalignment causes false peaks to be observed when performing DPA or DEMA. One possible solution to align the EM traces is to use a better trigger mechanism to the oscilloscope. However, with limited information on the PDA specifications, it is not possible to find another trigger mechanism that is better than the LED. Another solution to the problem is to use signal processing techniques after the acquisition of traces.

Previous literature stated that temporal misalignment is a significant problem, but did not give any specific methods to align traces. The frequency-based side channel attack, DFA, presented in this thesis is intended to overcome the problem of performing first order differential analysis on real embedded systems where uncorrelated temporal misalignment of traces is a big concern.

The essence of the Differential Frequency Analysis (DFA) is based on the time shifting property of Discrete Fourier Transform (DFT) for periodic signals [25]. This property states that a shift in time is equivalent to a linear phase shift in frequency. Since

the frequency content depends only on the shape of a signal, the frequency content remains unchanged in a time shift. Only the phase spectrum will be altered.

Discrete Fourier Transform of Non-Shifted Periodic Signal

$x[n] \rightarrow X(e^{jw})$

Power Spectral Density of Non-Shifted Periodic Signal

$\left|X(e^{jw})\right|^2 = X(e^{jw}) * X(e^{-jw})$

Discrete Fourier Transform of Temporally Shifted Periodic Signal

$x[n-m] \rightarrow e^{-jwm} X(e^{jw})$, where m is the shift in time

Power Spectral Density of Temporally Shifted Periodic Signal

$\left|X(e^{jw})\right|^2 = e^{-jwm} X(e^{jw}) * e^{jwm} X(e^{-jw}) = X(e^{jw}) * X(e^{-jw})$

As shown in the above formula, the PSD of a temporally shifted periodic signal is the same as the PSD of a non-shifted periodic signal. The power spectral density (PSD) [26] is a measure of how the power in a signal changes as a function of frequency. The power in this context is the square of Fast Fourier Transform's (FFT) magnitude. The unit for PSD of an EM signal measured in this thesis is $\mu V^2/MHz$, whereas the unit for PSD of a power signal is $\mu A^2/MHz$.

Although the EM or power data captured in this thesis are not periodic signals, it is still observed that the frequency contents of these discrete aperiodic signals are less vulnerable to the effects of time shifts. In fact, the power spectral density (PSD) of a temporally shifted EM trace is approximately the same as that of a non-shifted one. By analyzing the EM traces in the frequency domain, the effect of temporal misalignment in traces can be reduced.

A MATLAB simulation program is used to demonstrate that the frequency content of a discrete aperiodic signal remains approximately the same with time shifts. This MATLAB simulation program takes a set of real EM data from the ARM evaluation board as the input. All traces in this set of EM data are perfectly aligned. First, the

average of the power spectral density of this perfectly aligned set of data is computed. Then, the simulation program inserts time shifts to the same set of EM data in a random fashion. The average of the power spectral density of the temporally misaligned set of data is also computed. At the end, the power spectral density of the perfectly aligned and the misaligned EM traces are compared with each other as shown in Figure 11 below.

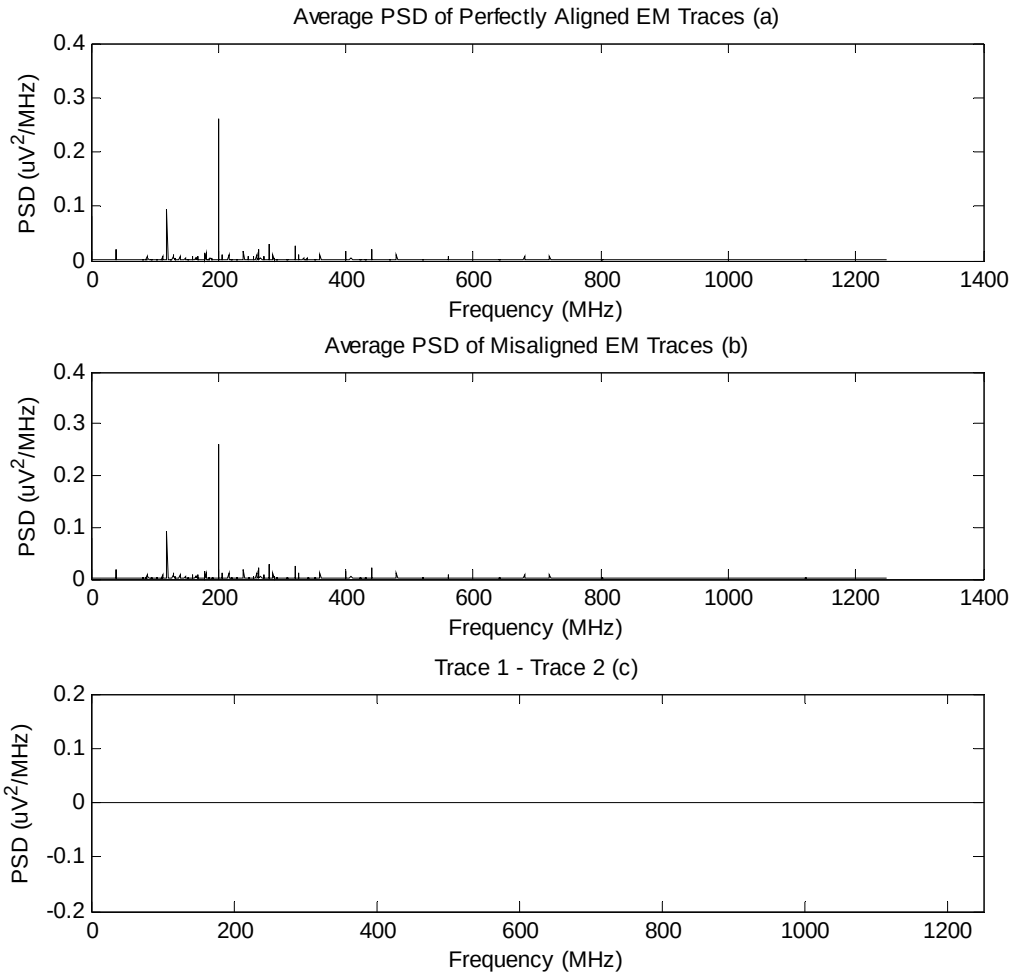

Figure 11: PSD of Misaligned & Aligned EM Traces (a&b), Trace 1 Minus Trace 2 (c)

In Figure 11, it is illustrated that the power spectral density for both perfectly aligned and misaligned EM signals are approximately the same. Figure 11a shows the average PSD of perfectly aligned EM Traces, whereas Figure 11b shows the average PSD

of misaligned EM Traces. Figure 11c is a plot of the difference of Figure 11a and Figure 11b. As indicated in the plot, the magnitude of the difference of these power spectral densities is so small implying that their difference is negligible. Therefore, it is shown that the time shifts effect can be minimized in the frequency domain.

According to Kocher's hypothesis of DPA [1], there is a significant difference in the differential time signal of traces in subset 0 and subset 1 if the key guess is correct. In theory, spikes in the differential signal in time domain should also appear in frequency domain, since any changes in the time domain signals would induce changes in the frequency domain signals. As a result, when computing the differential signal in the frequency domain, a significant difference between the PSD of traces in subset 0 and subset 1 is still present. In other words, subset 0 and subset 1 should have different power spectral density distributions. Thus, the DFA algorithm can decide whether the groupings, or the guess for the key, are correct by distinguishing these PSD distributions. If the key guess is incorrect, the PSD distributions of subset 0 and subset 1 are identically distributed. Hence, the differential PSD signal in the frequency domain is flat. If the key guess is correct, there is a significant difference in the PSD distributions of subset 0 and subset 1.

3.4.4 Runtime Analysis

Regarding the runtime analysis of DFA, the preprocessing, the PSD calculation of each trace, in DFA runs in time $\theta(nm\log m)$. Recall from Section 3.3 that n is the total number of traces and m is the number of sample points in each trace. After this preprocessing, each of the 256 groupings can be tested using DFA in time $\theta(nm)$. The total runtime of DFA attacking a 8-bit S-Box is therefore $\theta(nm\log m + 256nm)$. As for the runtime of normal DEMA or DPA attack, there is no need of preprocessing. The total runtime is $\theta(256nm)$. Although DFA has a slightly higher computational overhead, it is efficient in extracting secret key under severe trace misalignment experimental conditions.

3.5 Other Previously Researched Side Channel Attacks

To compare the effectiveness of DFA, this thesis compares the newly proposed technique to 3 previously researched attacks experimentally: the differential EM analysis (DEMA), the differential Spectrogram analysis (DSA), and Waddle's second order attack (FFT 2DPA). Before implementing these attacks in experiments, this section briefly introduces the methodology for each attack.

3.5.1 Differential Time Analysis (DPA & DEMA)

Since there are no conclusive results of DEMA on PDA's featuring AES in previous research, the attack is performed and results are presented in this thesis. The methodology of DPA or DEMA is described below.

i = trace number, $i \in \{0, \ldots, n-1\}$
b = set number, $b \in \{0,1\}$
T_i^b = EM signal of set b and trace i
STD_R = standard deviation of means (threshold signal)
K = key of AES encryption
maxPeak = vector of maximum spike outside 2 * STD_R for all 256 keys

DPA or *DEMA(T)*
1 : for each K, $K \in \{0, \cdots, 255\}$
2 : $\{T^0, T^1\} \leftarrow partitionTraces(T)$
3 : $STD\_R \leftarrow STD\_DOM(T^0, T^1)$
4 : $s \leftarrow \overline{T^0} - \overline{T^1}$
5 : maxPeak$(K) \leftarrow \max(abs(s) - 2 * STD_R)$
6 : return maxPeak

Also note that this thesis provides an improvement on the differential time analysis. In the past, the correct key is retrieved if there is a peak in the differential time signal. However, peaks caused by noise can also be observed in a wrong key guess. As indicated in the above function, the differential time signal is compared to the two times of standard deviation threshold signal. The standard deviation threshold signal is

computed using the raw time data instead of PSD. This is a better comparison since this approach minimizes the impact of false peaks as discussed in Section 3.3.

3.5.2 Differential Spectrogram Analysis (DSA)

Spectrogram is a type of time-dependant frequency analysis. It consists of both time and frequency information [15]. The only difference between DFA and DSA is instead of computing the power spectral density for each EM trace in the pre-processing stage, the spectrogram of each traces is calculated. The methodology of calculating spectrogram is described below.

i = trace number, $i \in \{0,\ldots,n-1\}$
b = set number, $b \in \{0,1\}$
T_i^b = EM signal of set b and trace i
V_i^b = spectrogram of set b and trace i
s = frame number in spectrogram, $s \in \{0,\ldots,p-1\}$
w = window size
f = frequency, $f \in \{0,\ldots,\frac{wp}{2}-1\} = \{0,\ldots,\frac{m}{2}-1\}$

SPECGRAM*(T)*
1 : for each $b, b \in \{0,1\}$ and for each $i, i \in \{0,\ldots,n-1\}$:
2 : for each $s, s \in \{0,\ldots,p\}$:
3 : $F \leftarrow \text{FFT}(T_i^b(s*w:(s+1)*w-1))$
4 : $V_i^b(s*\frac{w}{2}:(s+1)*\frac{w}{2}-1) \leftarrow F \bullet \text{HAMMING}(w)$
5 : return V

Then the set of spectrogram is partitioned into 2 groups based upon a key guess and the corresponding plaintext similar to the procedures in DFA. The value of the least significant bit (LSB) of the 8-bit S-Box output is computed in the first round of AES. The mean of each group of spectrogram is calculated. At the end, the differential spectrogram signal is computed and compared with the standard deviation threshold signal (STD_R) as described in Section 3.3. The standard deviation threshold signal (STD_R) is calculated in a same way as that for DFA, spectrogram is used instead of PSD.

3.5.3 Waddle's Second Order Differential Attack (FFT-2DPA)

Waddle's FFT2DPA [13] attack is a second-order differential analysis that is proposed to defect masking countermeasure. Autocorrelation, or correlation of a trace with itself, is used to overcome masking. The only difference between DFA and FFT-2DPA is instead of computing the power spectral density for each EM trace in the pre-processing stage, the autocorrelation of each traces is calculated. The methodology of calculating autocorrelation is described below.

$i = \text{trace number}, i \in \{0,\ldots,n-1\}$
$b = \text{set number}, b \in \{0,1\}$
$T_i^b = \text{EM signal of set } b \text{ and trace } i$

Autocorrelation*(T)*
1 : for each $b, b \in \{0,1\}$ and for each $i, i \in \{0,\ldots,n-1\}$:
2 : $F_i^b \leftarrow FFT(T_i^b)$
3 : $P_i^b \leftarrow \left|F_i^b\right|^2$
4 : $A_i^b \leftarrow INVFFT(P_i^b)$
5 : return A

It is also important to note that the analysis presented by Waddle *et al.* is transformed back in the time domain to perform differential analysis, whereas the DFA is performed in the frequency domain. Waddle's attack aims to defeat masking countermeasures, whereas DFA aims to overcome trace misalignment in first order differential analysis. In addition, DFA does not require computing the inverse FFT to transform the signal back to the time domain. Therefore, it requires less computational time.

4 Experiments

This chapter describes the side channel attack experiments on an ARM Integrator/C7TDMI core module and a personal digital assistant (PDA) whose identity is not revealed to protect the vendor. The purpose of this chapter is to introduce the equipment for measuring EM emanations and power consumption. Sections 4.1 and 4.2 present the experimental setup and results on the ARM Integrator/C7TDMI core module. Sections 4.3 and 4.4 present the experiment setup and results on the PDA.

4.1 Experimental Setup for ARM Integrator/C7TDMI core module

Figure 12 below shows the experimental setup for measuring EM emanation and power consumption from the ARM Integrator/C7TDMI core module. A digital oscilloscope, an EM probe connected to a pre-amplifier, a Multi-ICE debugger, a personal computer, and a inductive probe are used to acquire both EM and power traces from the ARM Tntegrator/C7TDMI core module. This section describes each of these instrument setups in details.

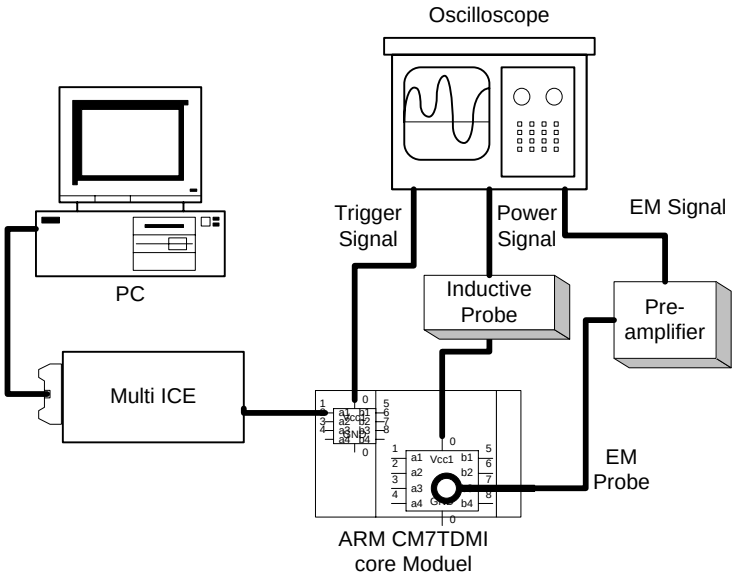

Figure 12: Power and EM Measurement Setup on ARM Integrator/C7TDMI.

4.1.1 ARM Integrator/CM7TDMI core module

The ARM Integrator/CM7TDMI core module is ideally suited for designs that require low power, small size and high performance. Hence, it is chosen as the experimental device in this thesis because it is widely used in embedded devices such as pagers, wireless handsets, and personal digital assistants (PDA).

This ARM evaluation board is used as a standalone development system connected to a Multi-ICE debugger. The Multi-ICE debugger is used to download image files, the byte code produced by the ARM compiler, to the core module. It is also used to debug the assembly code. There are seven main components in the ARM Integrator/CM7TDMI core module: two sets of diagnostic connectors and five test ports. The seven components consist of the ARM7TDMI microprocessor core [16], a core module FPGA, a volatile memory, a synchronous static RAM (SSRAM) controller, a clock generator, system bus connectors, and Multi-ICE connectors. Figure 13 illustrates the system architecture of the core module.

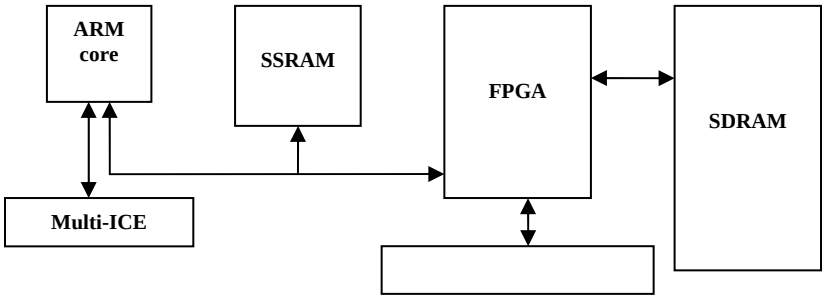

Figure 13: System Architecture of the ARM Integrator/CM7TDMI Core Module.

The ARM7TDMI microprocessor core is a separate chip on the evaluation board and is a 32-bit embedded Reduced Instruction Set Computer (RISC) processor. In addition, this processor core has both 32-bit unidirectional and bidirectional data bus, a 32-bit address bus going out to the memory and a three-stage pipeline [16]. The clock of the processor core is set to 40MHz for all experimental results collected in section 4.2 and section 4.4. In fact, EM emanations and power consumption both arise as a

consequence of current flows within the control, I/O, data processing or other parts of the ARM module. Of these numerous leakages, those induced by data processing operations carry the most compromising information. Therefore, the EM emanation from and the power consumption of the core processor are of particular interest in this thesis. For more information about the ARM evaluation board, see [16].

4.1.2 Trigger Setup

To measure EM and power traces, a trigger signal is needed to notify the oscilloscope when to start recording a trace. In the experiments on the ARM core module, the trigger signal is sent in the form of a software interrupt from the core module to the oscilloscope. To send an interrupt, the interrupt controller is enabled by setting the IRQ enable set register (CM_IRQ_ENSET) to '1'. The output pin of software interrupt is the nIRQ signal. Then, the oscilloscope starts recording a frame when the nIRQ signal goes from high to low. A program is written in assembly language to generate the trigger signal to the oscilloscope. The software interrupt clear register (CM_SOFT_INTCLR) is first set to '1' to put nIRQ to high, 40 "NOP" instructions are followed. Then, the software interrupt set register (CM_SOFT_INTSET) is then to '0' to put nIRQ to low, another 40 "NOP" instructions are executed to end the trigger. For more information on interrupts, refer to the ARM user's guide [17].

4.1.3 Digital Phosphor Oscilloscope

All the EM and power traces obtained in this thesis are captured with a Tektronix TDS 7254 digital oscilloscope. This section describes the features of the oscilloscope used in the experiments.

The oscilloscope can record up to 4 input signals simultaneously. For this particular experiment, Channel 1 is the input trigger signal from the ARM evaluation board. As mentioned in the trigger setup section above, the trigger signal nIRQ goes from high to low. Therefore, the trigger mode of the oscilloscope is set to the negative edge mode. The coupling mode is set to noise reject to minimize the noise in the trigger signal. Channel 2 is connected to an EM probe to measure the EM emanation from the ARM

core processor. Channel 3 is connected to an inductive probe to record the power consumption of the ARM core processor.

For all traces measured for the ARM experiments, the HiRes acquisition mode is used. In HiRes mode, the instrument creates a record point by averaging all samples taken during an acquisition interval. It results in a higher resolution, less noise, and lower bandwidth waveform. Of all the modes available, the HiRes mode gives the best quality.

To eliminate noise, averaging of at least several hundred of frames is required. The digital oscilloscope can acquire multiple frames in one recording in a special acquisition mode called FastFrame. In the FastFrame mode, multiple frames can be captured and each can be viewed and analyzed individually. The number of data points in each trace is specified by the record length and the number of traces is specified by the frame count. Each frame is captured when a valid trigger occurs.

The user can adjust the time duration of each frame and the sampling rate using the Scale and Resolution button of the scope. However, there is a limitation on the scope memory (record length) with a maximum of 32 million sample points. Therefore, users have to make tradeoffs between resolution and time duration accordingly. The following equations show the inter-dependency of scope memory (record length), time duration, and resolution.

Time Duration (s) = Sample Interval (s/sample) * Record Length (samples)
Sample Interval (s/sample) = Resolution (s/sample) = 1/Sampling Rate (samples/s)

As one can see from the above equations, increasing the time duration decreases the sampling rate, whereas increasing the sampling rate decreases the time duration. Resolution is sacrificed by having a long frame. For instance, in a frame of 2 μs with 5000 sample points, the sampling period is (2 μs / 5000) = 0.4 ns. The sampling rate is thus (1 / 0.4 ns) = 2.5 Giga samples/s. And the frequency span is half of the sampling rate, (0.5 * 2.5 Giga samples/s) = 1.25 GHz.

For instance, an AES encryption algorithm is too long to fit into one frame. In most cases, the adversary is only interested in a small section of the Rijndael encryption algorithm, particularly the output of the S-Box in 1 round of AES. Therefore, it is preferable to pinpoint the attack point by zooming into the section of interest using the delay mode on the oscilloscope. The delay mode allows the oscilloscope to start displaying waveform by a user-specified period after the start of the trigger signal. For instance, if the specified delay period is 1 ms, the scope will display a waveform 1 ms after the start of the trigger signal. As a result, the attacker can pinpoint the exact section that contains the load instruction at the output of the S-Box. Hence, this allows the attacker to increase the frame resolution and capture only the desired section into a single frame. For other features of the oscilloscope, see [22].

For all the tests on the ARM evaluation board, 2976 frames are captured in each acquisition. The sampling rate is 2.5GS/s. The duration of each frame is 2 μs. There are 5000 sample points in each frame. Table 1 below summarizes all the oscilloscope setup for the experiments on the ARM evaluation board.

Table 1: Summary of Oscilloscope Setup for Experiments on ARM

Acquisition mode	HiRes
	FastFrame
Trigger mode	Negative edge
Trigger coupling	Noise reject
Frame count	2976
Record length	5000
Delay	5.6 μs
Duration of 1 frame	2 μs
Frequency span	1.25 GHz
Sampling rate	2.5 Giga samples/s
Channel 1	Trigger signal
Channel 2	EM signal
Channel 3	Current signal

4.1.4 EM Probe

A near field EM probe by Electro-Metrics Inc. (Model EM-6992) is used to measure the EM emanation from the ARM processor core. The probe is connected to the scope using a 50-ohm coaxial cable. A preamplifier is inserted between the EM probe and the oscilloscope to improve the overall measurement sensitivity. The typical gain of the preamplifier is approximately 22 dB. The probe can test radiated emissions over a broad range of frequencies from below 100 kHz to 1 GHz. See [18] for more details about the pre-amplifier.

Probe choice is determined by the type of signal under observation, signal strength, and the physical size of the area to be investigated. It is important to note that the EM field can be decomposed into 2 primary components: an electric field and a magnetic field. The electric field fails for low frequency but carries different information than the magnetic one [19]. By trial and error, it is observed that the magnetic probe with a shape of a 1-cm loop gives the best EM signal quality. This probe has low sensitivity which helps in isolating an emission source more precisely. The loop is wound within a balanced Faraday shield that reduces its response to electric fields to a negligible factor. Therefore, it shows that the EM radiation from the ARM processor core has a stronger magnetic field component. The primary pickup direction is broadside to the loop, with sharp notches in the pickup pattern in the plane of the loop. Even small changes in the distance from the probe to the item under test can yield large variations in amplitude. This magnetic probe is put directly on top of the ARM processor core to obtain the best EM signal quality.

4.1.5 Inductive Probe

The power consumption of the ARM processor core is measured in a form of current in this thesis. The amount of voltage drawn on the ARM processor core is 3.3V, the current consumption can be easily converted to power consumption by multiplying the current with the supply voltage. On the Integrator/CM7TDMI core module, there are two test points (TP4 and TP5) located on the two ends of a zero-ohm resistor (R16). This zero-ohm resistor is a wire indeed. According to [17], these two test points can measure the

current drawn by the ARM7TDMI processor core. The zero-ohm resistor is the power line that connects the processor core to its power supply. Therefore, the core will draw different amount of power depending on the instructions executed. To measure this current, an inductive probe, Tektronix TCP202 DC/AC Inductive Current Probe [23], is connected to TP4 and TP5 and the zero-ohm resister is soldered off. The inductive probe is connected to the power line, and thus, the current change in the processor core will induce a change in the inductor. The inductive probe is connected to a digital phosphor oscilloscope where the change in current is recorded.

4.1.6 Experimental Methodology

Step 1: Loading the AES Encryption Program to the ARM Evaluation Board

The symmetric key algorithm undergoing the side channel attack is the AES encryption with a master key length of 128 bits. The program under test is written in assembly language using the optimized Rijndael implementation recommended in [20]. The test program is written such that the AES encryption is run in a loop for 2796 times with random plaintext inputs. The input plaintexts are kept in record for statistical analysis with MATLAB after the data capture step. The plaintexts are specially prepared in a way such that only the data at the output of the 1st S-Box would be different in the first round of Rijndael. The operands at the output of the rest of the 15 S-Boxes in the first round are kept constant in order to minimize the noise created by these S-Boxes. To create such effect, only the first 8 most significant bits of the plaintexts are random, the rest of all other bits are fixed.

The AES encryption assembly program is then converted into an image file using the ARM compiler. Next, turn on the ARM evaluation board. Launch the Multi-Ice server. First, load the image file using the AXD Debugger to set the clock speed of the ARM processor core to 40 MHz. Then load the image file containing the AES encryption test program.

Step 2: Capturing EM or Power Traces

Before running the encryption algorithm, place the EM probe on top of the core processor. The methodology for capturing power traces is basically the same as that for capturing EM traces. The only difference is that the inductive probe is used instead of the EM probe. The inductive probe is already connected to the test points on the board. Hence, no additional setup is required for measuring power data. Next, setup the oscilloscope according to Table 1 in Section 4.1.3. Then, execute the AES encryption program. After capturing the waveforms on the oscilloscope, generate a data file containing the EM or power data of the ARM core processor. Such data file is then exported to MATLAB for statistical analysis.

Step 3: Statistical Analysis with MATLAB

After step 2, statistical analysis is done on the raw EM or power data with a MATLAB program. The analysis program is written according to the DFA attack methodology described in Section 3.3. The MATLAB program produces a correct key guess after running through all possible keys. See Appendix 2 for the analysis program in MATLAB.

4.2 Experimental Results for Attacks on ARM Evaluation Board

This section presents the experimental results from the ARM evaluation board. Both EM emanation and power consumption are measured from the ARM evaluation board. First of all, results of EM analysis are illustrated, and then followed by results of power analysis. The purpose of this section is to evaluate the effectiveness of the proposed Differential Frequency Analysis to extract the secret key of the Rijndael encryption algorithm. In addition, an attack on the desynchronization countermeasure for AES against DEMA using the new frequency-based attack is performed. To verify that the results presented are consistent, all the experiments are repeated for 3 times.

4.2.1 EM Analysis on AES

In this experiment, EM traces are captured from the ARM core processor. The AES 128-bit encryption algorithm without countermeasure is loaded to the ARM evaluation board. The correct key of the S-Box being attacked is 0xD2. In order to recover the master key of the encryption algorithm, the differential EM analysis (DEMA) is first performed. Next, the differential EM frequency analysis (DEMFA) is performed on the same set of data.

4.2.1.1 Differential Electromagnetic Analysis (DEMA)

Before investigating the effectiveness of Differential Frequency Analysis for EM analysis, it is necessary to verify if the set of EM traces measured from the ARM evaluation board leaks any information about the master key of the AES encryption algorithm. Therefore, the first analysis done on the EM traces is the previously researched DEMA attack.

In the DEMA attack, EM traces are partitioned into 2 groups using the partition function *D(P, b, K)* according to the corresponding plaintext and key guess. The value of the least significant bit (LSB) of the 8-bit S-Box output is computed in the first round of AES. Refer to Figure 4 for the illustration of trace partitioning. Figure 14 below shows the differential time signal for the correct partition with the key value equals to 0xD2. The differential signal is computed by subtracting the average of traces in group 0 from the average of traces in group 1. Figure 14 shows a large spike occurring at regions around 0.8 μs suggesting that the partition function *D(P, b, K)* is correlated to the data being processed. The spike at regions around 0.8 μs means that the output of the S-Box is computed in this particular time. On the other hand, the differential time signal in Figure 15 is merely composed of noise for an incorrect key guess. There is no significant spike observed in this differential signal.

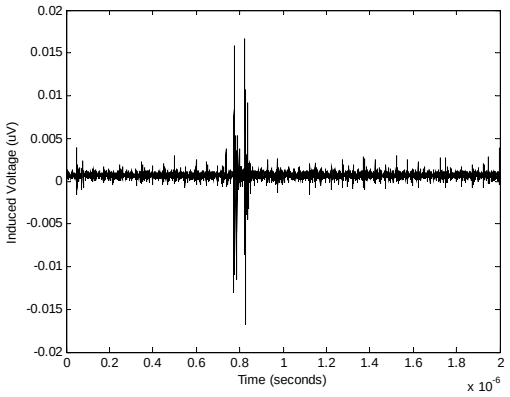

Figure 14: DEMA on ARM (correct key=0xD2)

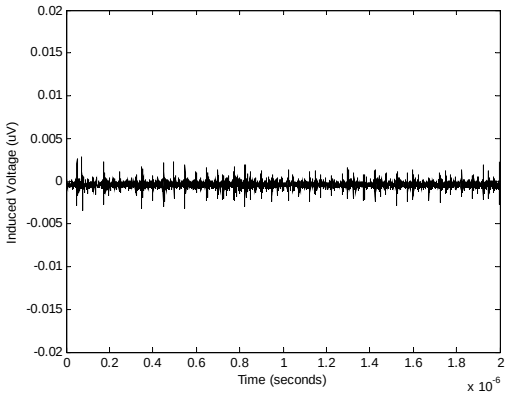

Figure 15: DEMA on ARM for (wrong key=0xA5)

Figure 16 below shows the all keys search of 256 possible key values. The analysis computes and compares the absolute value of the differential time signal and record the maximum peak outside 2*STD_R for each 256 possible key guesses. Refer to Section 3.5.1 for details of the DEMA methodology. As demonstrated in Figure 16, the key value 0xD2 has the biggest spike among all keys. Hence, the correct key is successfully extracted from the DEMA attack on the ARM evaluation board.

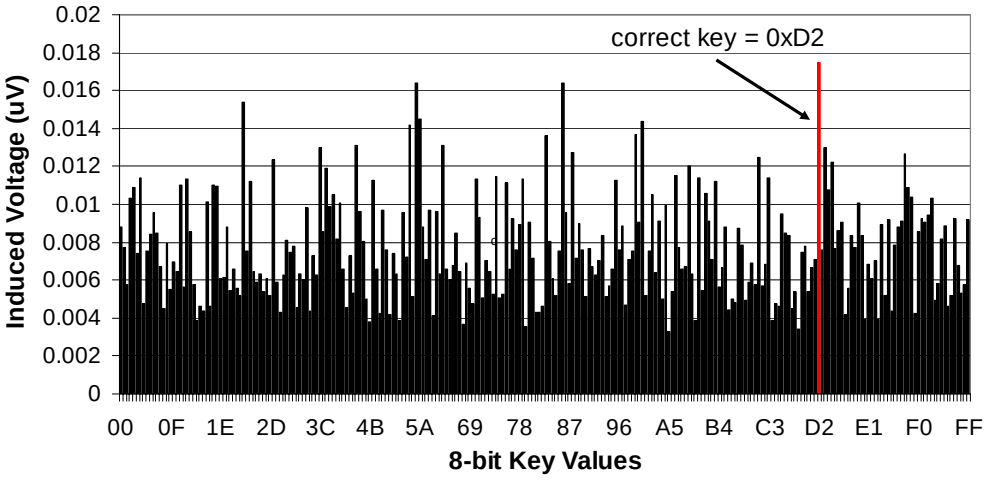

Figure 16: All Keys Search of DEMA on ARM

The above DEMA experimental results justify that the set of EM data measured leak compromising information about the master key of the AES encryption algorithm. Hence, Differential Frequency Analysis (DFA) is performed on the same set of EM data

in the next section to determine whether this new attack can extract the secret key as in DEMA.

4.2.1.2 Differential EM Frequency Analysis (DEMFA)

Knowing that DEMA is possible on the ARM evaluation board, the purpose of this test is to determine whether DEMFA could successfully extract the secret key of the Rijndael encryption algorithm. In this analysis, the EM traces are partitioned into 2 groups according to the corresponding plaintext and key guess similar to the DEMA attack. The value of the least significant bit (LSB) of the 8-bit S-Box output is computed in the first round of AES. Refer to Figure 4 for the illustration of trace partitioning. In this attack, one extra step is taken in this analysis, the raw time domain EM signal is transformed to the frequency domain. The power spectral density of each EM trace is computed in this pre-processing stage. Refer to Section 3.3 for details of the DEMFA methodology.

Figure 17 below shows the differential PSD signal for the correct partition with the key value equals to 0xD2. Note that this is a differential frequency signal. Unlike DEMA, the differential frequency signal is computed by subtracting the averaged power spectral density for traces in group 0 from the averaged power spectral density for traces in group 1. The y-axis now represents the PSD magnitude and the x-axis represents the frequency. The differential frequency signal in Figure 17 has notably higher amount of area outside of the ±2*STD_R region. Recall from Section 3.3 that the ±2*STD_R region serves as a benchmark to determine whether a spike in the differential frequency signal is significant. The differential frequency signal in Figure 18 has drastically less spikes outside the ±2*STD_R region for an incorrect key guess.

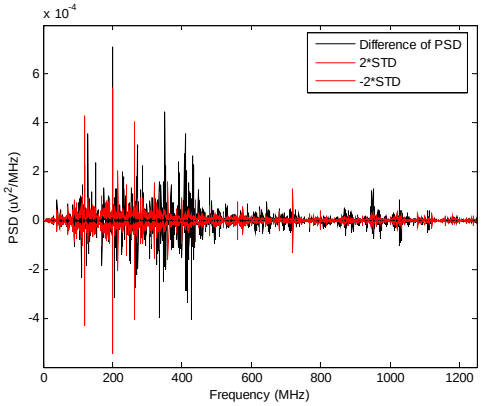

Figure 17: DEMFA on ARM (correct key=0xD2)

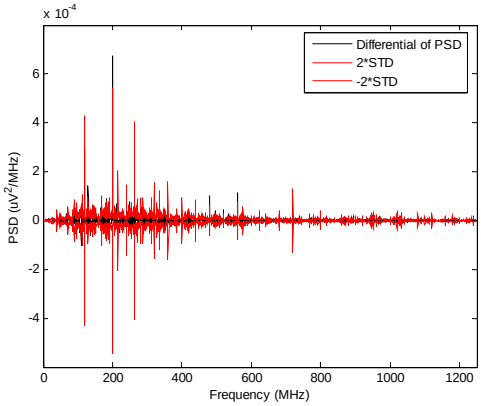

Figure 18: DEMFA on ARM (wrong key=0xA5)

Figure 19 shows the all keys search of 256 possible key values. The analysis computes and compares the total area of spikes that are beyond the ±2*STD_R region for all 256 possible key values of the AES S-Box being attacked. As demonstrated in Figure 19, the key value 0xD2 has the highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. Hence, the correct key is successfully recovered from the DEMFA.

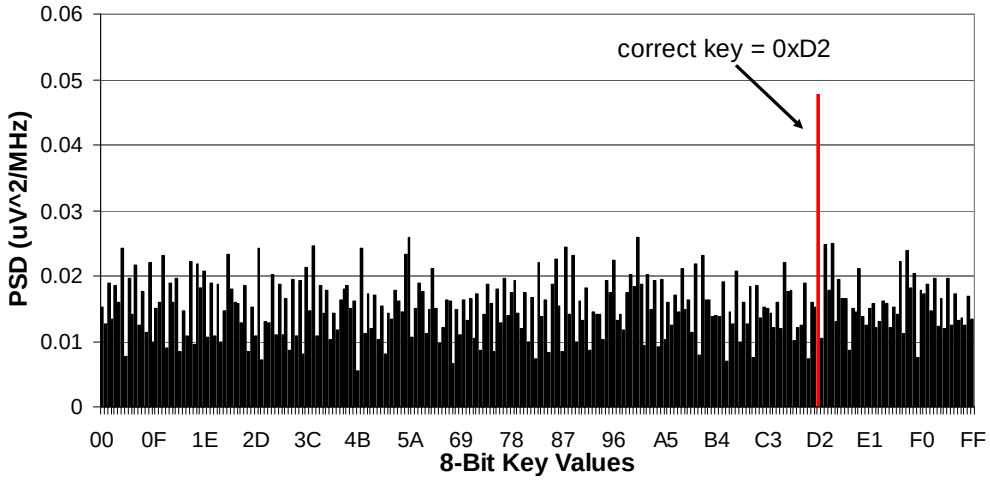

Figure 19: All Keys Search of DEMFA on ARM

The experimental results from above justify that the differential EM frequency analysis is at least as effective as the previously researched DEMA attack.

4.2.2 EM Analysis on a Single Load Instruction

To further investigate the behavior of the Differential Frequency Analysis (DFA), this section describes an experiment that measures EM radiation from the ARM evaluation board. Instead of running the Rijndael encryption algorithm, the test program executes a "Load" instruction of AES S-Box sandwiched between several "NOP" instructions. The purpose of this test is to verify that only the "Load" instruction is responsible of the difference in the differential PSD signal in the DEMFA attack.

In this test, one has to locate where the S-Box "Load" instruction occurs at first. Using the same techniques as DEMA, it is observed that the "Load" instruction occurs between sample points 2000 to 2300. Knowing exactly where the "Load" instruction is located in time, 2 verification tests has to be done. One test involves keeping the EM signals containing only "NOP" instructions but filtering out all signals of the S-Box "Load" instruction. The purpose of this attack is to determine whether the "NOP" instructions have any effects on the DEMFA results. Theoretically, these "NOP" instructions would not compromise any information of the secret key since they do not contain any information of the data manipulated in the algorithm. Figure 20 shows the all keys search results. The correct key cannot be determined since the S-Box "Load" instruction is not present in the EM traces.

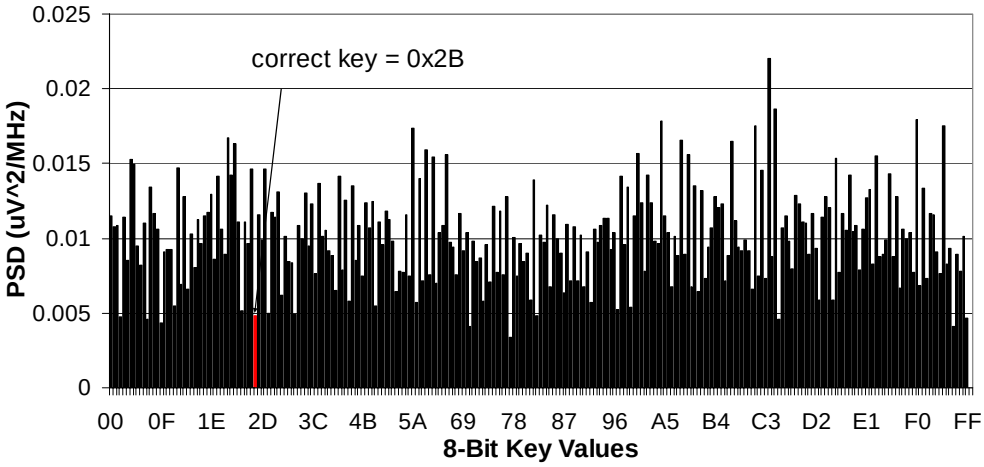

Figure 20: All Keys Search of DEMFA on ARM excluding the "Load" Instruction

The other test is the exact opposite of the previous one. It involves keeping the EM signals containing only the S-Box "Load" instruction and filtering out the rest of the "NOP" instructions. The DEMFA attack is performed on this set of signal with only the "Load" instruction EM data. Figure 21 shows the all keys search results by comparing the differential PSD signal for each key guess. It is clearly indicated that the attack is able to extract the correct master key from the EM data containing only the AES S-Box "Load" signals.

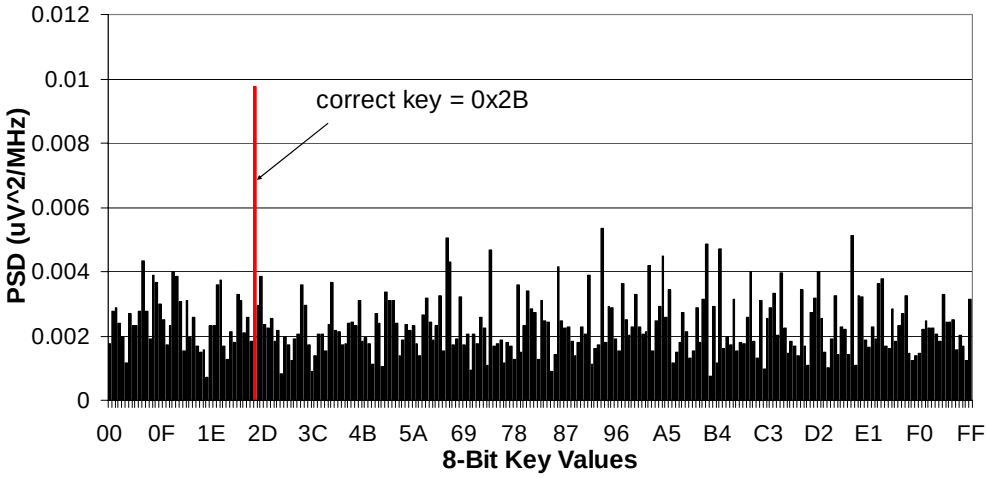

Figure 21: All Keys Search of DEMFA on ARM for the "Load" Instruction

These two tests, DEMFA on S-Box "Load" instruction and DEMFA on "NOP" instructions, demonstrate that only the S-Box "Load" is responsible for the significant difference in the averaged power spectral density distribution of group 0 and group 1 for a correct key guess. Therefore, it is confirmed experimentally that spikes appeared in the differential signal in time domain also appear in frequency domain, since any changes in the time domain signals would induce changes in the frequency domain signals. As a result, when computing the differential signal in the frequency domain, a significant difference between the PSD traces in subset 0 and subset 1 is still present.

4.2.3 EM Analysis on AES with Countermeasure

Having shown that the AES implementation is completely broken by both DEMA and the new DEMFA attack, this test perform the same EM attacks on the Rijndael encryption with a countermeasure implemented. Some countermeasures for DEMA consist in introducing desynchronization in the execution of the process so that the curves are not aligned anymore within a same acquisition set. For example, there exist various techniques such as fake cycle insertions, unstable clocking or random delays [21]. The purpose of this experiment is to investigate whether the desynchronization countermeasure can protect the symmetric key algorithm against DEMA and DEMFA attacks.

In this test, the original AES encryption algorithm is modified to randomly insert "NOP" instructions in order to create random delay in each EM frame. The aim of this countermeasure is to create the effect of temporal misalignment in EM traces.

4.2.3.1 Differential Electromagnetic Analysis (DEMA)

Figure 22 shows the all keys search of 256 possible key values of the DEMA attack. The analysis computes and compares the absolute value of the differential time signal and record the maximum peak outside 2*STD_R for all 256 possible key values of the S-Box. As demonstrated in Figure 22, the key value 0xD2 does not have the biggest spike among all keys. Hence, the correct key cannot be extracted from this attack. This test shows that the desynchronization countermeasure is effective against DEMA.

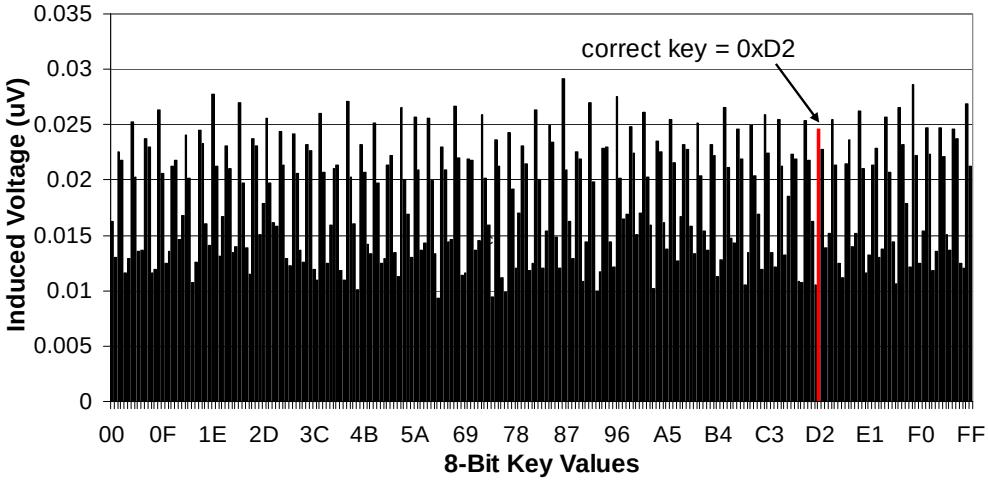

Figure 22: All Keys Search of DEMA on ARM for AES with Countermeasure

4.2.3.2 Differential EM Frequency Analysis (DEMFA)

Figure 23 shows the all keys search of 256 possible key values of the frequency attack. This attack computes and compares the total area of spikes that are beyond the ±2*STD_R region for all 256 possible key values of the AES S-Box being attacked. As illustrated in Figure 23, the key value 0xD2 has the highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. The correct key is successfully extracted. Hence, the proposed DEMFA attack is shown to be able to defect the desynchronization countermeasure that randomly inserts time shifts.

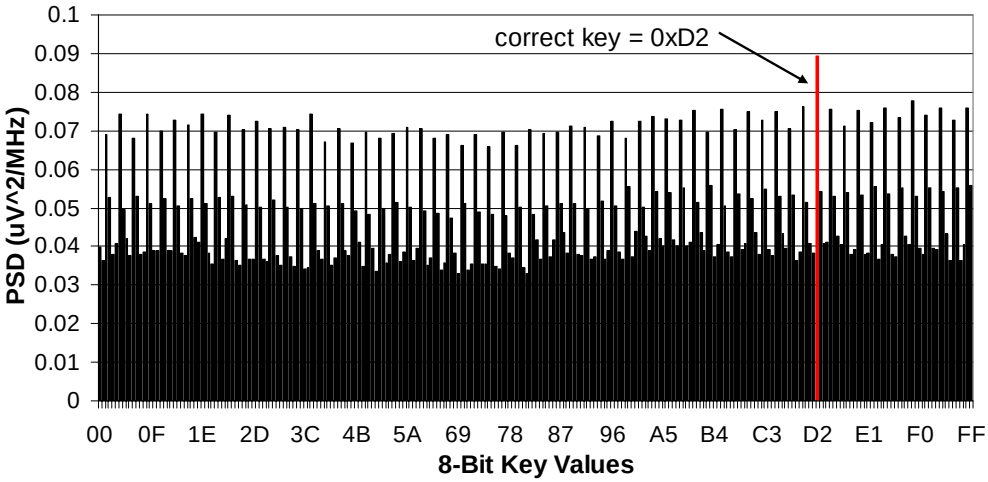

Figure 23: All Keys Search of DEMFA on ARM for AES with Countermeasure

4.2.4 Power Analysis on AES

This section presents the experimental results of power analysis on the ARM evaluation board. The purpose of the experiments is to verify that the results from the EM emanation agree with the power consumption of the ARM core processor.

Before investigating the effectiveness of Differential Frequency Analysis for power traces, it is once again necessary to verify if the set of power traces measured from the ARM evaluation board leaks any information about the master key of the AES encryption algorithm. Therefore, the first analysis done on the power traces is the previously researched DPA attack.

In this experiment, power traces are captured from the ARM core processor. The AES 128-bit encryption algorithm without countermeasure is loaded to the ARM evaluation board. The correct key of the S-Box being attacked is 0xD2. In order to recover the master key of the encryption algorithm, the differential power analysis (DPA) is first performed. Next, the differential power frequency analysis (DPFA) is performed on the same set of data.

4.2.4.1 Differential Power Analysis (DPA)

In this DPA attack, power traces are partitioned into 2 groups according to the corresponding plaintext and key guess. The value of the least significant bit (LSB) of the 8-bit S-Box output is computed in the first round of AES. Refer to Figure 4 for the illustration of trace partitioning. Figure 24 below shows the differential time signal for the correct partition with the key value equals to 0xD2. The differential signal is computed by subtracting the average of traces in group 0 from the average of traces in group 1. Figure 24 shows a large spike occurring at regions around 0.8 μs suggesting that the partition function *D(P, b, K)* is correlated to the data being processed. The spike at regions around 0.8 μs means that the output of the S-Box is computed in this particular time. On the other hand, the differential time signal in Figure 25 is merely composed of noise for an incorrect key guess. There is no significant spike observed in this differential time signal.

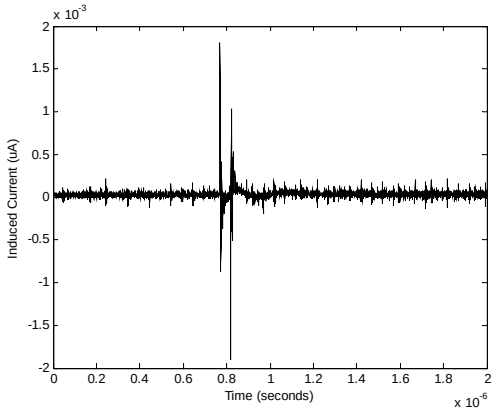

Figure 24: DPA on ARM (correct key=0xD2)

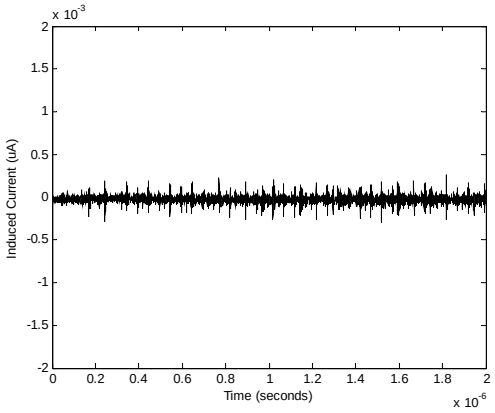

Figure 25: DPA on ARM (wrong key=0xA5)

The DPA results confirm that the attack point, the output of AES S-Box occurs at around time 0.8 µs. The results agree with the DEMA results presented earlier. Figure 26 shows the all keys search of 256 possible values. The analysis computes and compares the absolute value of the differential time signal and record the maximum peak outside 2*STD_R for each 256 possible key values of the S-Box. As demonstrated from Figure 26, the key value 0xD2 has the biggest spike among all keys. Hence, the correct key is successfully recovered using the DPA attack.

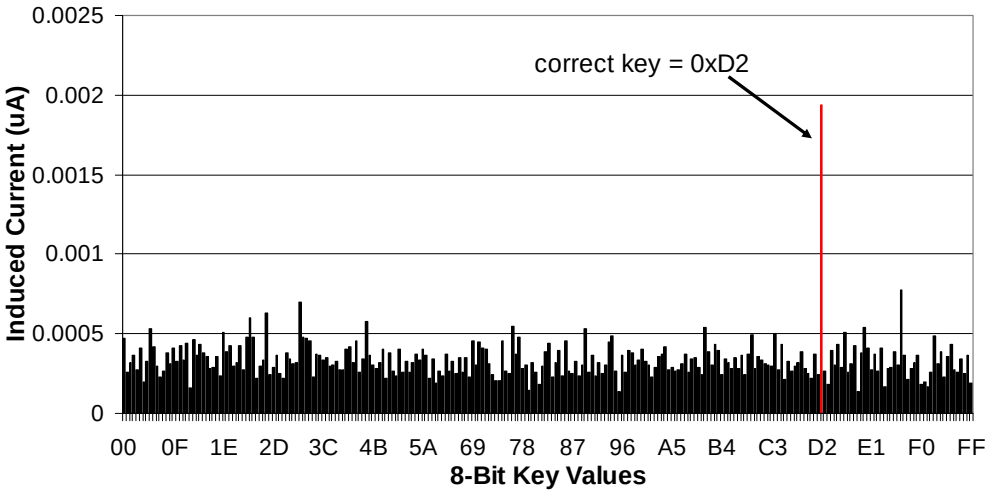

Figure 26: All Keys Search of DPA on ARM

The above DPA results justify that the set of power data measured leak compromising information about the master key of the AES encryption algorithm. Hence,

Differential Frequency Analysis is performed on the same set of power data in the next section determine whether this new attack can extract the secret key as in DPA.

4.2.4.2 Differential Power Frequency Analysis (DPFA)

Knowing that DPA is possible on the ARM evaluation board, the purpose of this test is to determine whether DFA could successful extract the master key of Rijndael encryption algorithm using the same set of power traces. In this analysis, the power traces are partitioned into 2 groups according to the corresponding plaintext and key guess similar to the DPA attack. The value of the least significant bit (LSB) of the 8-bit S-Box output is computed in the first round of AES. Refer to Figure 4 for the illustration of trace partitioning. In this attack, one extra step is taken in this analysis, the raw time domain power signal is transformed to the frequency domain. The power spectral density of each power trace is computed in this step. Refer to Section 3.3 for details of the DPFA methodology.

Figure 27 below shows the differential signal for the correct partition with the key value equals to 0xD2. Note that this is a differential frequency signal. Unlike DPA, the differential frequency signal is computed by subtracting the averaged power spectral density for traces in group 0 from the averaged power spectral density for traces in group 1. The differential frequency signal in Figure 27 has notably higher amount of area outside of the ±2*STD_R region. Recall from Section 3.3 that the ±2*STD_R region serves as a benchmark to determine whether a spike in the differential frequency signal is significant. The differential signal in Figure 28 has drastically less spikes outside the ±2*STD_R region for an incorrect key guess.

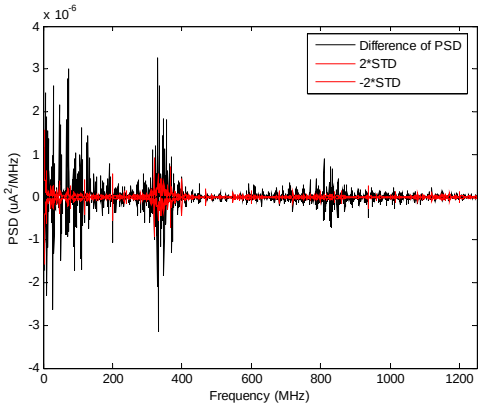

Figure 27: DPFA on ARM (correct key=0xD2)

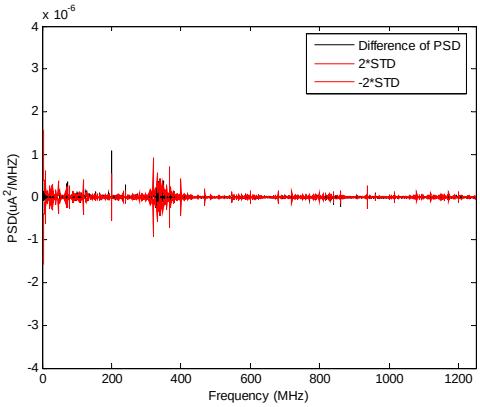

Figure 28: DPFA on ARM (wrong key=0xA5)

Figure 29 shows the all keys search of 256 possible values. The analysis computes and compares the total area of spikes that are beyond the ±2*STD_R region for all 256 possible key values of the Rijndael S-Box being attacked. As illustrated in Figure 29, the key value 0xD2 has the highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. Hence, the correct key is successfully extracted from the DPFA attack.

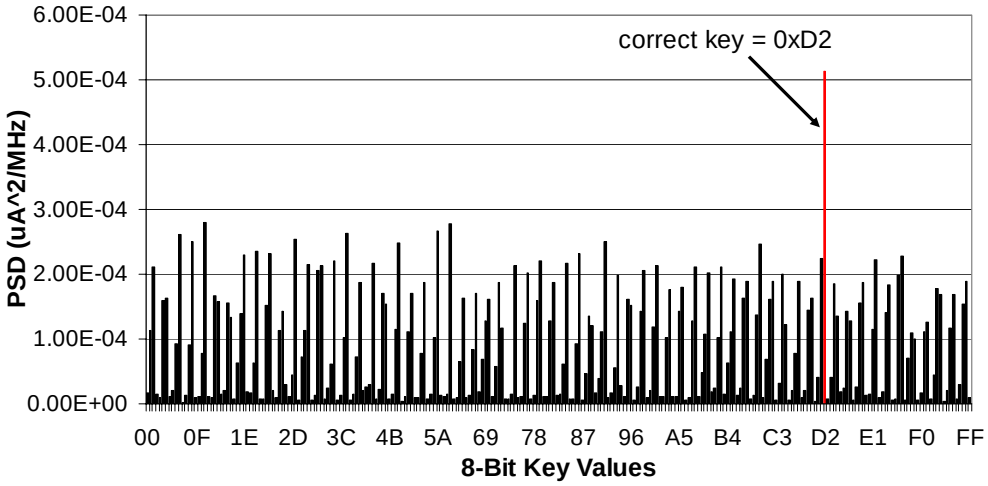

Figure 29: All Keys Search of DPFA on ARM

The experimental results from above justify that the DPFA is at least as effective as the previously researched DPA attack.

4.2.5 Power Analysis on AES with Countermeasure

Having shown that the AES implementation is completely broken by both DPA and the new DPFA attack, this test perform the same power attacks on the Rijndael encryption with a countermeasure implemented. Some countermeasures for DPA consist in inserting random delays. The purpose of this experiment is to investigate whether the desynchronization countermeasure can protect the symmetric key algorithm against DPA and DPFA attacks.

In this test, the original AES encryption algorithm is modified to randomly insert "NOP" instructions to create random delay in each power frame. The aim of this countermeasure is to create the effect of temporal misalignment in power traces.

4.2.5.1 Differential Power Analysis (DPA)

Figure 30 shows the all keys search of 256 possible key values of the DPA attack. The analysis computes and compares the absolute value of the differential time signal and record the maximum peak outside 2*STD_R for all 256 possible key values of the S-Box. As demonstrated in Figure 30, the key value 0xD2 does not have the biggest spike among all keys. Hence, the correct key cannot be extracted from this attack. The desynchronization countermeasure is effective against DPA.

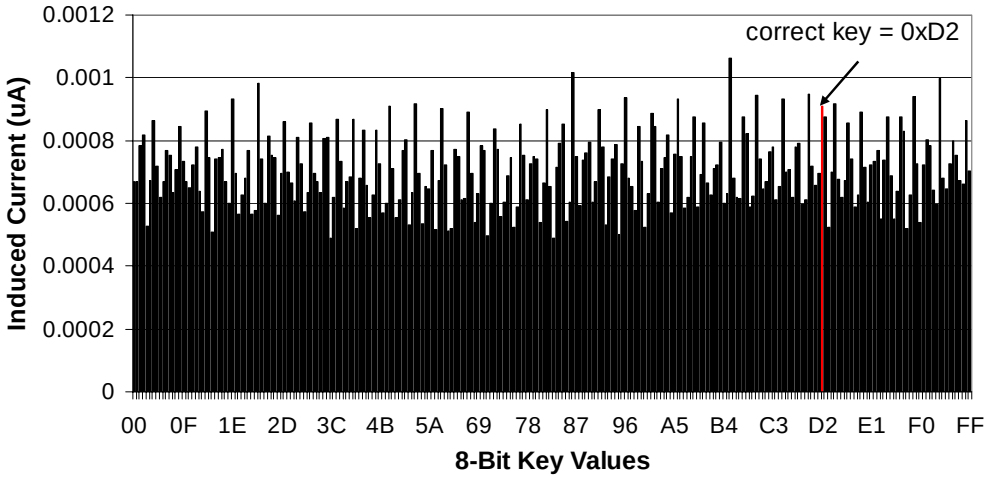

Figure 30: All Keys Search of DPA on ARM for AES with Countermeasure

4.2.5.2 Differential Power Frequency Analysis (DPFA)

Figure 31 shows the all keys search of 256 possible key values of the frequency attack. The analysis computes and compares the total area of spikes that are beyond the ±2*STD_R region for all 256 possible key values of the AES S-Box being attacked. As illustrated from Figure 31, the key value 0xD2 has the highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. The correct key is successfully extracted. Hence, the proposed DPFA attack is shown to be able to defect the desynchronization countermeasure that randomly inserts time shifts.

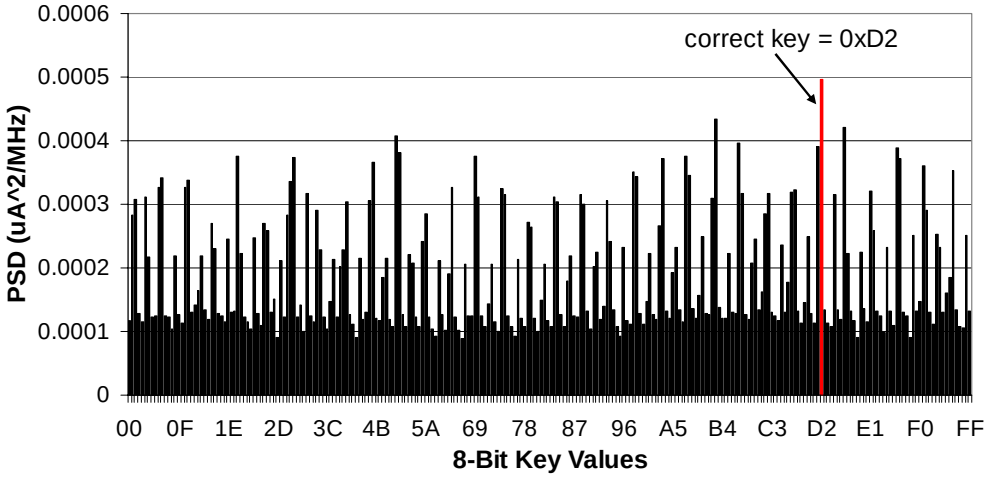

Figure 31: All Keys Search of DPFA on ARM for AES with Countermeasure

In summary, the experimental results in this section show that both the EM emanation and power consumption of the ARM evaluation board leak secret information about the symmetric key algorithm. The master key is also successfully extracted even when countermeasure is implemented using the new side channel attack, the Differential Frequency Analysis (DFA). It is confirmed experimentally that DFA is more effective than previously researched DEMA and DPA.

4.3 Experimental Setup for PDA

Figure 32 below shows the experimental setup for measuring EM emanation from a PDA. A digital oscilloscope and an EM probe connected to a pre-amplifier are used to acquire EM traces from the PDA. This section describes the instrumental setup in details.

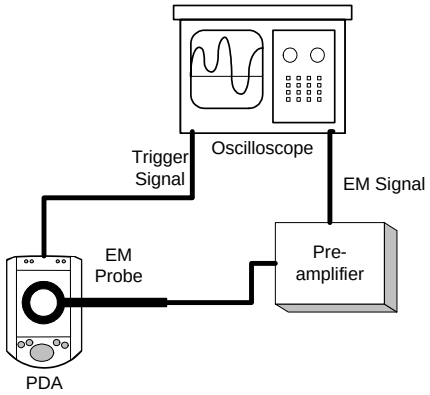

Figure 32: EM Measurement Setup on PDA

4.3.1 PDA

To protect the vendor identity, the PDA model is not revealed in this thesis. The shielding on the back of this wireless Java-based PDA is removed to expose the processor such that the EM probe can be placed directly on top. All applications for this PDA must be written in Java. The PDA has a much more complex architecture than the ARM Integrator/CM7TDMI core module. Its processor operates at a higher clock frequency. It also consists of other components such as LCD screen, radio antenna and receiver, non-volatile memory, etc. Only the EM side channel is available from this PDA.

4.3.2 Trigger Setup

To measure EM traces, a trigger signal is needed to notify the oscilloscope when to start recording a trace. In the experiments on the PDA, the trigger signal is generated by switching the light emitting diode (LED) of the PDA ON and OFF. The LED is turned ON and OFF using the Java API supported by the PDA vendor. To start the trigger signal, the LED is first turned ON and then turned OFF. The voltage difference between the ON and OFF state of the LED is used to trigger the oscilloscope.

4.3.3 Digital Phosphor Oscilloscope

The setup of the oscilloscope for PDA experiments is basically similar to the setup described in Section 4.3.3. Note that for all the traces measured from the PDA, the peak detect mode is used instead. Due to the memory restriction of the scope as discussed in Section 4.1.3, the frequency span of the EM data measured from the PDA is low comparing to that from the ARM evaluation board. Of all the modes available, the peak detect mode gives the best quality for EM signals captured at low frequency. In this acquisition mode, the scope alternates between saving the lowest sample in one acquisition interval and the highest sample in the next acquisition interval. Also note that the trigger signal goes from low to high, a positive edge trigger mode is used on the oscilloscope.

For all the tests on the PDA, 1030 frames are captured in each acquisition. The sampling rate is 25MS/s. The duration of one frame is 1 ms, There are 25000 sample points in each frame. Table 2 below summarizes the oscilloscope setup for experiments on PDA.

Table 2: Summary of Oscilloscope Setup for Experiments on PDA

Acquisition mode	Peak detect
	FastFrame
Trigger mode	Positive edge
Trigger coupling	Noise reject
Frame count	1303
Record length	25000
Delay	0 ms
Duration of 1 frame	1 ms
Frequency span	12.5 MHz
Sampling rate	25 Mega samples/s
Channel 1	Trigger signal
Channel 2	EM signal

4.3.4 EM Probe

The EM probe used is the same as the one discussed in section 4.1.4. The EM probe is placed directly on top of the processor of the PDA.

4.3.5 Experimental Methodology

Step 1: Loading the AES Encryption Program to the PDA

The symmetric key algorithm undergoing the side channel attack is the AES encryption with a master key length of 128 bits. The program under test is written in Java using the optimized Rijndael implementation recommended in [20]. The test program is written such that the AES encryption is run in a loop for 1303 times with random plaintext inputs. See Appendix 3 for the Java code of the AES implementation. The input plaintexts are kept in record for statistical analysis with MATLAB after the data capture step. The plaintexts are specially prepared in a way such that only the data at the output of the 1st S-Box would be different in the first round of Rijndael. The operands at the output of the rest of the 15 S-Boxes in the first round are kept constant in order to minimize the noise created by these S-Boxes. To create such effect, only the first 8 most significant bits of the plaintexts are random, the rest of all other bits are fixed. To capture EM signals from a PDA, load the AES encryption program to the PDA.

Step 2: Capturing EM Traces

Next, connect LED as the trigger signal to the oscilloscope. Before running the encryption algorithm, place the EM probe on top of the processor of PDA. It is observed that the location of the EM probe will affect the quality of the signal captured. By trial and error, it is observed that by placing the probe directly on top of the processor chip at a zero-degree angle gives the best EM signal quality. Next, setup the oscilloscope according to Table 2 in Section 4.3.3 and execute the AES encryption program. After capturing the waveform on the oscilloscope, generate a data file containing the EM emanation of the PDA processor. Such data file is then exported to MATLAB for statistical analysis.

Step 3: Statistical Analysis with MATLAB

After step 2, statistical analysis is done on the raw EM data with a MATLAB program. The analysis program is written according to the DFA attack methodology described in Section 3.3. The MATLAB program produces a correct key guess after running through all possible keys. See Appendix 2 for the analysis program in MATLAB.

4.4 Experimental Results for Attacks on PDA

This section presents the experimental results from the PDA. Since it is not possible to measure power consumption of the PDA, only EM emanations are measured from the PDA. This section investigates the threat of EM analysis on PDA's. The purpose of this section is to evaluate the effectiveness of the proposed Differential Frequency Analysis to extract the secret key of the Rijndael encryption algorithm when uncorrelated temporal misalignment of traces is severe. In order to make sure that the results are consistent, all the experiments are repeated for 3 times.

4.4.1 EM Analysis on AES

In this experiment, EM traces are captured from the PDA running the Rijndael encryption without any countermeasures implemented. First of all, the simple EM analysis (SEMA) is used to demonstrate that the sequence of instructions executed on the PDA can be revealed from a single EM trace. In order to extract the master key of the encryption algorithm, the differential EM analysis (DEMA) is first performed. Next, the differential EM frequency analysis (DEMFA) is performed on the same set of data.

4.4.1.1 Simple Electromagnetic Analysis (SEMA)

According to Kocher, simple power analysis can yield information about a device's operation as well as key material. Similarly for simple electromagnetic analysis (SEMA), experimental results show that the sequence of instructions executed on the device under test can be also revealed. Therefore, it can be used to break cryptographic implementations in which the execution path depends on the data being processed. Figure 33 below shows the scope capture of an AES encryption with 192-bit key length of a single EM frame. Twelve rounds of AES transformations are clearly shown in the figure. In fact, an adversary can determine the key length from simply inspecting the number of

rounds being executed during an AES encryption. In other words, for AES with 256 bit key length, one would expect to see 14 rounds from the EM capture. Hence, SEMA can reveal the number of rounds in the AES encryption to determine the key length.

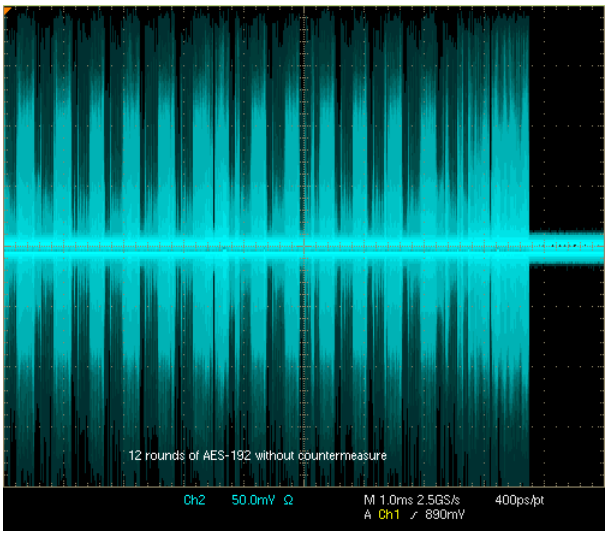

Figure 33: SEMA on PDA for AES 192-bit

4.4.1.2 Differential Electromagnetic Analysis (DEMA)

The AES 128-bit encryption algorithm without countermeasure is loaded to the PDA. The correct key of the S-Box is 0x5C. Before investigating the effectiveness of Differential Frequency Analysis for EM analysis, it is necessary to verify if the set of EM traces measured from the PDA leaks any information about the master key of the AES encryption algorithm. Therefore, the first analysis done on the EM traces is the previously researched DEMA attack.

In this DEMA attack, the EM traces are partitioned into 2 groups according to the corresponding plaintext and key guess. The value of the least significant bit (LSB) of the 8-bit S-Box output is computed in the first round of AES. Refer to Figure 4 for the illustration of trace partitioning. Figure 34 below shows the differential signal for the correct partition with the key value equals to 0x5C. Figure 34 below shows the differential signal for the wrong partition with the key value equals to 0xE6. The differential time signal is computed by subtracting the average of traces in group 0 from

the average of traces in group 1. As shown in Figure 34 and Figure 35, no significant spike is observed from the differential time signal for both key guesses. Misaligned traces cause large spurious peaks in a differential trace. When spikes are slightly out of alignment in time, they will cancel out rather than reinforced when averaging.

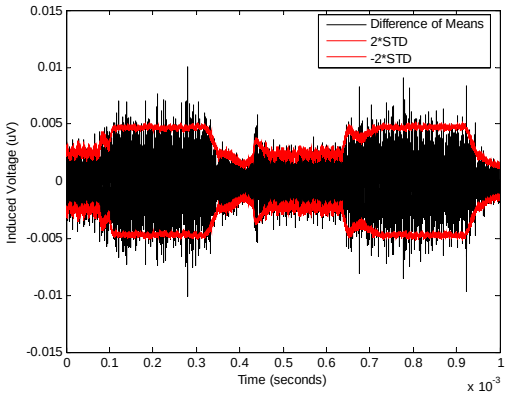

Figure 34: DEMA on PDA (correct key=0x5C)

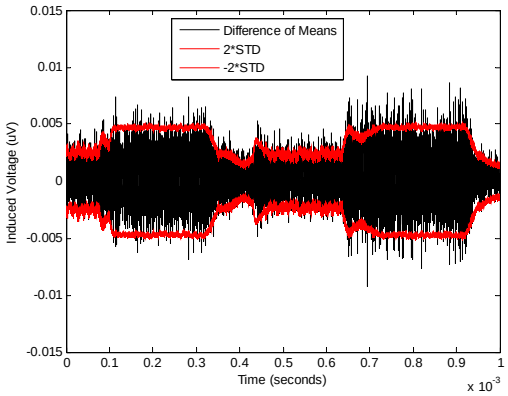

Figure 35: DEMA on PDA (wrong key=0xE6)

Figure 36 shows the all keys search for 256 possible key values. The analysis computes and compares the absolute value of the differential time signal and record the maximum peak outside 2*STD_R for all 256 possible key values of the S-Box. Refer to Section 3.5.1 for details of the DEMA methodology. As illustrated in Figure 36, the key value 0x5C does not have the biggest spike among all keys. Hence, the correct key is not recovered. DEMA is not effective when misalignment of traces is severe during experiments.

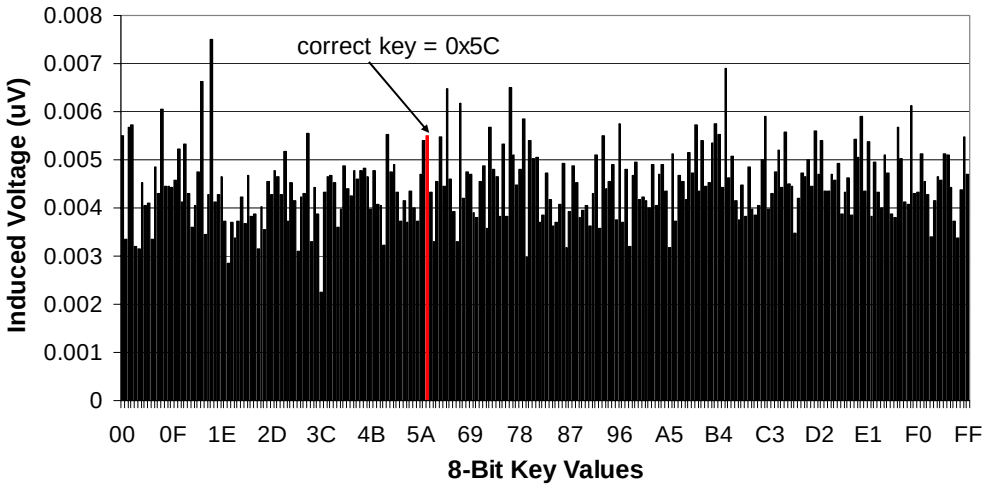

Figure 36: All Keys Search of DEMA on PDA

Since the previously researched DEMA is unsuccessful in extracting the correct key of the AES encryption, the next section investigates the effectiveness of analysis in frequency domain under severe experimental conditions.

4.4.1.3 Differential EM Frequency Analysis (DEMFA)

Knowing that DEMA is incapable of determining the Rijndael encryption key using the EM traces from the PDA, the purpose of this analysis is to determine whether the new DEMFA attack could successfully recover the master key. In this analysis, the EM traces are partitioned into 2 groups according to the corresponding plaintext and key guess similar to the DEMA attack. The value of the least significant bit (LSB) of the 8-bit S-Box output is computed in the first round of AES. Refer to Figure 4 for the illustration of trace partitioning. In this attack, one extra step is taken in this analysis, the raw time domain EM signal is transformed to the frequency domain. The power spectral density of each EM trace is computed in this step. Refer to Section 3.3 for details of the DEMFA methodology.

Figure 37 below shows the differential PSD signal for the correct partition with the key value equals to 0x5C. Note that this is a differential frequency signal. Unlike DEMA, the differential frequency signal is computed by subtracting the averaged power spectral density for traces in group 0 from the averaged power spectral density for traces

in group 1. The y-axis represents the PSD magnitude and the x-axis represents the frequency. The differential signal in Figure 37 has significantly higher amount of area outside the ±2*STD_R region. Please refer to section 3.2 for calculation of the STD_R region. The differential signal in Figure 38, on the other hand, has significantly less spikes outside the standard deviation region for an incorrect key guess of 0xE6.

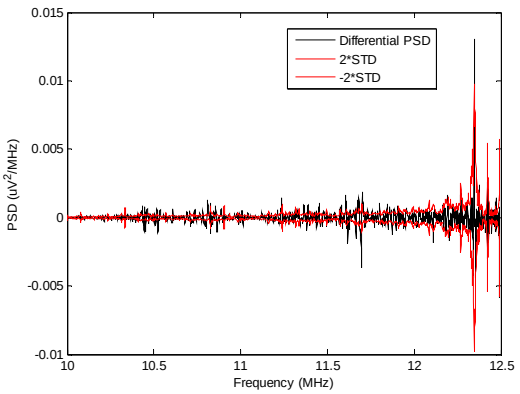

Figure 37: DEMFA on PDA (correct key=0x5C)

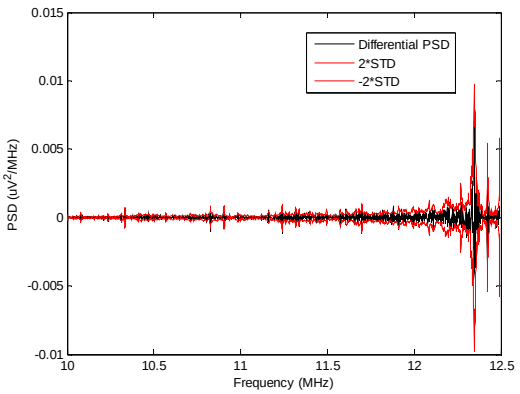

Figure 38: DEMFA on PDA (wrong key=0xE6)

Figure 39 shows the all keys search for 256 possible key values. The analysis computes and compares the total area of PSD spikes that are beyond the ±2*STD_R region for all 256 possible key values. The AES S-Box being attacked is used in the 8 most significant bit (MSB) of the 128-bit key. As shown in Figure 39, the key value 0x5C has highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. Hence, the correct key is successfully recovered from the DEMFA.

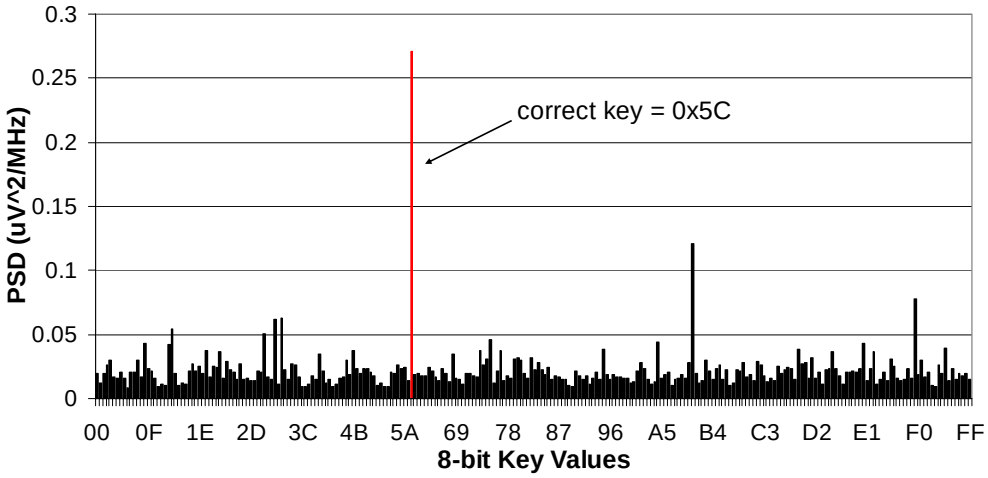

Figure 39: All Keys Search of DEMFA on PDA

The experimental results from above justify that the differential EM frequency analysis is more effective than the DEMA attack when the problem of misalignment of traces and noise is severe.

Figure 40 illustrates the results of attacking a different S-Box in the AES algorithm. The AES S-Box being attacked is used in the 8 least significant bit (LSB) of the 128-bit key. The correct key for this last S-Box is 0x3C. The correct key is successfully determined by the DEMFA. It is shown that the DEMFA attack can extract the complete 128-bit secret key by attacking all 16 S-Boxes one at a time.

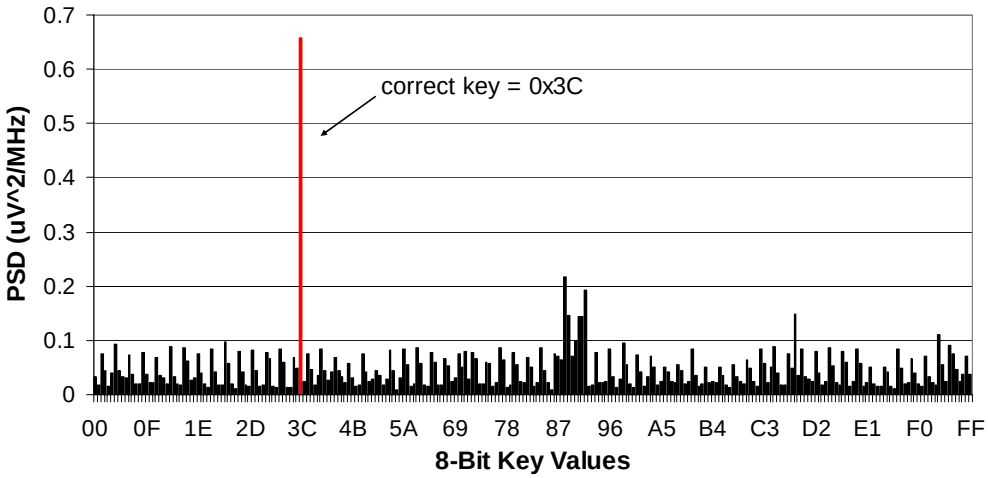

Figure 40: All Keys Search of DEMFA on PDA (Attacking Last S-Box)

4.4.1.4 Differential EM Spectrogram Analysis (DEMSA)

This thesis performs the differential EM spectrogram analysis (DEMSA) proposed by Gebotys *et al.* in [15] to PDA's running the Rijndael encryption algorithm. The purpose of this test is to investigate whether the spectrogram analysis is effective in extracting the AES encryption secret key. Recall from Section 3.5.2 that Spectrogram is a time-dependent frequency analysis. It consists of both time and frequency information, and therefore, has the advantage to pinpoint the time where there is a significant correlation between EM emanation and data values being manipulated.

Figure 41 below shows the differential spectrogram signal for the correct partition with the key value equals to 0x5C. Unlike DEMA and DEMFA, the differential spectrogram signal is computed by subtracting the averaged spectrogram for traces in group 0 from the averaged spectrogram for traces in group 1. The y-axis now represents the spectrogram magnitude and the x-axis represents the time. The window size of creating the spectrogram is 0.1 ms as illustrated in both Figure 41 and Figure 42. In between the 0.1 ms time intervals is the plot of the differential signal over a range of frequencies. The differential spectrogram signal in Figure 41 has significantly higher amount of area outside the ±2*STD_R region in blue. Please refer to section 3.2 for calculation of the STD_R region. Note the significant difference between times 0.7 ms to 0.9 ms in Figure 41. The differential signal in Figure 42, on the other hand, has significantly less spikes outside the standard deviation region for an incorrect key guess of 0xE6.

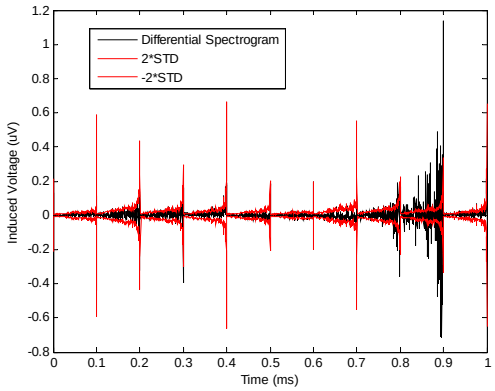

Figure 41: DEMSA on PDA (correct key=0x5C)

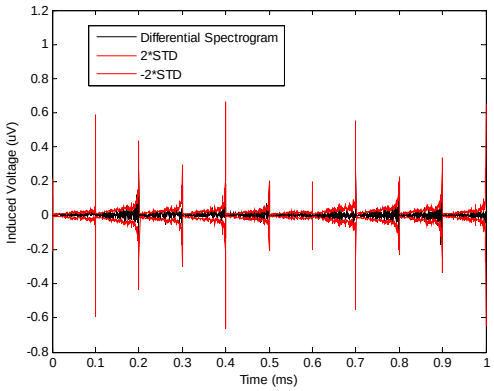

Figure 42: DEMSA on PDA (wrong key=0x37)

Figure 43 shows the all keys search of 256 possible S-Box key values. The analysis computes and compares the total area of spectrogram spikes that are beyond the ±2*STD_R region for all 256 possible key values of the AES S-Box being attacked. As demonstrated in Figure 43, the key value 0x5C has the highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. Hence, the correct key is successfully recovered using the differential EM spectrogram analysis.

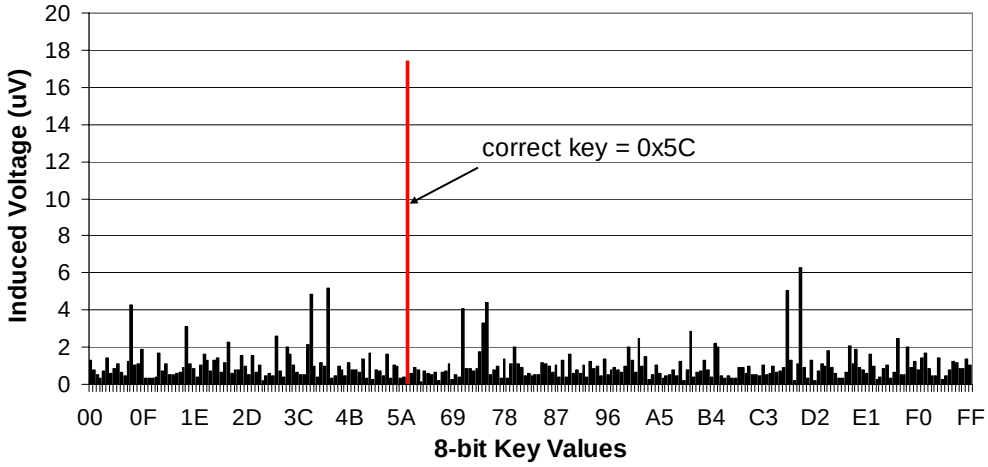

Figure 43: All Keys Search of DEMSA on PDA

4.4.2 EM Analysis on AES with Countermeasure

Without countermeasure implemented, DEMA already fails because of trace misalignment. Since DEMFA and DEMSA can both extract the correct key, the original AES encryption algorithm is modified to implement a countermeasure called Split Mask. For implementation details of the Split Mask countermeasure, refer to [9]. The purpose of this experiment is to investigate whether DEMFA and DEMSA can defect the masking countermeasure.

4.4.2.1 Differential EM Frequency Analysis (DEMFA)

First, the frequency analysis is performed on the AES implementation with Split Mask countermeasure. Figure 44 shows the all keys search of 256 possible key values. The analysis computes and compares the total area of PSD spikes that are beyond the ±2*STD_R region for all 256 possible key values of the AES S-Box being attacked. As shown in Figure 44, the correct key value 0x5C does not the highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. The correct key is not extracted using DEMFA. Hence, the Differential Frequency Analysis is not able to defect the Split Mask countermeasure.

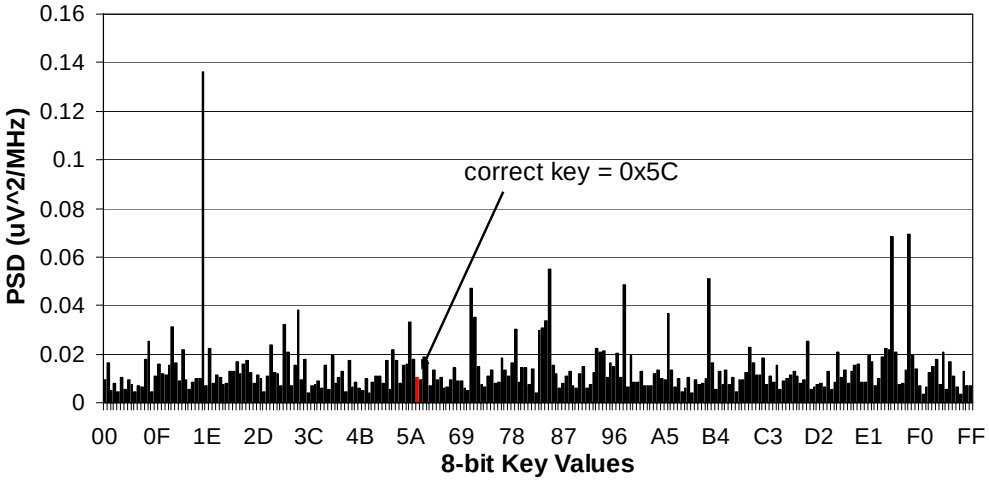

Figure 44: All Keys Search of DEMFA on PDA for AES with Countermeasure

4.4.2.2 Differential EM Spectrogram Analysis (DEMSA)

Next, the spectrogram analysis is performed on the AES implementation with Split Mask countermeasure. Figure 45 shows the all keys search of 256 possible key values. The

analysis computes and compares the total area of spectrogram spikes that are beyond the ±2*STD_R region for all 256 possible key values of the AES S-Box being attacked. As indicated in Figure 45, the key value 0x5C does not have the highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. Hence, the correct key also cannot be extracted using DEMSA when the Split Mask countermeasure is implemented.

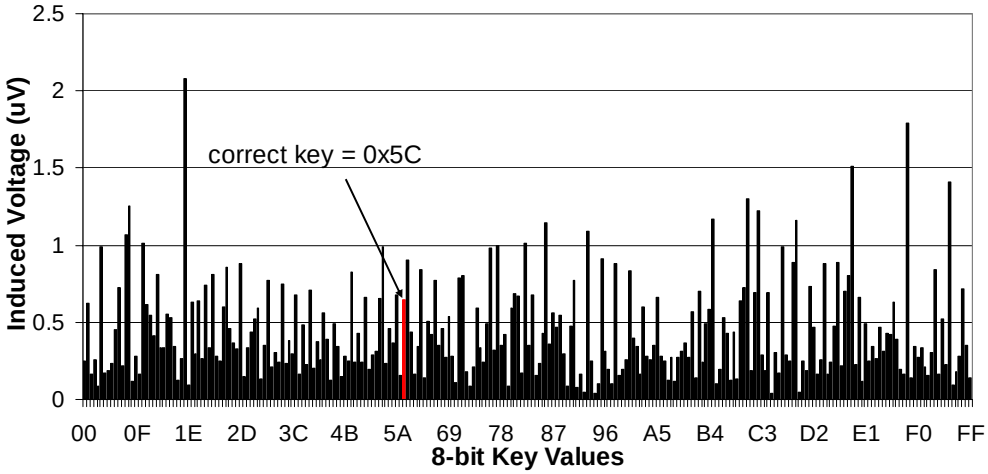

Figure 45: All Keys Search of DEMSA on PDA for AES with Countermeasure

4.4.2.3 Waddle's FFT 2DPA Attack

It is predicted that both frequency and spectrogram analysis cannot defect the masking countermeasure. Recall that the aim of these 2 attacks is to resolve trace misalignment encountered in first-order analysis. In fact, to overcome the masking countermeasure, a higher-order analysis is required. Waddle's FFT 2DPA attack is a second-order differential power analysis that is proposed to defeat masking countermeasure. Refer to Section 3.5.3 for details about this attack. The purpose of this test is to present experimental results of this attack.

Figure 46 shows the all keys search for 256 possible values. The differential time signal is computed by subtracting the average of autocorrelation of traces in group 0 from the average of autocorrelation of traces in group 1. The analysis computes and compares the total area of time spikes that are beyond the ±2*STD_R region for all 256 possible key values of the AES S-Box being attacked. As shown in Figure 46, the key value 0x5C

does not have the highest amount of total area of PSD spikes beyond the 2 times standard deviation threshold region among all keys. Hence, the correct key cannot be extracted using Waddle's FFT 2DPA analysis when the Split Mask countermeasure is implemented.

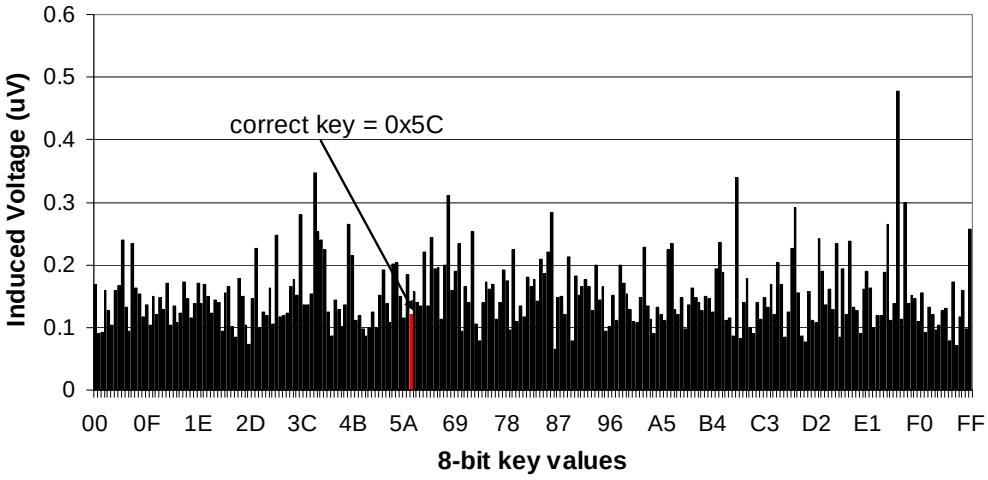

Figure 46: All Keys Search of FFT2DEMA on PDA for AES with Countermeasure

One possible reason of the failure of this attack is that the process of correlation amplifies the noise, hence increasing standard deviation and requiring more samples to reliably differentiate distributions. Due to measurement equipment limitations, the attack is not feasible in this thesis. The author also suggested that such attack will likely to work only if the traces are fairly short and the correlated bit influence fairly large. However, this is not the case in this experiment.

In summary, the experimental results from the PDA show that the EM emanation also leak key information about the Rijndael encryption algorithm. It is confirmed experimentally that DFA is more effective than previously researched DEMA when traces are misaligned in measurements.

4.5 Summary of Experimental Results

Table 3 and Table 4 below summarize all the experimental results presented in this chapter. Results have shown that the proposed differential frequency attack can extract secret key under severe trace misalignment conditions. In addition, the frequency attack is shown experimentally to be able to defect the desynchronization countermeasure.

Table 3: Summary of Experimental Results on ARM

Experiments on ARM	Normal AES	AES with Random Time Shifts Countermeasure
DEMA	Correct key extracted	Correct key NOT extracted
DEMFA	Correct key extracted	Correct key extracted
DPA	Correct key extracted	Correct key NOT extracted
DPFA	Correct key extracted	Correct key extracted

Table 4: Summary of Experimental Results on PDA

Experiments on PDA	Normal AES	AES with Split Mask Countermeasure
DEMA	Correct key NOT extracted	Correct key NOT extracted
DEMFA	Correct key extracted	Correct key NOT extracted
DEMSA	Correct key extracted	Correct key NOT extracted
FFT 2DEMA	Correct key NOT extracted	Correct key NOT extracted

To determine the signal quality, Table 5 summarizes the signal-to-noise ratio (SNR) of all data measured for this thesis. The SNR is computed as follows: SNR = $10*\log_{10}$(mean/standard deviation) (dB). The mean describes what is being measured, while the standard deviation represents noise the other interference. The standard deviation is not important in itself, but only in comparison to the mean. Therefore, it is valid to compare the SNR of EM and power data since the ratio of mean and standard deviation is unit-less.

As indicated in Table 5, the power data obtained from the ARM evaluation board has a higher SNR than the EM data obtained from the ARM evaluation board. Note that the higher the SNR, the better the signal quality. Moreover, the EM data obtained from the PDA has is slightly noisier than that from the ARM evaluation board since the PDA has a lower SNR.

Table 5: Comparison of Signal-to-Noise Ratio for All Measurements

Measurements	Normal AES	AES with Random Time Shifts Countermeasure
EM data from ARM	5.2525 dB	5.2671 dB
Power data from ARM	7.1339 dB	7.0647 dB
EM data from PDA	4.4677 dB	4.8179 dB

5 Discussion

This chapter discusses the experimental results, compares the DFA attack with previously researched side channel attacks, discusses the effectiveness of the DFA attack, and presents work to be done in the future.

5.1 Comparison of Experimental Results to Previous Research

5.1.1 Comparison to Previous Research on SPA and DPA

In [1], [7], and [10], the security of the newly developed encryption standard, AES, against power analysis is never put in practice. No real power measurements were presented in all these literatures. This thesis investigates the effectiveness of differential power analysis (DPA) of AES on the ARM Integrator/C7TDMI core module with a 32-bit processor. The secret key of the Rijndael encryption algorithm is successfully retrieved from the experiments. Experimental results indicate that like DES, AES is also vulnerable to DPA.

5.1.2 Comparison to Previous Research on SEMA and DEMA

Regarding EM measurements, a commercial EM probe is used for capturing EM signals from the ARM Integrator/C7TDMI core module and PDA in this thesis. By trial and error, it is observed that the magnetic probe with a shape of a 1-cm loop gives the best EM signal quality when the probe is placed in contact with the processor of the ARM Integrator/C7TDMI core module and PDA. No decapsulation is done to the processor chip. Comparing to previous research, different approaches were used to measure EM emanations. Some researchers built hand-made EM probes with different materials, shapes and sizes. EM probes were also located at a different distance from the chip in their experiments. In [5], Quisquater *et al.* used a simple flat coil so the variations of the electromagnetic field induce a current at the bounds. The sensor is placed under the smart card in the very close field. In [11], Gandolfi *et al.* used tiny hand-made probes, solenoids made of a coiled copper wire of outer diameters varying between 150 and 500 microns, for their EM measurements. They also stressed the importance to perform measurement as closely as possible to the chip by decapsulating the chip. Carlier *et al.* in [12] used

solenoid wires of copper consisting of a dozen of spires with a diameter of approximately 1 mm for their EM measurements. They placed the probe as close as possible to the FPGA to increase the magnetic flux collected by the probe. In [3], all EM emanations are measured either in the near field or in the far field away from the smart card unlike this thesis, [5], [11], and [12].

Regarding experimental results, this thesis shows a SEMA trace where one can see distinctively 12 rounds of AES 192-bit encryption computation. DEMA attacks of AES are also performed on the ARM Integrator/C7TDMI core module and on the PDA. Results indicate that DEMA is able to extract the secret AES key from the ARM Integrator/C7TDMI core module. However, DEMA fails on the PDA because of trace misalignment. The thesis presents conclusive EM analysis results from the ARM Integrator/C7TDMI core module and PDA both with 32-bit processors. Comparing to previous research, no real experiments of SEMA or DEMA were put in practice on 32-bit processors and PDA's. Gandolfi *et al.* only reported DEMA results of DES from an 8-bit CMOS microcontroller in [11]. The authors also compared DEMA results with DPA results in their paper. According to their experimental findings, although more noisy, EM measurements yield better differentials than power signals. DEMA's signal-to-noise ratio was higher than that of DPA. This thesis also compares the characteristics of EM emanation with power consumption using the ARM Integrator/C7TDMI core module. Unlike [11], results from the ARM evaluation board show that EM curves appear to be noisier than power curves. The signal-to-noise ratio of power traces is higher than that of EM traces. In [3], Agrawal *et al.* had successfully demonstrated DEMA attacks of DES on smart cards. Unlike this thesis, [5], [11], and [12], the raw EM signals were AM demodulated at different intermediate carrier frequencies (harmonics of the clock frequency). This thesis also attempts Agrawal *et al.*'s AM demodulation approach on DEMA attacks as researched in [3]. However, DEMA with AM demodulation is not feasible on PDA experiments. Therefore, no experimental results on AM demodulation are reported. One reason of failing is that AM demodulation only works best at higher frequencies, whereas experiments done in this thesis are limited in lower frequencies. To perform AM demodulation, the sampling frequency Fs must satisfy $Fs > 2 * Fc + BW$,

where *Fc* is the carrier frequency and *BW* is the bandwidth of the modulated signal. Because of limited scope memory as already discussed in Section 4.1.3, the sampling frequency (*Fs*) of the signals captured from the PDA is much lower than the fundamental clock frequency (*Fc*) of the PDA. Since the condition of $Fs > 2 * Fc + BW$ cannot be satisfied, it is thus not possible to perform the attack proposed in [3].

5.1.3 New Findings of Thesis

Past research focuses primarily on the security of smart cards, 8-bit processors, and FPGA's. No research has been done to study the threat of side channel attacks on 32-bit processors and PDA's. Most experimental results presented so far are attacks on DES implementation; AES implementation is never studied. Comparing to previous research, this thesis is the first to report conclusive side channel attack results of AES implementation on an ARM Integrator/C7TDMI core module and a PDA.

In addition, no methodology has been proposed to overcome these experimental issues in the past. Comparing to previous research, this thesis is the first one to address experimental issues encountered in PDA experiments where EM traces measured are temporally misaligned. This thesis proposed a new side channel attack called the Differential Frequency Analysis (DFA), which does not require perfect alignment of EM traces. Results from the ARM Integrator/C7TDMI core module also support the theory of DFA. It is confirmed experimentally that spikes appeared in the differential signal in time domain also appear in frequency domain, since any changes in the time domain signals would induce changes in the frequency domain signals. Hence, the correct key can be determined by examining the differential signal in the frequency domain.

It is also demonstrated experimentally that the new frequency-based attack can be applied to both power analysis and EM analysis. Experimental results from the ARM core module indicate that the new DFA attack is as effective as DPA and DEMA to extract the secret key of the Rijndael encryption algorithm when no countermeasure is implemented. For PDA experiments, DEMA fails when temporal misalignment of traces is severe. The DFA is shown to be effective to overcome this experimental issue; it is

able to reveal the secret key of the AES encryption. Therefore, it is shown that DFA has the advantage over DEMA since it can be applied under trace misalignment conditions.

In addition, results indicate that performing differential analysis in the frequency domain can defeat the desynchronization countermeasure against DPA and DEMA. Results show that the proposed first order DFA attack can efficiently overcome countermeasures that randomly insert delays without the need of launching a higher-order analysis. Thus, the proposed frequency-based attack is better then DPA and DEMA when the desynchronization countermeasure is implemented. However, when the Split Mask countermeasure is implemented to AES, DFA fails because DFA is a first order attack, it only aims to resolve trace misalignment problem. Therefore, higher order attack is required to defect the Split Mask countermeasure.

5.1.4 Comparison to Previous Research on High Order Attacks

Waddle *et al.* proposed an efficient second-order power analysis, FFT 2DPA, in [13]. Unlike Waddle's high order attack which uses Fast Fourier Transform to overcome the masking countermeasure as a higher order differential analysis, the new DFA attack proposed in this thesis uses FFT to eliminate misalignment problem in traces encountered in experimental measurements. The analysis presented by Waddle *et al.* is still performed in the time domain, whereas the Differential Frequency Analysis is performed in the frequency domain. In addition, DFA does not require computing the inverse FFT to transform the signal back to the time domain. Therefore, it requires less computation time. In addition, the attack proposed in this thesis use a threshold signal of multiple standard deviations of means than a constant threshold value to better characterize the significance of spikes found in the differential signal.

Once again, no real measurements are presented in [13]. This thesis is the first to put FFT 2DPA in practice. Results indicate that the secret key cannot be extracted from the AES implementation using Waddle's FFT 2DPA analysis when the Split Mask countermeasure is implemented. Experimental results indicate that the difference of means of the autocorrelation is very noisy, no clear spikes are present. One possible

reason of the failure of this attack is that the process of correlation amplifies the noise, hence increasing standard deviation and requiring more samples to reliably differentiate the autocorrelation distributions. Due to measurement equipment limitations, the attack is not feasible in this thesis. The author also suggested that such attack will likely to work only if the traces are fairly short and the correlated bit influence fairly large. However, this is not the case in the experiment in this thesis. In practice, FFT 2DPA suffers from too much noise amplification to be generally effective.

5.1.5 Comparison to Previous Research on Frequency Analysis

Differential spectrogram analysis (DSA) is a time-dependent frequency-based attack proposed by Gebotys *et al.* in [15]. Comparing to DFA, DSA has the advantage of being capable to locate the time segment where the differential time signal occurs. However, DSA is computationally longer than DFA. Unlike DSA, the new attack proposed in this thesis computes the power spectral density of each trace instead of the spectrogram. DFA is best in practice when the adversary already know about where the attack point occurs in time. Spectrogram comes in handy when the attacker has no idea where the differential occurs. Both DFA and DSA can extract the secret key when DEMA fails.

Regarding experiments on PDA, both DFA and DSA can overcome trace misalignment problem and extract the secret key effectively. However, DFA and DSA both fail when the Split Mask countermeasure is implemented to AES.

5.2 Advantages

There are 4 major advantages of using frequency domain signals in differential analysis. The main advantage of DFA is its capability of breaking a cryptosystem under severe temporal misalignment of traces. Before the DFA is proposed, one would need to first align traces and then perform normal differential time analysis such as DEMA and DPA. Some signal processing techniques such as cross-correlation might be able to align these temporally shifted traces. However, misalignment problems are not constant within each acquisition. Normally, there are more than a thousand traces in each acquisition. In order to compute the cross-correlation of 2 misaligned traces, one has to find out the shifts, n,

between these 2 traces. Hence, computing the cross-correlation for thousands of traces is difficult and computationally time consuming. With DFA, attackers are not required to perform the extra step to align every single temporally shifted trace. The DFA attack only requires a simple pre-processing stage to transform time signals to the frequency domain.

Secondly, the frequency domain signals are better than time domain signals for differential analysis. Frequency analysis may reveal loops and other repeating structures in a signal, which is not possible with time domain analysis. More importantly, frequency signals are less sensitive to random jitters and delays than time signals. Also, DFA is applicable to both power consumption and EM emanation.

Thirdly, DFA is proved to be capable of defecting desynchronization countermeasures that randomly inserts time shifts.

Lastly, our results showed that the use of DFA on the PDA has the advantage of reducing the key search space, unlike brute-force attacks. A brute-force attack is impossible for an AES 128-bit key since there are 2^{128} possible key searches. On the other hand, the Differential Frequency Analysis is only performed on each of the 8-bit S-Boxes. For AES 128-bit, there are 16 S-Boxes in total. Therefore, the key search space is reduced to 16 * 256 = 4096.

5.3 Disadvantages

There are 2 major disadvantages of using frequency domain signals in differential analysis. A major flaw in the DFA attack is the fact that it reveals no information of when data-dependant operations occur in time unlike normal DEMA and DPA. However, it is important to note that the main interest of an attacker is the secret key. Therefore, DFA is still an attractive technique for attacking mobile devices.

Secondly, there is a computation overhead of transforming time domain signals to frequency domain. The total runtime of DFA attacking an 8-bit S-Box is $\theta(nm\log m+256nm)$. As for the runtime of normal DEMA or DPA attack, there is no need

of preprocessing. The total runtime is $\theta(256nm)$. Although DFA has a slightly higher computational overhead, but it is worthwhile for its effectiveness of breaking the conventional symmetric key algorithm under severe trace misalignment experimental conditions.

5.4 Limitations

Differential Frequency Analysis (DFA) has fewer limitations than DEMA and DPA since it does not require perfect trace alignment to retrieve the secret key of AES. DFA is general and can be applied to embedded systems other than the PDA and to power traces as well as EM traces. The proposed DFA attack is not limited to AES. It could be used in other symmetric key algorithms such as DES, CAST 128, etc., where attacks at the output of S-Box are applicable.

All experiments presented in this thesis are subject to certain limitations. One of the major limitations is the number of traces acquired. Being a high-level programming language, Java is much slower than the assembly language. Since the number of traces measured is limited by the scope memory as described in Section 4.1.3, it is not possible to capture a large number of traces from the PDA unlike the ARM evaluation board. In fact, the AES encryption program written in assembly is at least hundred times faster than the AES Java program for the PDA. As a result, having a longer frame would sacrifice the number of traces acquired. In addition, other portable devices, unlike the PDA under test in this thesis, would shut down automatically after running the encryption over a certain number of times for security reason. Therefore, the number of traces acquired for DFA may be restricted by the automatic shutdown of these embedded systems.

In addition, the fixed scope memory size also restricts the frequency span of the signal captured. A longer frame not only sacrifices the number of traces acquired as discussed earlier but also the sampling rate. Recall the frequency span is half of the sampling rate. As a result, all the measured data from the PDA have a low frequency span. AM demodulation is not feasible to raw EM data obtained from the PDA.

Another limitation of DFA is that it requires the attacker to have knowledge about where the data dependency occurs. DFA works at best when the adversary can pinpoint the time where there is a significant correlation between EM emanation and data values being manipulated. Moreover, the attacker is required to have knowledge and control of the input plaintexts for the AES encryption in order to partition traces into 2 groupings.

5.5 Future Work

There are several experiments worth undertaking in the future. The first experiment is to investigate the effectiveness of the Differential Frequency Analysis for higher-order attacks. Due to the time constraint and some limitations discussed in this chapter, this thesis only uses the frequency analysis in first order attacks. The extension of DFA to higher order analysis is definitely the subject of future work.

The proposed DFA attack is not limited to its application on symmetric key algorithms. It could also be applied to other cryptographic algorithms such as public key algorithms. It is worth to experiment DFA attack on the Elliptic Curve Cryptography (ECC) which is widely implemented on embedded systems for its efficient computation.

Knowing that DFA is such a powerful technique, it is also of interest to investigate countermeasures for this frequency-based attack in order to better protect wireless embedded systems from adversaries in future research.

6 Conclusion

In summary, this thesis is the first to investigate the threat of EM analysis on PDA's. This thesis also presents for the first time EM analysis measurement results of AES implementation on a PDA.

This thesis first compares the characteristics of EM emanation with power consumption using the ARM Integrator/C7TDMI core module. Results show that EM curves appear to be noisier than power curves because the signal-to-noise ratio of power traces is higher than that of EM traces. For a normal AES implementation on the ARM Integrator/C7TDMI core module, both DEMA and DPA can extract the secret key easily. However, when the desynchronization countermeasure is implemented on AES, DEMA and DPA fail. The proposed Differential Frequency Analysis (DFA) is shown to be able to defect such countermeasure that inserts random delay. It is also illustrated in this thesis that the Differential Frequency Analysis can be applied to both power and EM data. It is also confirmed experimentally that spikes appeared in the differential signal in time domain also appear in frequency domain, since any changes in the time domain signals would induce changes in the frequency domain signals.

There is a difficulty of measuring power consumption from the PDA under test. Therefore, EM emanation is the preferred source for differential analysis in this thesis. It is shown that one can determine the key length used in a Rijndael encryption by simply observing a single EM trace in the attack known as the simple EM analysis (SEMA).

In addition, this thesis is the first to address the severe issues of trace misalignment on PDA experiments. This work proposes a new side channel attack called the Differential Frequency Analysis, which does not require perfect alignment of EM traces, thus supporting attacks on wireless embedded systems. DEMA fail when traces are misaligned. On the other hand, DFA is shown experimentally that it can overcome this problem and extract the secret key successfully.

Other than the new DFA attack, this thesis also performs other side channel attacks proposed in previous research. Results from differential spectrogram analysis (DSA) and Waddle's FFT 2DPA attack, and are also presented in this work. DSA is as effective as DFA in retrieving the secret key of a normal Rijndael implementation. Furthermore, this thesis studies the effectiveness of 2 previously researched countermeasures for the Rijndael encryption implementation: the desynchronization countermeasure and the Split Mask countermeasure. The proposed Differential Frequency Analysis (DFA) is shown to be able to defect countermeasure that inserts random delay. However, DFA and DSA fail when the Split Mask countermeasure is implemented to AES on the PDA. Waddle's FFT 2DPA attack is also performed on the Split Mask countermeasure. However, this attack cannot extract the secret key as it has claimed by the authors.

In conclusion, this thesis makes progress in side channel attacks and is important for future wireless embedded systems, which will increasingly demand higher levels of data security measures. This work can help users, developers, and product designers to gain a deeper understanding of the side channel security risks that these portable devices introduce.

References

[1] Paul Kocher, Joshua Jaffe, and Benjamin Jun, "Differential Power Analysis," *Advances in Cryptography – CRYPTO'99*, *Lecture Notes in Computer Science vol. 1666*, pp. 388-397, Springer-Verlag 1999

[2] National Institute of Standards and Technology, "FIPS 197 Advanced Encryption Standard," Available at *http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf*, 2001

[3] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi, "The EM Side-Channel(s): Attacks and Assessment Methodologies," Available at *http://www.research.ibm.com/intsec/emf.html*, 2002

[4] Thomas S. Messerges, "Using Second-Order Power Analysis to Attack DPA Resistant Software," *CHES 2000*, *Lecture Notes in Computer Science vol. 1965*, pp.238-251, Springer-Verlag 2000

[5] Jean-Jacques Quisquater and David Samyde, "Electromagnetic analysis (EMA): measures and countermeasures for smart cards," *Lectures Notes in Computer Science vol. 2140*, pp. 200-210, 2001

[6] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao and Pankaj Rohtagi, "Towards Sound Approaches to Counteract Power Analysis Attacks," *Advances in Cryptography – CRYPTO'99*, *Lecture Notes in Computer Science vol. 1666*, pp. 398-412, Springer-Verlag 1999

[7] Louis Goubin and Jacques Patarin, "DES and Differential Power Analysis – The Duplication Method," *CHES 1999*, *Lecture Notes in Computer Science vol. 1717*, pp. 158-172, Springer-Verlag 1999

[8] Kouichi Itoh, Masahiko Takenaka, and Naoya Torii, "DPA Countermeasure Based on the "Masking Method"," *Lecture Notes in Computer Science 2288*, pp.4440-456, Springer-Verlag 2002

[9] C.H. Gebotys, C.C. Tiu, and X. Chen, "A Countermeasure for EM Attack of a Wireless PDA," *To appear in Proceedings of IEEE International Conference on Information Technology Coding and Computing, session number 90*, 2005

[10] Jovan Dj. Golić, "Multiplicative Masking and Power Analysis of AES," *Lecture Notes in Computer Science vol. 2523*, pp.198-212, Springer-Verlag 2003

[11] Karine Gandolfi, Christophe Mourtel and Francis Olivier, "Electromagnetic analysis: Concrete results," *Lecture Notes in Computer Science vol. 2162*, pp. 251-261, Springer-Verlag 2001

[12] Vincent Carlier, Hervé Chabanne, Emmanuelle Dottax, and Hervé Pelletier, "Electromagnetic Side-Channels of an FPGA Implementation of AES," Available at *http://eprint.iacr.org/2004/145.pdf*, 2004

[13] J.Waddle, and D.Wagner "Towards Efficient Second-Order Power Analysis," *CHES 2004*, *Lecture Notes in Computer Science vol. 3156*, pp. 1-15, Springer-Verlag 2004

[14] Eric Kohlbrenner, Dana Morris, Brett Morris, "The Java Virtual Machine Model," Available at *http://cne.gmu.edu/itcore/virtualmachine/jvm.htm*, 1999

[15] C.H. Gebotys, C.C. Tiu, and S. Ho, "EM Analysis of Rijndael and ECC on a Wireless Java-based PDA," S*ubmitted to CHES 2005*, 2005

[16] ARM Limited, "ARM7TDMI (Rev 4) Technical Reference Manual," Available at *http://www.arm.com/documentation/ARMProcessor_Cores/index.html*, 2001

[17] ARM Limited, "Integer/CM7TDMI User Guide," Available at *http://www.arm.com/documentation/Boards_and_Firmware/index.html*, 2001

[18] Electro-Metrics Inc., "Broadband Amplifier Model EM-6992 Instruction Manual," Available at *http://www.electro-metrics.com*, 2002

[19] Electro-Metrics Inc., "Near Field Probe Set Broadband Response Model EM-6992 Instruction Manual," Available at *http://www.electro-metrics.com, 2002*

[20] Dr.Brian Gladman, "A Specification for Rijndael, the AES Algorithm," Available at *fp.gladman.plus.com/cryptography_technology/rijndael/aes.spec.311.pdf*, 2003

[21] J.Daemen, V.Rijmen "Resistance against Implementation Attacks: A comparative study of the AES proposals," Available at *http://csrc.nist.gov/CryptoToolkit/aes/round1/conf2/papers/daemen.pdf*, Rome, 1999

[22] Tektronix, "TDS7254 Digital Phosphor Oscilloscope User Manual," Available at *http:// http://www.tek.com/site/mn/mnfinder_detail/1,1096,,00.html?id=54&pn=071701002*, 2003

[23] Tektronix, "TCP202 DC/AC Inductive Current Probe Instruction Manual," Available at *http://www.tek.com/site/mn/mnfinder_detail/1,1096,,00.html?id=3751&pn=070954202*, 2004

[24] National Institute of Standards and Technology, "FIPS 46-3 Data Encryption Standard (DES)," Available at *http://csrc.nist.gov/publications/fips/fips46-3/fips46-3.pdf*, 1999

[25] S. Smith, "The Scientist and Engineer's Guide to Digital Signal Processing (online version) – Chapter 8: The Discrete Fourier Transform", Available at *http://www.dspguide.com/ch8.htm*, California Technical Publishing 1997

[26] W.H. Press, S.A. Teukolsky, W.T. Vetterling, B.P. Flannery, "Numerical Recipes in C: The Art of Scientific Computing Second Edition (online version)," pp. 549-550, Available at *http://www.library.cornell.edu/nr/bookcpdf/c13-4.pdf*, Cambridge University Press1992

[27] M. Akkar, R. Bevan, P. Dischamp, D. Moyart, "Power analysis, What is Now Possible," *Lecture Notes in Computer Science vol. 1976*, pp. 489-502, Springer-Verlag 2000

[28] A. Raghunathan, N. Potlapally, S. Ravi, "Securing Wireless Data: System architecture challenges," *Proceedings of ISSS 2002*, pp.195-200, 2002

[29] P. Kocher, R. Lee, G. McGraw, A. Raghunathan, S. Ravi, "Security as a New Dimension in Embedded System Design, " *Proceedings of DAC 2004*, Available at *http://videos.dac.com/41st/papers/46_1.pdf*, 2004

[30] S. Chari, C. Jutla, J.R. Rao, P. Rohatgi, "A Cautionary Note Regarding Evaluation of AES Candidates on Smart-Cards," Available at *http://csrc.nist.gov/CryptoToolkit/aes/round1/conf2/papers/chari.pdf*, Rome, 1999

[31] T. Lash, "A Study of Power Analysis and the Advanced Encryption Standard," MS Scholarly Paper, George Mason University, Feb. 2002

[32] J.S. Coron, L. Goubin, "On Boolean and Arithmetic Masking Against Differential Power Analysis," CHES 2000, *Lecture Notes in Computer Science vol. 1965*, pp. 231, Springer-Verlag 2000

[33] Kingpin, Mudge, "Security Analysis of the Palm Operating System and its Weaknesses Against Malicious Code Threats," Available at *http://downloads.securityfocus.com/library/security_analysis_palm_os.pdf*, Washington, 2001

Appendix

Appendix 1 – MATLAB program of DFA attack

```
% pdaDFA -- Differential Frequency Analysis for PDA
% datFileStr = data file name from scope (string)
% hdrFileStr = header file name from scope(string)

function pdaDFA(datFileStr,hdrFileStr)

raw_samples = load(datFileStr);
res = dlmread(hdrFileStr);

record_length=res(1);
frame_count=res(6)/2;
tot_frame=res(6);
hdr_sample_freq=1/res(2);
hdr_sample_period=res(2);
f=0:hdr_sample_freq/1e6/record_length:hdr_sample_freq/1e6-
hdr_sample_freq/1e6/record_length;
t=0:hdr_sample_period:hdr_sample_period*record_length-hdr_sample_period;

samples = zeros(frame_count*2, record_length);
for k = 0:(frame_count*2-1)
    samples(k+1,:) = (raw_samples(k*record_length+1:(k+1)*record_length))';
end

samples = samples';
clear raw_samples;
save('samples.mat');

sumPeaks=zeros(256, 1);
save sumPeaks.mat datFileStr sumPeaks;
clear;

for k=0:255
    keyStr = num2str(dec2hex(k, 2));
    save('keyStr');
    display(keyStr);

clear;
load('keyStr');
load('samples.mat');

if(size(samples,2)==record_length)
    display('need transpose');
    samples = samples';
end

javaMethod('pda1SBoxSplitPt', 'splitPt', hex2dec(keyStr), tot_frame);

% Set 0
index0 = load(strcat(keyStr, '_pda_index0.txt'));
```

```
array0 = zeros(record_length, length(index0));
array0_length = length(index0);

for i=1:length(index0)
    array0(:,i) = samples(:, index0(i)+1);
end

% Set 1
index1 = load(strcat(keyStr, '_pda_index1.txt'));
array1 = zeros(record_length, length(index1));
array1_length = length(index1);

for i=1:length(index1)
    array1(:, i) = samples(:, index1(i)+1);
end

delete(strcat(keyStr, '_pda_index0.txt'));
delete(strcat(keyStr, '_pda_index1.txt'));

clear samples;

% set 0
Y_bit0=fft(array0);

tmp = zeros(record_length/2, length(index0));
tmp=Y_bit0(1:record_length/2,:);

Pyy_bit0 = zeros(record_length/2, length(index0));
for i=1:record_length/2
    Pyy_bit0(i,:) = (tmp(i,:).*conj(tmp(i,:)))/record_length;
end

clear Y_bit0;
clear tmp;
clear index0;

% compute PSD
PSD_mean_bit0 = mean(Pyy_bit0');
PSD_std_bit0 = std(Pyy_bit0');
clear Pyy_bit0;

%set1
Y_bit1=fft(array1);

tmp = zeros(record_length/2, length(index1));
tmp=Y_bit1(1:record_length/2,:);

Pyy_bit1 = zeros(record_length/2, length(index1));
for i=1:record_length/2
    Pyy_bit1(i,:) = (tmp(i,:).*conj(tmp(i,:)))/record_length;
end

clear Y_bit1;
clear tmp;
clear index1;

% compute PSD
```

```
PSD_mean_bit1 = mean(Pyy_bit1');
PSD_std_bit1 = std(Pyy_bit1');
clear Pyy_bit1;

% differential PSD
% for each point of frequency, check whether it's outside +/- 2*STD
std_dom= sqrt( (PSD_std_bit0').^2./array0_length +
(PSD_std_bit1').^2./array1_length);
PSD_mean_diff = PSD_mean_bit0-PSD_mean_bit1;
load('sumPeaks');

for i=1:length(PSD_mean_diff)
    if(PSD_mean_diff(i) > 2*std_dom(i))
        sumPeaks(hex2dec(keyStr)+1)=sumPeaks(hex2dec(keyStr)+1)+PSD_mean_diff(i)-
2*std_dom(i);
    elseif (PSD_mean_diff(i) < -2*std_dom(i))
        sumPeaks(hex2dec(keyStr)+1)=sumPeaks(hex2dec(keyStr)+1)-2*std_dom(i)-
PSD_mean_diff(i);
    end
end

% save data into text file
save sumPeaks.mat datFileStr sumPeaks;

    clear;
    load('keyStr');
    k = hex2dec(keyStr) + 1;
end

% Key Guess
[maxSumPeaks, correct_key] = max(sumPeaks);
correct_key = correct_key - 1;
display(strcat('Correct key is 0x',dec2hex(correct_key, 2)));

load('sumPeaks.mat');
dlmwrite(strcat(datFileStr(1:length(datFileStr)-4), '_DFA.txt'), sumPeaks);
delete('samples.mat');
delete('sumPeaks.mat');
delete('keyStr.mat');

return;
```

Appendix 2 – Java program of AES encryption algorithm on PDA

```
public class AESencrypt
{
    private final int Nb = 4; // words in a block, always 4 for now
    private int Nk; // number of 32-bit words, 4 (128-bit), 6 (192-bit), 8 (256-bit)
    private int Nr; // number of rounds, = Nk + 6
    private int wCount; // position in w for RoundKey (= 0 each encrypt)
    private AEStables tab; // all the tables needed for AES
    private byte[] w; // the expanded key
    private long[] t; // for 4 SBox implementation
    private byte[][] state; // the state array

    // AESencrypt: constructor for class. Mainly expands key
    public AESencrypt(byte[] key)
    {
        // words in a key, = 4, or 6, or 8
        if (key.length == 16)
            Nk = 4;
        else if (key.length == 24)
            Nk = 6;
        else if (key.length == 32)
            Nk = 8;

        Nr = Nk + 6; // corresponding number of rounds
        tab = new AEStables(); // class to give values of various functions
        w = new byte[4*Nb*(Nr+1)]; // room for expanded key
        t = new long[4];
        state = new byte[4][Nb];
        KeyExpansion(key, w); // length of w depends on Nr
    }

    public void normalAES(byte[] in, byte[] out)
    {
        wCount = 0; // count bytes in expanded key throughout encryption
        Copy.copy(state, in); // actual component-wise copy
        AddRoundKey(state); // xor with expanded key
        for (int round = 1; round < Nr; round++)
        {
            SubBytes(state); // S-box substitution
            ShiftRows(state); // mix up rows
            MixColumns(state); // complicated mix of columns
            AddRoundKey(state); // xor with expanded key
        }
        SubBytes(state); // S-box substitution
        ShiftRows(state); // mix up rows
        AddRoundKey(state); // xor with expanded key
        Copy.copy(out, state);
    }

    public void normalAESwSplitMask(byte[] in, byte[] out)
    {
        wCount = 0; // count bytes in expanded key throughout encryption
        Copy.copy(state, in); // actual component-wise copy
```

```
    AddRoundKey(state); // xor with expanded key
    for (int round = 1; round < Nr; round++)
    {
        SubBytesSplitMask(state); // S-box substitution
        ShiftRows(state); // mix up rows
        MixColumns(state); // complicated mix of columns
        AddRoundKeySplitMask(state); // xor with expanded key
    }
    SubBytesSplitMask(state); // S-box substitution
    ShiftRows(state); // mix up rows
    AddRoundKeySplitMask(state); // xor with expanded key
    Copy.copy(out, state);
}

public void optAES(byte[] in, byte[] out)
{
    wCount = 0; // count bytes in expanded key throughout encryption
    Copy.copy(state, in); // actual component-wise copy
    AddRoundKey(state); // xor with expanded key
    t[0] = 0;
    t[1] = 0;
    t[2] = 0;
    t[3] = 0;

    for (int round = 1; round < Nr; round++)
    {
        t[0] = tab.Te0(state[0][0]) ^ tab.Te1(state[1][1]) ^
               tab.Te2(state[2][2]) ^ tab.Te3(state[3][3]);
        t[1] = tab.Te0(state[1][0]) ^ tab.Te1(state[2][1]) ^
               tab.Te2(state[3][2]) ^ tab.Te3(state[0][3]);
        t[2] = tab.Te0(state[2][0]) ^ tab.Te1(state[3][1]) ^
               tab.Te2(state[0][2]) ^ tab.Te3(state[1][3]);
        t[3] = tab.Te0(state[3][0]) ^ tab.Te1(state[0][1]) ^
               tab.Te2(state[1][2]) ^ tab.Te3(state[2][3]);

        state[0][0] = (byte) (t[0] >> 24);
        state[1][0] = (byte) (t[0] >> 16);
        state[2][0] = (byte) (t[0] >> 8);
        state[3][0] = (byte) (t[0]);

        state[0][1] = (byte) (t[3] >> 16);
        state[1][1] = (byte) (t[3] >> 8);
        state[2][1] = (byte) (t[3]);
        state[3][1] = (byte) (t[3] >> 24);

        state[0][2] = (byte) (t[2] >> 8);
        state[1][2] = (byte) (t[2]);
        state[2][2] = (byte) (t[2] >> 24);
        state[3][2] = (byte) (t[2] >> 16);

        state[0][3] = (byte) (t[1]);
        state[1][3] = (byte) (t[1] >> 24);
        state[2][3] = (byte) (t[1] >> 16);
        state[3][3] = (byte) (t[1] >> 8);

        AddRoundKey(state); // xor with expanded key
    }
```

```
    // table Te4 doesn't have the MixColumn operation
    t[0] = (tab.Te4(state[2][3]) & 0xff0000L)
           ^ (tab.Te4(state[0][0]) & 0xff000000L)
           ^ (tab.Te4(state[0][2]) & 0xff00L)
           ^ (tab.Te4(state[2][1]) & 0xffL);
    t[1] = (tab.Te4(state[3][2]) & 0xff0000L)
           ^ (tab.Te4(state[1][3]) & 0xff000000L)
           ^ (tab.Te4(state[1][1]) & 0xff00L)
           ^ (tab.Te4(state[3][0]) & 0xffL);
    t[2] = (tab.Te4(state[0][1]) & 0xff0000L)
           ^ (tab.Te4(state[2][2]) & 0xff000000L)
           ^ (tab.Te4(state[2][0]) & 0xff00L)
           ^ (tab.Te4(state[0][3]) & 0xffL);
    t[3] = (tab.Te4(state[1][0]) & 0xff0000L)
           ^ (tab.Te4(state[3][1]) & 0xff000000L)
           ^ (tab.Te4(state[3][3]) & 0xff00L)
           ^ (tab.Te4(state[1][2]) & 0xffL);

    state[0][0] = (byte) (t[0] >> 24);
    state[1][0] = (byte) (t[1] >> 8);
    state[2][0] = (byte) (t[2] >> 24);
    state[3][0] = (byte) (t[3] >> 8);

    state[0][1] = (byte) (t[2] >> 16);
    state[1][1] = (byte) (t[3]);
    state[2][1] = (byte) (t[0] >> 16);
    state[3][1] = (byte) (t[1]);

    state[0][2] = (byte) (t[0] >> 8);
    state[1][2] = (byte) (t[1] >> 24);
    state[2][2] = (byte) (t[2] >> 8);
    state[3][2] = (byte) (t[3] >> 24);

    state[0][3] = (byte) (t[2]);
    state[1][3] = (byte) (t[3] >> 16);
    state[2][3] = (byte) (t[0]);
    state[3][3] = (byte) (t[1] >> 16);

    AddRoundKey(state); // xor with expanded key
    Copy.copy(out, state);
}

// Cipher: actual AES encrytion
public void optAESwSplitMask(byte[] in, byte[] out)
{
    wCount = 0; // count bytes in expanded key throughout encryption
    Copy.copy(state, in); // actual component-wise copy
    AddRoundKey(state); // xor with expanded key
    long t0 = 0;
    long t1 = 0;
    long t2 = 0;
    long t3 = 0;
    long m0 = 0;
    long m1 = 0;
    long m2 = 0;
    long m3 = 0;
```

```
for (int round = 1; round < Nr; round++)
{
    t0 = (tab.MTe0(state[0][0]) ^ tab.MTe1(state[1][1])
          ^ tab.MTe2(state[2][2]) ^ tab.MTe3(state[3][3])) & 0xffffffffL;
    m0 = (tab.MOpt(state[0][0]) ^ tab.MOpt(state[1][1])
          ^ tab.MOpt(state[2][2]) ^ tab.MOpt(state[3][3])) & 0xffffffffL;
    t0 = t0 ^ m0;

    t1 = (tab.MTe0(state[1][0]) ^ tab.MTe1(state[2][1])
          ^ tab.MTe2(state[3][2]) ^ tab.MTe3(state[0][3])) & 0xffffffffL;
    m1 = (tab.MOpt(state[1][0]) ^ tab.MOpt(state[2][1])
          ^ tab.MOpt(state[3][2]) ^ tab.MOpt(state[0][3])) & 0xffffffffL;
    t1 = t1 ^ m1;

    t2 = (tab.MTe0(state[2][0]) ^ tab.MTe1(state[3][1])
          ^ tab.MTe2(state[0][2]) ^ tab.MTe3(state[1][3])) & 0xffffffffL;
    m2 = (tab.MOpt(state[2][0]) ^ tab.MOpt(state[3][1])
          ^ tab.MOpt(state[0][2]) ^ tab.MOpt(state[1][3])) & 0xffffffffL;
    t2 = t2 ^ m2;

    t3 = (tab.MTe0(state[3][0]) ^ tab.MTe1(state[0][1])
          ^ tab.MTe2(state[1][2]) ^ tab.MTe3(state[2][3])) & 0xffffffffL;
    m3 = (tab.MOpt(state[3][0]) ^ tab.MOpt(state[0][1])
          ^ tab.MOpt(state[1][2]) ^ tab.MOpt(state[2][3])) & 0xffffffffL;
    t3 = t3 ^ m3;

    state[0][0] = (byte) (t0 >> 24);
    state[1][0] = (byte) (t0 >> 16);
    state[2][0] = (byte) (t0 >> 8);
    state[3][0] = (byte) (t0);

    state[0][1] = (byte) (t3 >> 16);
    state[1][1] = (byte) (t3 >> 8);
    state[2][1] = (byte) (t3);
    state[3][1] = (byte) (t3 >> 24);

    state[0][2] = (byte) (t2 >> 8);
    state[1][2] = (byte) (t2);
    state[2][2] = (byte) (t2 >> 24);
    state[3][2] = (byte) (t2 >> 16);

    state[0][3] = (byte) (t1);
    state[1][3] = (byte) (t1 >> 24);
    state[2][3] = (byte) (t1 >> 16);
    state[3][3] = (byte) (t1 >> 8);

    AddRoundKey(state); // xor with expanded key
}

// table Te4 doesn't have the MixColumn operation
t0 = ((tab.MTe4(state[2][3]) & 0xff0000L)
      ^ (tab.MTe4(state[0][0]) & 0xff000000L)
      ^ (tab.MTe4(state[0][2]) & 0xff00L)
      ^ (tab.MTe4(state[2][1]) & 0xffL)) & 0xffffffffL;
t1 = ((tab.MTe4(state[3][2]) & 0xff0000L)
      ^ (tab.MTe4(state[1][3]) & 0xff000000L)
      ^ (tab.MTe4(state[1][1]) & 0xff00L)
```

```
            ^ (tab.MTe4(state[3][0]) & 0xffL)) & 0xffffffffL;
        t2 = ((tab.MTe4(state[0][1]) & 0xff0000L)
            ^ (tab.MTe4(state[2][2]) & 0xff000000L)
            ^ (tab.MTe4(state[2][0]) & 0xff00L)
            ^ (tab.MTe4(state[0][3]) & 0xffL)) & 0xffffffffL;
        t3 = ((tab.MTe4(state[1][0]) & 0xff0000L)
            ^ (tab.MTe4(state[3][1]) & 0xff000000L)
            ^ (tab.MTe4(state[3][3]) & 0xff00L)
            ^ (tab.MTe4(state[1][2]) & 0xffL)) & 0xffffffffL;

        m0 = ((tab.MOpt(state[2][3]) & 0xff0000L)
            ^ (tab.MOpt(state[0][0]) & 0xff000000L)
            ^ (tab.MOpt(state[0][2]) & 0xff00L)
            ^ (tab.MOpt(state[2][1]) & 0xffL)) & 0xffffffffL;
        m1 = ((tab.MOpt(state[3][2]) & 0xff0000L)
            ^ (tab.MOpt(state[1][3]) & 0xff000000L)
            ^ (tab.MOpt(state[1][1]) & 0xff00L)
            ^ (tab.MOpt(state[3][0]) & 0xffL)) & 0xffffffffL;
        m2 = ((tab.MOpt(state[0][1]) & 0xff0000L)
            ^ (tab.MOpt(state[2][2]) & 0xff000000L)
            ^ (tab.MOpt(state[2][0]) & 0xff00L)
            ^ (tab.MOpt(state[0][3]) & 0xffL)) & 0xffffffffL;
        m3 = ((tab.MOpt(state[1][0]) & 0xff0000L)
            ^ (tab.MOpt(state[3][1]) & 0xff000000L)
            ^ (tab.MOpt(state[3][3]) & 0xff00L)
            ^ (tab.MOpt(state[1][2]) & 0xffL)) & 0xffffffffL;

        t0 = t0 ^ m0 ^ tab.maskOpt();
        t1 = t1 ^ m1 ^ tab.maskOpt();
        t2 = t2 ^ m2 ^ tab.maskOpt();
        t3 = t3 ^ m3 ^ tab.maskOpt();

        state[0][0] = (byte) (t0 >> 24);
        state[1][0] = (byte) (t1 >> 8);
        state[2][0] = (byte) (t2 >> 24);
        state[3][0] = (byte) (t3 >> 8);

        state[0][1] = (byte) (t2 >> 16);
        state[1][1] = (byte) (t3);
        state[2][1] = (byte) (t0 >> 16);
        state[3][1] = (byte) (t1);

        state[0][2] = (byte) (t0 >> 8);
        state[1][2] = (byte) (t1 >> 24);
        state[2][2] = (byte) (t2 >> 8);
        state[3][2] = (byte) (t3 >> 24);

        state[0][3] = (byte) (t2);
        state[1][3] = (byte) (t3 >> 16);
        state[2][3] = (byte) (t0);
        state[3][3] = (byte) (t1 >> 16);

        AddRoundKey(state); // xor with expanded key
        Copy.copy(out, state);
    }

    // KeyExpansion: expand key, byte-oriented code, but tracks words
```

```
// KeyExpansion generates a total of Nb*(Nr+1) words each of 4-byte
private void KeyExpansion(byte[] key, byte[] w)
{
    byte[] temp = new byte[4];
    // first just copy key to w
    int j = 0;
    while (j < 4*Nk)
    {
        w[j] = key[j++];
    }
    // here j == 4*Nk;
    int i;
    while(j < 4*Nb*(Nr+1))
    {
        i = j/4; // j is always multiple of 4 here
        // handle everything word-at-a time, 4 bytes at a time
        for (int iTemp = 0; iTemp < 4; iTemp++)
            temp[iTemp] = w[j-4+iTemp];

        if (i % Nk == 0)
        {
            byte ttemp, tRcon;
            byte oldtemp0 = temp[0];
            for (int iTemp = 0; iTemp < 4; iTemp++)
            {
                if (iTemp == 3) ttemp = oldtemp0;
                else ttemp = temp[iTemp+1];
                if (iTemp == 0) tRcon = tab.Rcon(i/Nk);
                else tRcon = 0;
                temp[iTemp] = (byte)(tab.SBox(ttemp) ^ tRcon);
            }
        }
        else if (Nk > 6 && (i%Nk) == 4)
        {
            for (int iTemp = 0; iTemp < 4; iTemp++)
                temp[iTemp] = tab.SBox(temp[iTemp]);
        }

        for (int iTemp = 0; iTemp < 4; iTemp++)
            w[j+iTemp] = (byte)(w[j - 4*Nk + iTemp] ^ temp[iTemp]);
        j = j + 4;
    }
}

// ShiftRows: simple circular shift of rows 1, 2, 3 by 1, 2, 3
private void ShiftRows(byte[][] state)
{
    byte[] t = new byte[4];
    for (int r = 1; r < 4; r++)
    {
        for (int c = 0; c < Nb; c++)
            t[c] = state[r][(c + r)%Nb];
        for (int c = 0; c < Nb; c++)
            state[r][c] = t[c];
    }
}
```

```
// MixColumns: complex and sophisticated mixing of columns
private void MixColumns(byte[][] s)
{
    int[] sp = new int[4];
    byte b02 = (byte)0x02, b03 = (byte)0x03;
    for (int c = 0; c < 4; c++)
    {
        sp[0] = tab.FFMul(b02, s[0][c]) ^ tab.FFMul(b03, s[1][c]) ^
                s[2][c] ^ s[3][c];
        sp[1] = s[0][c] ^ tab.FFMul(b02, s[1][c]) ^
                tab.FFMul(b03, s[2][c]) ^ s[3][c];
        sp[2] = s[0][c] ^ s[1][c] ^
                tab.FFMul(b02, s[2][c]) ^ tab.FFMul(b03, s[3][c]);
        sp[3] = tab.FFMul(b03, s[0][c]) ^ s[1][c] ^
                        s[2][c] ^ tab.FFMul(b02, s[3][c]);
        for (int i = 0; i < 4; i++)
            s[i][c] = (byte)(sp[i]);
    }
}

// AddRoundKey: xor a portion of expanded key with state
private void AddRoundKey(byte[][] state)
{
    for (int c = 0; c < Nb; c++)
        for (int r = 0; r < 4; r++)
            state[r][c] = (byte)(state[r][c] ^ w[wCount++]);
}

// need to unmask after each round for the normal AES impl.
private void AddRoundKeySplitMask(byte[][] state)
{
    for (int c = 0; c < Nb; c++)
    {
        for (int r = 0; r < 4; r++)
        {
            state[r][c] = (byte)(state[r][c] ^ tab.maskNorm());
            state[r][c] = (byte)(state[r][c] ^ w[wCount++]);
        }
    }
}

// SubBytes: apply Sbox substitution to each byte of state
private void SubBytes(byte[][] state)
{
    for (int row = 0; row < 4; row++)
        for (int col = 0; col < Nb; col++)
            state[row][col] = tab.SBox(state[row][col]);
}

// need to unmask after each round for the normal AES impl.
private void SubBytesSplitMask(byte[][] state)
{
    for (int row = 0; row < 4; row++)
    {
        for (int col = 0; col < Nb; col++)
        {
            byte index = state[row][col];
```

```
                state[row][col] = tab.MSBox(index);
                state[row][col] = (byte)(state[row][col] ^ tab.M(index));
            }
        }
    }
}

public class AEStables
{
    public AEStables()
    {
        loadE();
        loadL();
        loadInv();
        loadS();
        loadInvS();
        loadPowX();
        genMTablesOrigAES();
        genMTablesOptAES();
    }

    private byte[] E = new byte[256]; // "exp" table (base 0x03)
    private byte[] L = new byte[256]; // "Log" table (base 0x03)
    private byte[] S = new byte[256]; // SubBytes table
    private byte[] invS = new byte[256]; // inverse of SubBytes table
    private byte[] inv = new byte[256]; // multiplicative inverse table
    private byte[] powX = new byte[15]; // powers of x = 0x02

    /**
     * MTe0 to MTe4 are the 4 masked S-Boxes for split mask countermeasure
     * on the optimized AES implementation (same as ARM)
     * MOpt is the M table for the optimized AES
     */
    private long[] MTe0 = new long[256]; // masked Te0
    private long[] MTe1 = new long[256]; // masked Te1
    private long[] MTe2 = new long[256]; // masked Te2
    private long[] MTe3 = new long[256]; // masked Te3
    private long[] MTe4 = new long[256]; // masked Te4
    private long[] MOpt = new long[256]; // M table
    // this is the mask for the optimized AES implementation
    private int maskOpt;

    /**
     * MS the masked S-Box for split mask countermeasure
     * on the normal AES implementation with only 1 S-Box
     * MOpt is the M table for the normal AES
     */
    private byte[] MS = new byte[256]; // masked SBox
    private byte[] M = new byte[256]; // M table for masked SBox
    // this is the mask for the normal AES implementation
    private byte maskNorm;

    /**
     * Te0 to Te4 are S-Boxes for the optimized AES implementation
     * by Dr. Brian Gladman, they are taken from the assembly
     * code from the ARM
     */
```

```
private long[] Te0 = {
0xc66363a5L, 0xf87c7c84L, 0xee777799L, 0xf67b7b8dL,
0xfff2f20dL, 0xd66b6bbdL, 0xde6f6fb1L, 0x91c5c554L,
0x60303050L, 0x02010103L, 0xce6767a9L, 0x562b2b7dL,
0xe7fefe19L, 0xb5d7d762L, 0x4dababe6L, 0xec76769aL,
0x8fcaca45L, 0x1f82829dL, 0x89c9c940L, 0xfa7d7d87L,
0xeffafa15L, 0xb25959ebL, 0x8e4747c9L, 0xfbf0f00bL,
0x41adadecL, 0xb3d4d467L, 0x5fa2a2fdL, 0x45afafeaL,
0x239c9cbfL, 0x53a4a4f7L, 0xe4727296L, 0x9bc0c05bL,
0x75b7b7c2L, 0xe1fdfd1cL, 0x3d9393aeL, 0x4c26266aL,
0x6c36365aL, 0x7e3f3f41L, 0xf5f7f702L, 0x83cccc4fL,
0x6834345cL, 0x51a5a5f4L, 0xd1e5e534L, 0xf9f1f108L,
0xe2717193L, 0xabd8d873L, 0x62313153L, 0x2a15153fL,
0x0804040cL, 0x95c7c752L, 0x46232365L, 0x9dc3c35eL,
0x30181828L, 0x379696a1L, 0x0a05050fL, 0x2f9a9ab5L,
0x0e070709L, 0x24121236L, 0x1b80809bL, 0xdfe2e23dL,
0xcdebeb26L, 0x4e272769L, 0x7fb2b2cdL, 0xea75759fL,
0x1209091bL, 0x1d83839eL, 0x582c2c74L, 0x341a1a2eL,
0x361b1b2dL, 0xdc6e6eb2L, 0xb45a5aeeL, 0x5ba0a0fbL,
0xa45252f6L, 0x763b3b4dL, 0xb7d6d661L, 0x7db3b3ceL,
0x5229297bL, 0xdde3e33eL, 0x5e2f2f71L, 0x13848497L,
0xa65353f5L, 0xb9d1d168L, 0x00000000L, 0xc1eded2cL,
0x40202060L, 0xe3fcfc1fL, 0x79b1b1c8L, 0xb65b5bedL,
0xd46a6abeL, 0x8dcbcb46L, 0x67bebed9L, 0x7239394bL,
0x944a4adeL, 0x984c4cd4L, 0xb05858e8L, 0x85cfcf4aL,
0xbbd0d06bL, 0xc5efef2aL, 0x4faaaae5L, 0xedfbfb16L,
0x864343c5L, 0x9a4d4dd7L, 0x66333355L, 0x11858594L,
0x8a4545cfL, 0xe9f9f910L, 0x04020206L, 0xfe7f7f81L,
0xa05050f0L, 0x783c3c44L, 0x259f9fbaL, 0x4ba8a8e3L,
0xa25151f3L, 0x5da3a3feL, 0x804040c0L, 0x058f8f8aL,
0x3f9292adL, 0x219d9dbcL, 0x70383848L, 0xf1f5f504L,
0x63bcbcdfL, 0x77b6b6c1L, 0xafdada75L, 0x42212163L,
0x20101030L, 0xe5ffff1aL, 0xfdf3f30eL, 0xbfd2d26dL,
0x81cdcd4cL, 0x180c0c14L, 0x26131335L, 0xc3ecec2fL,
0xbe5f5fe1L, 0x359797a2L, 0x884444ccL, 0x2e171739L,
0x93c4c457L, 0x55a7a7f2L, 0xfc7e7e82L, 0x7a3d3d47L,
0xc86464acL, 0xba5d5de7L, 0x3219192bL, 0xe6737395L,
0xc06060a0L, 0x19818198L, 0x9e4f4fd1L, 0xa3dcdc7fL,
0x44222266L, 0x542a2a7eL, 0x3b9090abL, 0x0b888883L,
0x8c4646caL, 0xc7eeee29L, 0x6bb8b8d3L, 0x2814143cL,
0xa7dede79L, 0xbc5e5ee2L, 0x160b0b1dL, 0xaddbdb76L,
0xdbe0e03bL, 0x64323256L, 0x743a3a4eL, 0x140a0a1eL,
0x924949dbL, 0x0c06060aL, 0x4824246cL, 0xb85c5ce4L,
0x9fc2c25dL, 0xbdd3d36eL, 0x43acacefL, 0xc46262a6L,
0x399191a8L, 0x319595a4L, 0xd3e4e437L, 0xf279798bL,
0xd5e7e732L, 0x8bc8c843L, 0x6e373759L, 0xda6d6db7L,
0x018d8d8cL, 0xb1d5d564L, 0x9c4e4ed2L, 0x49a9a9e0L,
0xd86c6cb4L, 0xac5656faL, 0xf3f4f407L, 0xcfeaea25L,
0xca6565afL, 0xf47a7a8eL, 0x47aeaee9L, 0x10080818L,
0x6fbabad5L, 0xf0787888L, 0x4a25256fL, 0x5c2e2e72L,
0x381c1c24L, 0x57a6a6f1L, 0x73b4b4c7L, 0x97c6c651L,
0xcbe8e823L, 0xa1dddd7cL, 0xe874749cL, 0x3e1f1f21L,
0x964b4bddL, 0x61bdbddcL, 0x0d8b8b86L, 0x0f8a8a85L,
0xe0707090L, 0x7c3e3e42L, 0x71b5b5c4L, 0xcc6666aaL,
0x904848d8L, 0x06030305L, 0xf7f6f601L, 0x1c0e0e12L,
0xc26161a3L, 0x6a35355fL, 0xae5757f9L, 0x69b9b9d0L,
0x17868691L, 0x99c1c158L, 0x3a1d1d27L, 0x279e9eb9L,
```

```
0xd9e1e138L, 0xebf8f813L, 0x2b9898b3L, 0x22111133L,
0xd26969bbL, 0xa9d9d970L, 0x078e8e89L, 0x339494a7L,
0x2d9b9bb6L, 0x3c1e1e22L, 0x15878792L, 0xc9e9e920L,
0x87cece49L, 0xaa5555ffL, 0x50282878L, 0xa5dfdf7aL,
0x038c8c8fL, 0x59a1a1f8L, 0x09898980L, 0x1a0d0d17L,
0x65bfbfdaL, 0xd7e6e631L, 0x844242c6L, 0xd06868b8L,
0x824141c3L, 0x299999b0L, 0x5a2d2d77L, 0x1e0f0f11L,
0x7bb0b0cbL, 0xa85454fcL, 0x6dbbbbd6L, 0x2c16163aL};

private long[] Te1 = {
0xa5c66363l, 0x84f87c7cl, 0x99ee7777l, 0x8df67b7bl,
0x0dfff2f2l, 0xbdd66b6bl, 0xb1de6f6fl, 0x5491c5c5l,
0x50603030l, 0x03020101l, 0xa9ce6767l, 0x7d562b2bl,
0x19e7fefel, 0x62b5d7d7l, 0xe64dababl, 0x9aec7676l,
0x458fcacal, 0x9d1f8282l, 0x4089c9c9l, 0x87fa7d7dl,
0x15effafal, 0xebb25959l, 0xc98e4747l, 0x0bfbf0f0l,
0xec41adadl, 0x67b3d4d4l, 0xfd5fa2a2l, 0xea45afafl,
0xbf239c9cl, 0xf753a4a4l, 0x96e47272l, 0x5b9bc0c0l,
0xc275b7b7l, 0x1ce1fdfdl, 0xae3d9393l, 0x6a4c2626l,
0x5a6c3636l, 0x417e3f3fl, 0x02f5f7f7l, 0x4f83ccccl,
0x5c683434l, 0xf451a5a5l, 0x34d1e5e5l, 0x08f9f1f1l,
0x93e27171l, 0x73abd8d8l, 0x53623131l, 0x3f2a1515l,
0x0c080404l, 0x5295c7c7l, 0x65462323l, 0x5e9dc3c3l,
0x28301818l, 0xa1379696l, 0x0f0a0505l, 0xb52f9a9al,
0x090e0707l, 0x36241212l, 0x9b1b8080l, 0x3ddfe2e2l,
0x26cdebebl, 0x694e2727l, 0xcd7fb2b2l, 0x9fea7575l,
0x1b120909l, 0x9e1d8383l, 0x74582c2cl, 0x2e341a1al,
0x2d361b1bl, 0xb2dc6e6el, 0xeeb45a5al, 0xfb5ba0a0l,
0xf6a45252l, 0x4d763b3bl, 0x61b7d6d6l, 0xce7db3b3l,
0x7b522929l, 0x3edde3e3l, 0x715e2f2fl, 0x97138484l,
0xf5a65353l, 0x68b9d1d1l, 0x00000000l, 0x2cc1ededl,
0x60402020l, 0x1fe3fcfcl, 0xc879b1b1l, 0xedb65b5bl,
0xbed46a6al, 0x468dcbcbl, 0xd967bebel, 0x4b723939l,
0xde944a4al, 0xd4984c4cl, 0xe8b05858l, 0x4a85cfcfl,
0x6bbbd0d0l, 0x2ac5efefl, 0xe54faaaal, 0x16edfbfbl,
0xc5864343l, 0xd79a4d4dl, 0x55663333l, 0x94118585l,
0xcf8a4545l, 0x10e9f9f9l, 0x06040202l, 0x81fe7f7fl,
0xf0a05050l, 0x44783c3cl, 0xba259f9fl, 0xe34ba8a8l,
0xf3a25151l, 0xfe5da3a3l, 0xc0804040l, 0x8a058f8fl,
0xad3f9292l, 0xbc219d9dl, 0x48703838l, 0x04f1f5f5l,
0xdf63bcbcl, 0xc177b6b6l, 0x75afdadal, 0x63422121l,
0x30201010l, 0x1ae5ffffl, 0x0efdf3f3l, 0x6dbfd2d2l,
0x4c81cdcdl, 0x14180c0cl, 0x35261313l, 0x2fc3ececl,
0xe1be5f5fl, 0xa2359797l, 0xcc884444l, 0x392e1717l,
0x5793c4c4l, 0xf255a7a7l, 0x82fc7e7el, 0x477a3d3dl,
0xacc86464l, 0xe7ba5d5dl, 0x2b321919l, 0x95e67373l,
0xa0c06060l, 0x98198181l, 0xd19e4f4fl, 0x7fa3dcdcl,
0x66442222l, 0x7e542a2al, 0xab3b9090l, 0x830b8888l,
0xca8c4646l, 0x29c7eeeel, 0xd36bb8b8l, 0x3c281414l,
0x79a7dedel, 0xe2bc5e5el, 0x1d160b0bl, 0x76addbdbl,
0x3bdbe0e0l, 0x56643232l, 0x4e743a3al, 0x1e140a0al,
0xdb924949l, 0x0a0c0606l, 0x6c482424l, 0xe4b85c5cl,
0x5d9fc2c2l, 0x6ebdd3d3l, 0xef43acacl, 0xa6c46262l,
0xa8399191l, 0xa4319595l, 0x37d3e4e4l, 0x8bf27979l,
0x32d5e7e7l, 0x438bc8c8l, 0x596e3737l, 0xb7da6d6dl,
0x8c018d8dl, 0x64b1d5d5l, 0xd29c4e4el, 0xe049a9a9l,
0xb4d86c6cl, 0xfaac5656l, 0x07f3f4f4l, 0x25cfeaeal,
```

```
0xafca6565l, 0x8ef47a7al, 0xe947aeael, 0x18100808l,
0xd56fbabal, 0x88f07878l, 0x6f4a2525l, 0x725c2e2el,
0x24381c1cl, 0xf157a6a6l, 0xc773b4b4l, 0x5197c6c6l,
0x23cbe8e8l, 0x7ca1ddddl, 0x9ce87474l, 0x213e1f1fl,
0xdd964b4bl, 0xdc61bdbdl, 0x860d8b8bl, 0x850f8a8al,
0x90e07070l, 0x427c3e3el, 0xc471b5b5l, 0xaacc6666l,
0xd8904848l, 0x05060303l, 0x01f7f6f6l, 0x121c0e0el,
0xa3c26161l, 0x5f6a3535l, 0xf9ae5757l, 0xd069b9b9l,
0x91178686l, 0x5899c1c1l, 0x273a1d1dl, 0xb9279e9el,
0x38d9e1e1l, 0x13ebf8f8l, 0xb32b9898l, 0x33221111l,
0xbbd26969l, 0x70a9d9d9l, 0x89078e8el, 0xa7339494l,
0xb62d9b9bl, 0x223c1e1el, 0x92158787l, 0x20c9e9e9l,
0x4987cecel, 0xffaa5555l, 0x78502828l, 0x7aa5dfdfl,
0x8f038c8cl, 0xf859a1a1l, 0x80098989l, 0x171a0d0dl,
0xda65bfbfl, 0x31d7e6e6l, 0xc6844242l, 0xb8d06868l,
0xc3824141l, 0xb0299999l, 0x775a2d2dl, 0x111e0f0fl,
0xcb7bb0b0l, 0xfca85454l, 0xd66dbbbbl, 0x3a2c1616l};

private long[] Te2 = {
0x63a5c663l, 0x7c84f87cl, 0x7799ee77l, 0x7b8df67bl,
0xf20dfff2l, 0x6bbdd66bl, 0x6fb1de6fl, 0xc55491c5l,
0x30506030l, 0x01030201l, 0x67a9ce67l, 0x2b7d562bl,
0xfe19e7fel, 0xd762b5d7l, 0xabe64dabl, 0x769aec76l,
0xca458fcal, 0x829d1f82l, 0xc94089c9l, 0x7d87fa7dl,
0xfa15effal, 0x59ebb259l, 0x47c98e47l, 0xf00bfbf0l,
0xadec41adl, 0xd467b3d4l, 0xa2fd5fa2l, 0xafea45afl,
0x9cbf239cl, 0xa4f753a4l, 0x7296e472l, 0xc05b9bc0l,
0xb7c275b7l, 0xfd1ce1fdl, 0x93ae3d93l, 0x266a4c26l,
0x365a6c36l, 0x3f417e3fl, 0xf702f5f7l, 0xcc4f83ccl,
0x345c6834l, 0xa5f451a5l, 0xe534d1e5l, 0xf108f9f1l,
0x7193e271l, 0xd873abd8l, 0x31536231l, 0x153f2a15l,
0x040c0804l, 0xc75295c7l, 0x23654623l, 0xc35e9dc3l,
0x18283018l, 0x96a13796l, 0x050f0a05l, 0x9ab52f9al,
0x07090e07l, 0x12362412l, 0x809b1b80l, 0xe23ddfe2l,
0xeb26cdebl, 0x27694e27l, 0xb2cd7fb2l, 0x759fea75l,
0x091b1209l, 0x839e1d83l, 0x2c74582cl, 0x1a2e341al,
0x1b2d361bl, 0x6eb2dc6el, 0x5aeeb45al, 0xa0fb5ba0l,
0x52f6a452l, 0x3b4d763bl, 0xd661b7d6l, 0xb3ce7db3l,
0x297b5229l, 0xe33edde3l, 0x2f715e2fl, 0x84971384l,
0x53f5a653l, 0xd168b9d1l, 0x00000000l, 0xed2cc1edl,
0x20604020l, 0xfc1fe3fcl, 0xb1c879b1l, 0x5bedb65bl,
0x6abed46al, 0xcb468dcbl, 0xbed967bel, 0x394b7239l,
0x4ade944al, 0x4cd4984cl, 0x58e8b058l, 0xcf4a85cfl,
0xd06bbbd0l, 0xef2ac5efl, 0xaae54faal, 0xfb16edfbl,
0x43c58643l, 0x4dd79a4dl, 0x33556633l, 0x85941185l,
0x45cf8a45l, 0xf910e9f9l, 0x02060402l, 0x7f81fe7fl,
0x50f0a050l, 0x3c44783cl, 0x9fba259fl, 0xa8e34ba8l,
0x51f3a251l, 0xa3fe5da3l, 0x40c08040l, 0x8f8a058fl,
0x92ad3f92l, 0x9dbc219dl, 0x38487038l, 0xf504f1f5l,
0xbcdf63bcl, 0xb6c177b6l, 0xda75afdal, 0x21634221l,
0x10302010l, 0xff1ae5ffl, 0xf30efdf3l, 0xd26dbfd2l,
0xcd4c81cdl, 0x0c14180cl, 0x13352613l, 0xec2fc3ecl,
0x5fe1be5fl, 0x97a23597l, 0x44cc8844l, 0x17392e17l,
0xc45793c4l, 0xa7f255a7l, 0x7e82fc7el, 0x3d477a3dl,
0x64acc864l, 0x5de7ba5dl, 0x192b3219l, 0x7395e673l,
0x60a0c060l, 0x81981981l, 0x4fd19e4fl, 0xdc7fa3dcl,
0x22664422l, 0x2a7e542al, 0x90ab3b90l, 0x88830b88l,
```

```
0x46ca8c46l, 0xee29c7eel, 0xb8d36bb8l, 0x143c2814l,
0xde79a7del, 0x5ee2bc5el, 0x0b1d160bl, 0xdb76addbl,
0xe03bdbe0l, 0x32566432l, 0x3a4e743al, 0x0a1e140al,
0x49db9249l, 0x060a0c06l, 0x246c4824l, 0x5ce4b85cl,
0xc25d9fc2l, 0xd36ebdd3l, 0xacef43acl, 0x62a6c462l,
0x91a83991l, 0x95a43195l, 0xe437d3e4l, 0x798bf279l,
0xe732d5e7l, 0xc8438bc8l, 0x37596e37l, 0x6db7da6dl,
0x8d8c018dl, 0xd564b1d5l, 0x4ed29c4el, 0xa9e049a9l,
0x6cb4d86cl, 0x56faac56l, 0xf407f3f4l, 0xea25cfeal,
0x65afca65l, 0x7a8ef47al, 0xaee947ael, 0x08181008l,
0xbad56fbal, 0x7888f078l, 0x256f4a25l, 0x2e725c2el,
0x1c24381cl, 0xa6f157a6l, 0xb4c773b4l, 0xc65197c6l,
0xe823cbe8l, 0xdd7ca1ddl, 0x749ce874l, 0x1f213e1fl,
0x4bdd964bl, 0xbddc61bdl, 0x8b860d8bl, 0x8a850f8al,
0x7090e070l, 0x3e427c3el, 0xb5c471b5l, 0x66aacc66l,
0x48d89048l, 0x03050603l, 0xf601f7f6l, 0x0e121c0el,
0x61a3c261l, 0x355f6a35l, 0x57f9ae57l, 0xb9d069b9l,
0x86911786l, 0xc15899c1l, 0x1d273a1dl, 0x9eb9279el,
0xe138d9e1l, 0xf813ebf8l, 0x98b32b98l, 0x11332211l,
0x69bbd269l, 0xd970a9d9l, 0x8e89078el, 0x94a73394l,
0x9bb62d9bl, 0x1e223c1el, 0x87921587l, 0xe920c9e9l,
0xce4987cel, 0x55ffaa55l, 0x28785028l, 0xdf7aa5dfl,
0x8c8f038cl, 0xa1f859a1l, 0x89800989l, 0x0d171a0dl,
0xbfda65bfl, 0xe631d7e6l, 0x42c68442l, 0x68b8d068l,
0x41c38241l, 0x99b02999l, 0x2d775a2dl, 0x0f111e0fl,
0xb0cb7bb0l, 0x54fca854l, 0xbbd66dbbl, 0x163a2c16l};

private long[] Te3 = {
0x6363a5c6l, 0x7c7c84f8l, 0x777799eel, 0x7b7b8df6l,
0xf2f20dffl, 0x6b6bbdd6l, 0x6f6fb1del, 0xc5c55491l,
0x30305060l, 0x01010302l, 0x6767a9cel, 0x2b2b7d56l,
0xfefe19e7l, 0xd7d762b5l, 0xababe64dl, 0x76769aecl,
0xcaca458fl, 0x82829d1fl, 0xc9c94089l, 0x7d7d87fal,
0xfafa15efl, 0x5959ebb2l, 0x4747c98el, 0xf0f00bfbl,
0xadadec41l, 0xd4d467b3l, 0xa2a2fd5fl, 0xafafea45l,
0x9c9cbf23l, 0xa4a4f753l, 0x727296e4l, 0xc0c05b9bl,
0xb7b7c275l, 0xfdfd1ce1l, 0x9393ae3dl, 0x26266a4cl,
0x36365a6cl, 0x3f3f417el, 0xf7f702f5l, 0xcccc4f83l,
0x34345c68l, 0xa5a5f451l, 0xe5e534d1l, 0xf1f108f9l,
0x717193e2l, 0xd8d873abl, 0x31315362l, 0x15153f2al,
0x04040c08l, 0xc7c75295l, 0x23236546l, 0xc3c35e9dl,
0x18182830l, 0x9696a137l, 0x05050f0al, 0x9a9ab52fl,
0x0707090el, 0x12123624l, 0x80809b1bl, 0xe2e23ddfl,
0xebeb26cdl, 0x2727694el, 0xb2b2cd7fl, 0x75759feal,
0x09091b12l, 0x83839e1dl, 0x2c2c7458l, 0x1a1a2e34l,
0x1b1b2d36l, 0x6e6eb2dcl, 0x5a5aeeb4l, 0xa0a0fb5bl,
0x5252f6a4l, 0x3b3b4d76l, 0xd6d661b7l, 0xb3b3ce7dl,
0x29297b52l, 0xe3e33eddl, 0x2f2f715el, 0x84849713l,
0x5353f5a6l, 0xd1d168b9l, 0x00000000l, 0xeded2cc1l,
0x20206040l, 0xfcfc1fe3l, 0xb1b1c879l, 0x5b5bedb6l,
0x6a6abed4l, 0xcbcb468dl, 0xbebed967l, 0x39394b72l,
0x4a4ade94l, 0x4c4cd498l, 0x5858e8b0l, 0xcfcf4a85l,
0xd0d06bbbl, 0xefef2ac5l, 0xaaaae54fl, 0xfbfb16edl,
0x4343c586l, 0x4d4dd79al, 0x33335566l, 0x85859411l,
0x4545cf8al, 0xf9f910e9l, 0x02020604l, 0x7f7f81fel,
0x5050f0a0l, 0x3c3c4478l, 0x9f9fba25l, 0xa8a8e34bl,
0x5151f3a2l, 0xa3a3fe5dl, 0x4040c080l, 0x8f8f8a05l,
```

```
0x9292ad3fl, 0x9d9dbc21l, 0x38384870l, 0xf5f504f1l,
0xbcbcdf63l, 0xb6b6c177l, 0xdada75afl, 0x21216342l,
0x10103020l, 0xffff1ae5l, 0xf3f30efdl, 0xd2d26dbfl,
0xcdcd4c81l, 0x0c0c1418l, 0x13133526l, 0xecec2fc3l,
0x5f5fe1bel, 0x9797a235l, 0x4444cc88l, 0x1717392el,
0xc4c45793l, 0xa7a7f255l, 0x7e7e82fcl, 0x3d3d477al,
0x6464acc8l, 0x5d5de7bal, 0x19192b32l, 0x737395e6l,
0x6060a0c0l, 0x81819819l, 0x4f4fd19el, 0xdcdc7fa3l,
0x22226644l, 0x2a2a7e54l, 0x9090ab3bl, 0x8888830bl,
0x4646ca8cl, 0xeeee29c7l, 0xb8b8d36bl, 0x14143c28l,
0xdede79a7l, 0x5e5ee2bcl, 0x0b0b1d16l, 0xdbdb76adl,
0xe0e03bdbl, 0x32325664l, 0x3a3a4e74l, 0x0a0a1e14l,
0x4949db92l, 0x06060a0cl, 0x24246c48l, 0x5c5ce4b8l,
0xc2c25d9fl, 0xd3d36ebdl, 0xacacef43l, 0x6262a6c4l,
0x9191a839l, 0x9595a431l, 0xe4e437d3l, 0x79798bf2l,
0xe7e732d5l, 0xc8c8438bl, 0x3737596el, 0x6d6db7dal,
0x8d8d8c01l, 0xd5d564b1l, 0x4e4ed29cl, 0xa9a9e049l,
0x6c6cb4d8l, 0x5656faacl, 0xf4f407f3l, 0xeaea25cfl,
0x6565afcal, 0x7a7a8ef4l, 0xaeaee947l, 0x08081810l,
0xbabad56fl, 0x787888f0l, 0x25256f4al, 0x2e2e725cl,
0x1c1c2438l, 0xa6a6f157l, 0xb4b4c773l, 0xc6c65197l,
0xe8e823cbl, 0xdddd7ca1l, 0x74749ce8l, 0x1f1f213el,
0x4b4bdd96l, 0xbdbddc61l, 0x8b8b860dl, 0x8a8a850fl,
0x707090e0l, 0x3e3e427cl, 0xb5b5c471l, 0x6666aaccl,
0x4848d890l, 0x03030506l, 0xf6f601f7l, 0x0e0e121cl,
0x6161a3c2l, 0x35355f6al, 0x5757f9ael, 0xb9b9d069l,
0x86869117l, 0xc1c15899l, 0x1d1d273al, 0x9e9eb927l,
0xe1e138d9l, 0xf8f813ebl, 0x9898b32bl, 0x11113322l,
0x6969bbd2l, 0xd9d970a9l, 0x8e8e8907l, 0x9494a733l,
0x9b9bb62dl, 0x1e1e223cl, 0x87879215l, 0xe9e920c9l,
0xcece4987l, 0x5555ffaal, 0x28287850l, 0xdfdf7aa5l,
0x8c8c8f03l, 0xa1a1f859l, 0x89898009l, 0x0d0d171al,
0xbfbfda65l, 0xe6e631d7l, 0x4242c684l, 0x6868b8d0l,
0x4141c382l, 0x9999b029l, 0x2d2d775al, 0x0f0f111el,
0xb0b0cb7bl, 0x5454fca8l, 0xbbbbd66dl, 0x16163a2cl};

private long[] Te4 = {
0x63636363l, 0x7c7c7c7cl, 0x77777777l, 0x7b7b7b7bl,
0xf2f2f2f2l, 0x6b6b6b6bl, 0x6f6f6f6fl, 0xc5c5c5c5l,
0x30303030l, 0x01010101l, 0x67676767l, 0x2b2b2b2bl,
0xfefefefel, 0xd7d7d7d7l, 0xabababablabl, 0x76767676l,
0xcacacacal, 0x82828282l, 0xc9c9c9c9l, 0x7d7d7d7dl,
0xfafafafal, 0x59595959l, 0x47474747l, 0xf0f0f0f0l,
0xadadadadl, 0xd4d4d4d4l, 0xa2a2a2a2l, 0xafafafafl,
0x9c9c9c9cl, 0xa4a4a4a4l, 0x72727272l, 0xc0c0c0c0l,
0xb7b7b7b7l, 0xfdfdfdfdl, 0x93939393l, 0x26262626l,
0x36363636l, 0x3f3f3f3fl, 0xf7f7f7f7l, 0xccccccccl,
0x34343434l, 0xa5a5a5a5l, 0xe5e5e5e5l, 0xf1f1f1f1l,
0x71717171l, 0xd8d8d8d8l, 0x31313131l, 0x15151515l,
0x04040404l, 0xc7c7c7c7l, 0x23232323l, 0xc3c3c3c3l,
0x18181818l, 0x96969696l, 0x05050505l, 0x9a9a9a9al,
0x07070707l, 0x12121212l, 0x80808080l, 0xe2e2e2e2l,
0xebebebebl, 0x27272727l, 0xb2b2b2b2l, 0x75757575l,
0x09090909l, 0x83838383l, 0x2c2c2c2cl, 0x1a1a1a1al,
0x1b1b1b1bl, 0x6e6e6e6el, 0x5a5a5a5al, 0xa0a0a0a0l,
0x52525252l, 0x3b3b3b3bl, 0xd6d6d6d6l, 0xb3b3b3b3l,
0x29292929l, 0xe3e3e3e3l, 0x2f2f2f2fl, 0x84848484l,
```

```
0x53535353l, 0xd1d1d1d1l, 0x00000000l, 0xededededl,
0x20202020l, 0xfcfcfcfcl, 0xb1b1b1b1l, 0x5b5b5b5bl,
0x6a6a6a6al, 0xcbcbcbcbl, 0xbebebebel, 0x39393939l,
0x4a4a4a4al, 0x4c4c4c4cl, 0x58585858l, 0xcfcfcfcfl,
0xd0d0d0d0l, 0xefefefefl, 0xaaaaaaaal, 0xfbfbfbfbl,
0x43434343l, 0x4d4d4d4dl, 0x33333333l, 0x85858585l,
0x45454545l, 0xf9f9f9f9l, 0x02020202l, 0x7f7f7f7fl,
0x50505050l, 0x3c3c3c3cl, 0x9f9f9f9fl, 0xa8a8a8a8l,
0x51515151l, 0xa3a3a3a3l, 0x40404040l, 0x8f8f8f8fl,
0x92929292l, 0x9d9d9d9dl, 0x38383838l, 0xf5f5f5f5l,
0xbcbcbcbcl, 0xb6b6b6b6l, 0xdadadadal, 0x21212121l,
0x10101010l, 0xffffffffl, 0xf3f3f3f3l, 0xd2d2d2d2l,
0xcdcdcdcdl, 0x0c0c0c0cl, 0x13131313l, 0xececececl,
0x5f5f5f5fl, 0x97979797l, 0x44444444l, 0x17171717l,
0xc4c4c4c4l, 0xa7a7a7a7l, 0x7e7e7e7el, 0x3d3d3d3dl,
0x64646464l, 0x5d5d5d5dl, 0x19191919l, 0x73737373l,
0x60606060l, 0x81818181l, 0x4f4f4f4fl, 0xdcdcdcdcl,
0x22222222l, 0x2a2a2a2al, 0x90909090l, 0x88888888l,
0x46464646l, 0xeeeeeeeel, 0xb8b8b8b8l, 0x14141414l,
0xdedededel, 0x5e5e5e5el, 0x0b0b0b0bl, 0xdbdbdbdbl,
0xe0e0e0e0l, 0x32323232l, 0x3a3a3a3al, 0x0a0a0a0al,
0x49494949l, 0x06060606l, 0x24242424l, 0x5c5c5c5cl,
0xc2c2c2c2l, 0xd3d3d3d3l, 0xacacacacl, 0x62626262l,
0x91919191l, 0x95959595l, 0xe4e4e4e4l, 0x79797979l,
0xe7e7e7e7l, 0xc8c8c8c8l, 0x37373737l, 0x6d6d6d6dl,
0x8d8d8d8dl, 0xd5d5d5d5l, 0x4e4e4e4el, 0xa9a9a9a9l,
0x6c6c6c6cl, 0x56565656l, 0xf4f4f4f4l, 0xeaeaeaeal,
0x65656565l, 0x7a7a7a7al, 0xaeaeaeael, 0x08080808l,
0xbababaal, 0x78787878l, 0x25252525l, 0x2e2e2e2el,
0x1c1c1c1cl, 0xa6a6a6a6l, 0xb4b4b4b4l, 0xc6c6c6c6l,
0xe8e8e8e8l, 0xdddddddl, 0x74747474l, 0x1f1f1f1fl,
0x4b4b4b4bl, 0xbdbdbdbdl, 0x8b8b8b8bl, 0x8a8a8a8al,
0x70707070l, 0x3e3e3e3el, 0xb5b5b5b5l, 0x66666666l,
0x48484848l, 0x03030303l, 0xf6f6f6f6l, 0x0e0e0e0el,
0x61616161l, 0x35353535l, 0x57575757l, 0xb9b9b9b9l,
0x86868686l, 0xc1c1c1c1l, 0x1d1d1d1dl, 0x9e9e9e9el,
0xe1e1e1e1l, 0xf8f8f8f8l, 0x98989898l, 0x11111111l,
0x69696969l, 0xd9d9d9d9l, 0x8e8e8e8el, 0x94949494l,
0x9b9b9b9bl, 0x1e1e1e1el, 0x87878787l, 0xe9e9e9e9l,
0xcecececel, 0x55555555l, 0x28282828l, 0xdfdfdfdfl,
0x8c8c8c8cl, 0xa1a1a1a1l, 0x89898989l, 0x0d0d0d0dl,
0xbfbfbfbfl, 0xe6e6e6e6l, 0x42424242l, 0x68686868l,
0x41414141l, 0x99999999l, 0x2d2d2d2dl, 0x0f0f0f0fl,
0xb0b0b0b0l, 0x54545454l, 0xbbbbbbbbl, 0x16161616l};

public long Te0(byte b)
{
    return Te0[b & 0xff];
}

public long Te1(byte b)
{
    return Te1[b & 0xff];
}

public long Te2(byte b)
{
```

```
    return Te2[b & 0xff];
}

public long Te3(byte b)
{
    return Te3[b & 0xff];
}

public long Te4(byte b)
{
    return Te4[b & 0xff];
}

public long MTe0(byte b)
{
    return MTe0[b & 0xff];
}

public long MTe1(byte b)
{
    return MTe1[b & 0xff];
}

public long MTe2(byte b)
{
    return MTe2[b & 0xff];
}

public long MTe3(byte b)
{
    return MTe3[b & 0xff];
}

public long MTe4(byte b)
{
    return MTe4[b & 0xff];
}

public long MOpt(byte b)
{
    return MOpt[b & 0xff];
}

// Routines to access table entries
public byte MSBox(byte b)
{
    return MS[b & 0xff];
}

// Routines to access table entries
public byte M(byte b)
{
    return M[b & 0xff];
}

public int maskOpt()
{
```

```
    return maskOpt;
}

public int maskNorm()
{
    return maskNorm;
}

// Routines to access table entries
public byte SBox(byte b)
{
    return S[b & 0xff];
}

public byte invSBox(byte b)
{
    return invS[b & 0xff];
}

public byte Rcon(int i)
{
    return powX[i-1];
}

// FFMulFast: fast multiply using table lookup
public byte FFMulFast(byte a, byte b)
{
    int t = 0;;
    if (a == 0 || b == 0) return 0;
    t = (L[(a & 0xff)] & 0xff) + (L[(b & 0xff)] & 0xff);
    if (t > 255) t = t - 255;
    return E[(t & 0xff)];
}

// FFMul: slow multiply, using shifting
public byte FFMul(byte a, byte b)
{
    byte aa = a, bb = b, r = 0, t;
    while (aa != 0)
    {
        if ((aa & 1) != 0)
            r = (byte)(r ^ bb);
        t = (byte)(bb & 0x80);
        bb = (byte)(bb << 1);
        if (t != 0)
            bb = (byte)(bb ^ 0x1b);
        aa = (byte)((aa & 0xff) >> 1);
    }
    return r;
}

// loadE: create and load the E table
private void loadE()
{
    byte x = (byte)0x01;
    int index = 0;
    E[index++] = (byte)0x01;
```

```
    for (int i = 0; i < 255; i++)
    {
        byte y = FFMul(x, (byte)0x03);
        E[index++] = y;
        x = y;
    }
}

// loadL: load the L table using the E table
private void loadL()
{ // careful: had 254 below several places
    int index;
    for (int i = 0; i < 255; i++)
    {
        L[E[i] & 0xff] = (byte)i;
    }
}

// loadS: load in the table S
private void loadS()
{
    int index;
    for (int i = 0; i < 256; i++)
    S[i] = (byte)(subBytes((byte)(i & 0xff)) & 0xff);
}

// loadInv: load in the table inv
private void loadInv()
{
    int index;
    for (int i = 0; i < 256; i++)
        inv[i] = (byte)(FFInv((byte)(i & 0xff)) & 0xff);
}

// loadInvS: load the invS table using the S table
private void loadInvS()
{
    int index;
    for (int i = 0; i < 256; i++)
    {
        invS[S[i] & 0xff] = (byte)i;
    }
}

// loadPowX: load the powX table using multiplication
private void loadPowX()
{
    int index;
    byte x = (byte)0x02;
    byte xp = x;
    powX[0] = 1; powX[1] = x;
    for (int i = 2; i < 15; i++)
    {
        xp = FFMul(xp, x);
        powX[i] = xp;
    }
}
```

```
// FFInv: the multiplicative inverse of a byte value
public byte FFInv(byte b)
{
    byte e = L[b & 0xff];
    return E[0xff - (e & 0xff)];
}

// ithBIt: return the ith bit of a byte
public int ithBit(byte b, int i)
{
    int m[] = {0x01, 0x02, 0x04, 0x08, 0x10, 0x20, 0x40, 0x80};
    return (b & m[i]) >> i;
}

// subBytes: the subBytes function
public int subBytes(byte b)
{
    //byte inB = b;
    int res = 0;
    if (b != 0) // if b == 0, leave it alone
        b = (byte)(FFInv(b) & 0xff);
    byte c = (byte)0x63;
    for (int i = 0; i < 8; i++)
    {
        int temp = 0;
        temp = ithBit(b, i) ^ ithBit(b, (i+4)%8) ^ ithBit(b, (i+5)%8) ^
        ithBit(b, (i+6)%8) ^ ithBit(b, (i+7)%8) ^ ithBit(c, i);
        res = res | (temp << i);
    }
    return res;
}

private void genMTablesOrigAES() // for original S Box
{
    Random randGen = new Random(System.currentTimeMillis());
    maskNorm = (byte)(Math.abs(randGen.nextInt()) & 0x000000ff);
    byte tmpByte;

    System.out.println("mask1SBox="+Integer.toHexString(maskNorm));

    for (int i = 0; i < 256; i++)
    {
        tmpByte = (byte) (randGen.nextInt() & 0x000000ff);
        MS[i] = (byte)(S[i] ^ tmpByte);
        M[i] = (byte)(tmpByte ^ maskNorm);
    }
}


private void genMTablesOptAES() // for optimized S Boxes
{
    Random randGen = new Random(System.currentTimeMillis());
    maskOpt = Math.abs(randGen.nextInt());
    System.out.println("mask="+Integer.toHexString(maskOpt));
    int tmp;
```

```
        for (int i = 0; i < 256; i++)
        {
            tmp = randGen.nextInt();
            MTe0[i] = Te0[i] ^ tmp;
            MTe1[i] = Te1[i] ^ tmp;
            MTe2[i] = Te2[i] ^ tmp;
            MTe3[i] = Te3[i] ^ tmp;
            MTe4[i] = Te4[i] ^ tmp;
            MOpt[i] = tmp ^ maskOpt; // use M0 for the case of only 1 M table
        }
    }
}

public class Copy
{
    private static final int Nb = 4;

    // copy: copy in to state
    public static void copy(byte[][] state, byte[] in)
    {
        int inLoc = 0;
        for (int c = 0; c < Nb; c++)
            for (int r = 0; r < 4; r++)
                state[r][c] = in[inLoc++];
    }

    // copy: copy state to out
    public static void copy(byte[] out, byte[][] state)
    {
        int outLoc = 0;
        for (int c = 0; c < Nb; c++)
            for (int r = 0; r < 4; r++)
                out[outLoc++] = state[r][c];
    }
}
```