Electromagnetic analysis of AES-256 on Xilinx Artix-7

Oskar Westman
`elt13owe@student.lu.se`

Department of Electrical and Information Technology
Lund University

Supervisor: Martin Hell (LTH) and Lars Lindqvist (Advenica)

Examiner: Thomas Johansson

Tuesday 19th June, 2018

© 2018
Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

In this project, an electromagnetic side-channel attack has been made by exploiting the information leakage from a field programmable gate array (FPGA) and an implemented advanced encryption standard with a 256-bit key (AES-256). The FPGA-board was a Nexys-4 from Digilent with Artix-7 FPGA. The attack was partially successful. A few subkeys were successfully extracted from AES-256 with only 2000-3000 electromagnetic (EM) traces. The rest of the key guesses were ranked accordingly and presented in a chart. Three different data acquisitions were made on AES-256, and no average values were taken. Most of the previous work used an average value of 10-100 EM traces per plaintext input. In this thesis, only one plaintext per EM trace was used. The purpose of this was to simulate a real-world scenario where an attacker has access to the cryptographic device for approximately one hour.

The experiments also included an electromagnetic side-channel attack on an isolated hardware area in the AES algorithm by designing only the initial round and the SubBytes operation using single 8-bit data blocks. The purpose of this attack was to make the analysis less complex and more adapted to the simulation model.

Due to the parallelism in the FPGA, there was a low correlation between the key guesses and the correct key. The low correlation was expected but created obstacles when collecting data for key extraction.

There was also interference from the power supply. Every time someone plugged in, e.g., a cell phone charger or a laptop charger in the neighboring rooms it made the data acquisition corrupt. The random interference made longer test runs harder to conduct. The experiment needed constant supervision to detect if an interference occurred.

For future work, the side-channel attack needs more data points per EM trace, more EM traces, faster oscilloscope (or data acquisition unit), low-pass filter and an amplifier with a wider bandwidth.

Acknowledgements

I would like to thank my university supervisor Martin Hell for his support and patience regarding the number of questions I have thrown upon him. I would like to thank Lars Lindqvist at Advenica, who gave good support when we were discussing different approaches to conduct the experiments.

I would also like to thank Göran Jönsson for his support regarding the equipment and Christian Antfolk for helping me when I got stuck in LabView-programming.

Popular Science Summary

Every time a computer makes computations in its central processing unit (CPU) or dedicated hardware support, different currents start flowing in the circuits. When an electric current is moving, it generates a magnetic field. There is also an electric field coming from the differences in electrical potential between wires.

Cryptographic devices protect sensitive data from unauthorized personnel, making the information unreadable for everyone who does not have the secret key. Encryption algorithms often rely on advanced mathematics, to create a protection for the sensitive data. The only threats to a strong cryptographic algorithm are the computing capabilities of quantum computers and the risk of side-channel attacks.

In the year 1996, there was research published where the scientist discovered a way of analyzing the power consumption of the cryptographic device and with that information extract the secret key used for encrypting the data. This type of power analysis was given the name side-channel analysis (SCA). Later on, there were several other side-channel analysis where the attacker exploited the temperature, noise and electromagnetic emissions from the electronic device to extract the key. There exist historically older examples of people utilizing the side-channels to retrieve information, but not on modern computers.

In this thesis, an electromagnetic side-channel analysis (EM-SCA) was made on an FPGA. FPGA is a chip with unique abilities to reconfigure its hardware depending on the bitstream uploaded to it. FPGAs are becoming more and more integrated into the cybersecurity applications to accelerate different mathematical operations in the cryptographic device. Several companies design and deliver different cryptographic intellectual properties (IPs). Crypto-IPs is an architecture for FPGAs and application specific integrated circuits (ASICs). The electromagnetic footprint from an FPGA depends totally on the design of the IP. If an attacker gets physically close to the cryptographic device, one will have a good opportunity to record the electromagnetic emissions and the output/input from/to the device. With only these two parameters an attack can extract the secret key and decrypt information that was unreadable before.

The IP that was attacked is an implementation of advanced encryption standard with a 256-bit key (AES-256). AES-256 is considered post-quantum computer secure, meaning that a quantum computer will not be able to brute force the encryption within a reasonable time. A side-channel attack is much easier and cheaper to conduct and can bypass strong mathematical encryption algorithms, but the attacker needs to be physically close. The experiments made in the thesis were partially successful; the attacks were able to extract subkeys from a data acquisition below 10000 electromagnetic traces.

Contents

List of Figures

List of Tables

Abbreviations

AES	-	Advanced Encryption Standard
CEMA	-	Correlation Electromagnetic Analysis
CMOS	-	Complementary Metal Oxide Semiconductor
CPU	-	Central Processing Unit
DEMA	-	Differential Electromagnetic Analysis
DPA	-	Differential Power Analysis
EM	-	Electromagnetic
EMA	-	Electromagnetic Analysis
EM-SCA	-	Electromagnetic Side-Channel Analysis
I/O	-	Input/Output
FSM	-	Finite State Machine
LFSR	-	Linear Feedback Shift Register
NMOS	-	N-channel Metal Oxide Semiconductor
PCI-e	-	Peripheral Component Interconnect Express
PGE	-	Partial Guessing Entropy
PMOS	-	P-channel Metal Oxide Semiconductor
RTL	-	Register Transfer Level
s-box	-	Substitution box
FPGA	-	Field Programmable Gate Array
SCA	-	Side-Channel Analysis
SEMA	-	Simple Electromagnetic Analysis
SNR	-	Signal-to-Noise-Ratio
SoC	-	System-on-Chip
SPA	-	Simple Power Analysis
USB	-	Universal Serial Bus
XOR	-	eXclusive OR
VHSIC	-	Very High Speed Integrated Circuit
VHDL	-	VHSIC Hardware Description Language

Chapter 1

Introduction

1.1 Background

With increasing use of advanced cryptographic algorithms, when protecting sensitive data, the computational strength to brute force the password has increased a lot. The time effort needed is so large that it makes the information useless, and outdated, when it is decrypted. The information is not valid anymore. Today there are two generic threats to cryptographic systems; quantum computers and side-channel analysis. In this project, the focus will be on the more inexpensive option; side-channel analysis.

Cryptographic operations such as encryption/decryption, message-authentication, and digital signatures rely on secret keys that must be kept securely within a device and protected from unauthorized personnel. With the use of different methods, an attacker can exploit the side-channels from a security device to extract the secret key. The methods are passive or active with an invasive or non-invasive approach.

A passive non-invasive attack is often called **side-channel analysis** [1]. It is a way of analyzing metadata of an encryption scheme, to obtain the actual secret key. A side-channel analysis is often limited to different physical aspects; analyzing electromagnetic radiation, power consumption, time or audio [2]. With the use of side-channel analysis, the attacker can bypass the mathematical protection, and still acquire the secret key without the help of computational strong computers.

Advenica is a company which has specialized in cybersecurity with hardware applications. National armed forces and other authorities use their products. EU, NATO and Swedish armed forces have certified Advenicas products to the EU-standard SECRET UE/EU SECRET, TOP SECRET for Swedish armed forces and NATO SECRET [3].

Security systems use mathematical algorithms to provide confidentiality, integrity, and authentication. According to [1, p.3] the definition for a cryptographic device is: *Cryptographic devices are electronic devices that implement cryptographic algorithms and that store cryptographic keys.* The algorithm is implemented in either hardware or software and accepts two inputs; the message and the cryptographic key, see Figure 1.1.

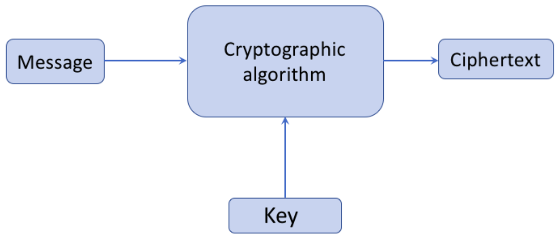


Figure 1.1: Main concept of cryptographic algorithm.

The encrypted message is called ciphertext. Nowadays the encrypting algorithm and the ciphertext are considered known to the public. Depending on which model the attacker is using, the plaintext can also be known. The only thing that is kept a secret is the cryptographic key. If the attacker acquires the key all future secret messages belonging to that key, and associated older messages, are considered lost and public.

Most of the cryptographic algorithms today are stored at personal computers. The laptop is more than capable of encrypting a message with different encryption schemes. To make this possible the private key also has to be stored on the computer. The issues with personal computers are their security. The computer can be a target for viruses, trojans, malwares, and worms which mostly spread through the internet. This bad intension software can steal the private key and expose the encrypted data to the public or a high bidding customer.

Nowadays there is specific equipment to store secret keys and encrypt messages. The equipment exists to provide better security standards and routines, e.g., hardware virtual private network (VPN) tunnels and universal serial bus (USB) with implemented security hardware.
These devices are capable of conducting cryptographic operations with the private key stored on it. Now it is not just the secret key that is sensitive but the whole cryptographic system and its implementation.

When an attacker "break" a cryptographic device, one has succeeded to extract the private key from the device. To extract a key from a cryptographic device, the attacker needs to have some knowledge regarding the actual hardware; therefore always assume the attacker has full knowledge of the device [1, p.3]. What the attacker does not have is the private key stored in it. As mentioned before there are different methods to extract this key.

There are examples of commercial electronics that reduce the electromagnetic radiation, e.g., in Sweden, it is called RÖS-protected electronics. The EM radiation could expose a security system to an outsider. The solution is to build thicker shielding which increases the weight drastically of the computer. It could be printers, laptops, and servers. If one could alter the actual layout of a device without adding extra shielding much could be gained; space, weight and economic cost. This type of protection does not prevent the attacker from analyzing the power consumption on the device.

To be able to conduct a side-channel analysis, physical access to the product is needed. There are examples of side-channel attacks with the use of cell phones. The cell phone is placed on top of the cryptographic device and begin collecting data through different electromagnetic sensors. If we consider a larger data storage center with servers, there could be computer racks with cryptographic devices. These data centers need maintenance of a vast variety. A large rotation of personnel creates an opportunity for the attacker to get physically close to the security system and its power supply. The attacker could extract the key without the owner knowing it. Alternatively, the owner can lose the cryptographic device and notice it days later. In Figure 1.2 a fictive threat scenario is presented.

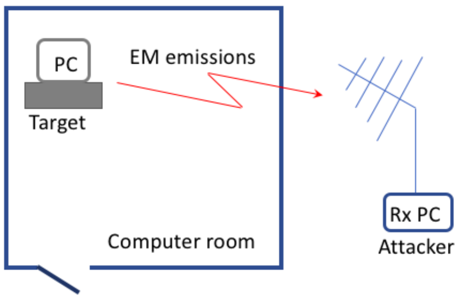


Figure 1.2: Illustrative method for attacking a cryptographic device.

This thesis will focus on the electromagnetic radiation from an FPGA.
The FPGAs have increased in popularity in the computer security industry because of their flexibility to generate a unique, inexpensive implementation of an encryption algorithm. FPGA is a good solution for accelerating different cryptographic operations.

1.2 Project aims

The primary goal of this project is to investigate possible data leakage from the FPGA. The Nexys-4 board with Artix-7 will be running an AES-encryption with a 256-bit key. The FPGA will be encrypting random text on the loop.

1.2.1 Formalization of questions

1. How is it possible, with EM measurements, to extract the AES subkeys directly from the FPGA? If extraction of the key is not successful, what could be the reason for this?

2. How many traces, and how much time, is needed before a correct subkey can be extracted?
3. Discuss the economic aspect of conducting a partial successful EM side-channel analysis?
4. With the result of question 1. How is it possible to extract the same information outside the cryptographic device, at a distance of 10 cm?

1.3 Related work

There has been some previous work in this area. The thesis relies heavily on [1] to study the theory regarding simple and differential analysis. There are examples of both successful and unsuccessful EM side-channel attacks. In [4] the researchers made a correlation electromagnetic analysis (CEMA) on Kintex-7 Xilinx FPGA chip. The researchers used a specially designed FPGA board for side-channel analysis and were successful in retrieving all 16 subkeys with 7000 EM traces. Their results vary depending on their parameters, but overall they made a successful attack and came to the conclusion that it is the actual position of the key that affects its signal-to-noise ratio (SNR) value and not the actual value of the key. The experiments in this thesis contain a side-channel attack on the FPGA Artix-7 with AES-256. The side-channel attack is also made with less expensive equipment and at a distance from the FPGA. There is also a master thesis [5] where the students only attacked an implemented AddRoundKey and SubBytes in the initial and first round of AES.

A great source of information is also Elke De Mulders work [6]. She has studied different techniques for conducting side-channel attacks and provides much experience regarding which type of equipment and method to use when making an electromagnetic side-channel attack.

The results from this thesis were analyzed and compared with other research on side-channel attacks. The numbers of EM traces, data points, and frequencies were compared with our methods and results to check if the findings were unique in some way.

1.4 Scope

There are several different side-channel attacks besides electromagnetic analysis (EMA) which is mentioned in Section 2. The electromagnetic analysis (EMA) can be divided into subgroups. The thesis will only include the theory of simple electromagnetic analysis (SEMA) and differential electromagnetic analysis (DEMA). Most of the theory is based on power analysis, but the method is still the same except for some of the equipment. The research will be more on ideal, than realistic, scenarios. The use of ideal situations is primarily because of the time needed to conduct the study. The thesis focuses mainly on Nexys-4 with Artix-7 FPGA.

The thesis will only consider the AES-256 and only the encryption, not decryption. There is no (known to the student) research regarding side-channel attacks on Artix-7 chip and a side-channel attack at a distance from the FPGA. To control and verify some of the results from the thesis a comparison with previous results will be made.

1.5 Outline

Five sections build the foundation of the thesis. First, there is a chapter dedicated to the theory for the project in Section 2. Section 2 describes the side-channel attacks and the mathematics behind retrieving the secret key. Section 3 includes the method which describes the practical aspects of the research. The method informs the reader about the equipment and how the experiments were conducted.

Section 4 includes the results from Section 3. The findings are presented without discussion. Finally, Section 5 includes a discussion regarding the results in Section 4. The section also provides answers to the formalized questions.

Chapter 2

Theory

There are five different side-channel attacks known today. The different side-channel attacks take advantage of physical data leakage through audio, electromagnetic emissions, power consumption, temperature, and timing analysis [7, p.28]. The thesis will focus on one of them; electromagnetic side-channel analysis/attack. This chapter presents FPGA, the cryptographic algorithm AES, and simple/differential analysis.

2.1 Field programmable gate array

An FPGA is an integrated circuit which contains different logic cells, look-up-tables, and block random access memory (RAM). It is a reconfigurable chip and can produce the same hardware logic as an application specific integrated circuit (ASIC), but it is not equally fast when doing computations, and it consumes more power. The advantage of an FPGA is the flexibility of creating dedicated hardware for specific operations. There are different kinds of FPGAs with different properties. Some FPGAs can only be configured once. After the first configuration, it will act as an ASIC but with higher power consumption and slower speed. An advantage with FPGA is that small companies, which can not afford to design their own ASIC, can buy FPGAs and then accelerate different computational operations. Some FPGAs are reconfigurable after the first configuration, which opens up possibilities to make flexible solutions. The FPGA is still much faster than a microprocessor with included software.

2.2 Advanced encryption standard

National Institute of standards and technology (NIST) initiated the work for finding a replacement for data encryption standard (DES) January 1997. The new standard is named advanced encryption standard (AES). AES was announced November 26, 2001 [8]. The AES algorithm must have the following properties:

- 128-bit block cipher with three different key sizes of 128, 192 and 256 bits.
- At least as secure as two-key triple-DES.
- Royalty free.

2.2.1 AES architectural overview

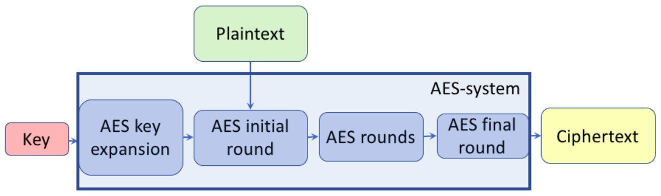
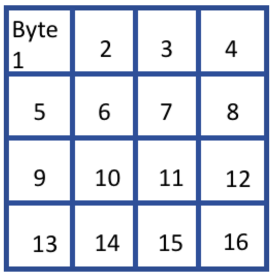


Figure 2.1: Block overview of the AES algorithm.

Figure 2.1 presents the overall architecture of the AES-algorithm. Depending on the size of the key, the number of rounds will vary. Each round is a number of operations with parts of the expanded key. When sending the secret key into the AES-algorithm, it is expanded to fit the number of rounds. This project uses a 256-bit key. The 128-bit key will generate ten rounds, the 192-bit key will be 12 rounds, and the 256-bit key will produce 14 rounds. Using more rounds implies more security against cryptanalysis. Using heavy computational cryptographic algorithm will require more power. Therefore, it is a balance between the need for security, performance and power consumption. The initial round and the final round, in AES, is partly modified and differ from the usual rounds. The differences will be described in the Section 2.2.2.

2.2.2 AES design

AES uses substitutions and permutations. It requires 10, 12 or 14 rounds plus the initial round. The number of rounds depends on the length of the secret key (128, 192 or 256 bits). The encryption algorithm accepts 16 bytes as input for encryption. Consider the input data as a matrix of size 4x4, see Figure 2.2. The purpose of the different operations is not within the scope of this project.



Byte 1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	16

Figure 2.2: The 16-byte AES matrix.

The typical structure is divided into several suboperations, see list below and Figure 2.3.

1. Key expansion
2. Initial round (Not included in the 14 rounds)
   - AddRoundKey
3. Rounds (iterated 9,11 or 13 times depending on key length.)
   - SubBytes
   - ShiftRows
   - MixColumns
   - AddRoundKey
4. Final round
   - SubBytes
   - ShiftRows
   - AddRoundKey

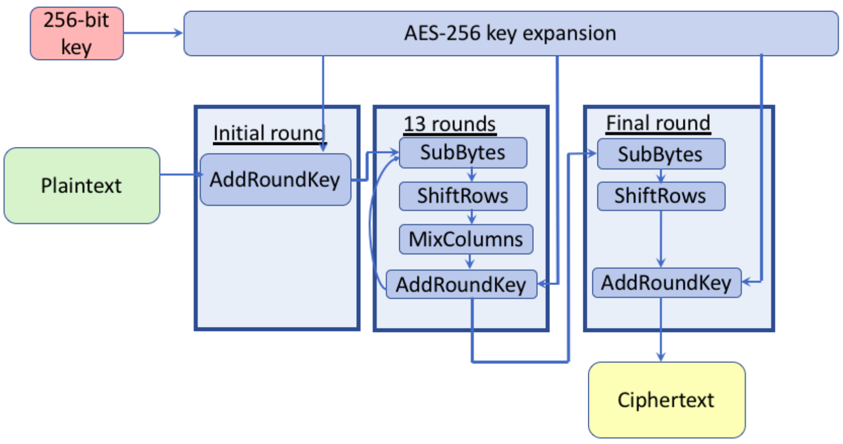


Figure 2.3: The AES scheme with suboperations and rounds. Figure presents a detailed version of Figure 2.3.

The final round is slightly different compared to the rest; it has no MixColumn operation.

Key expansion

Each round uses a modified version of the original secret key. This phase is also called **Rijndael key schedule**. Each variant of AES (128, 192, 256) uses a 128-bit key for each round, plus the initial round. Therefore the key expansion uses the 256-bit key and generates several subkeys for the different rounds. To expand the key for each round, a specific key schedule is used. The theory behind the key expansion is not within the scope of this project.

AddRoundKey

The initial round starts with the AddRoundKey. AddRoundKey is an XOR-operation (eXclusive OR) between the key and the plaintext. Consider the XOR-operation as: $f(d, k) = b$, where d is the data block (byte) and k is the key byte. The whole operation can be viewed in Figure 2.4. Notice the modified 256-bit key, which is now a 128-bit round key (4x4-key matrix). See Figure 2.4.

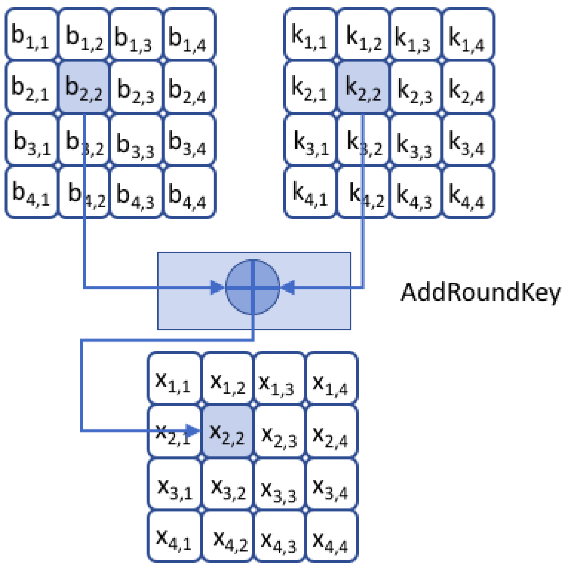
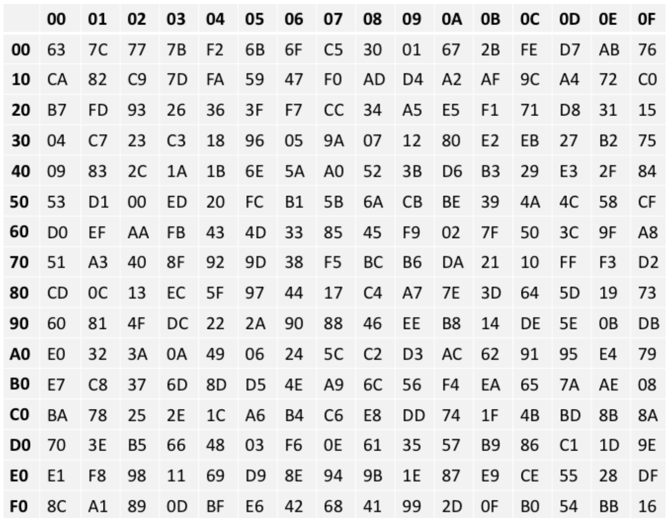

$b_{1,1}$ $b_{1,2}$ $b_{1,3}$ $b_{1,4}$
$b_{2,1}$ $b_{2,2}$ $b_{2,3}$ $b_{2,4}$
$b_{3,1}$ $b_{3,2}$ $b_{3,3}$ $b_{3,4}$
$b_{4,1}$ $b_{4,2}$ $b_{4,3}$ $b_{4,4}$
$k_{1,1}$ $k_{1,2}$ $k_{1,3}$ $k_{1,4}$
$k_{2,1}$ $k_{2,2}$ $k_{2,3}$ $k_{2,4}$
$k_{3,1}$ $k_{3,2}$ $k_{3,3}$ $k_{3,4}$
$k_{4,1}$ $k_{4,2}$ $k_{4,3}$ $k_{4,4}$
AddRoundKey
$x_{1,1}$ $x_{1,2}$ $x_{1,3}$ $x_{1,4}$
$x_{2,1}$ $x_{2,2}$ $x_{2,3}$ $x_{2,4}$
$x_{3,1}$ $x_{3,2}$ $x_{3,3}$ $x_{3,4}$
$x_{4,1}$ $x_{4,2}$ $x_{4,3}$ $x_{4,4}$

Figure 2.4: The AddRoundKey operation.

SubBytes

In this operation, the byte is sent into an s-box. The s-box is a substitution box. The byte sent in gets a new value according to a schematic, see Figure 2.5. When implementing AES in software or hardware, the designer often uses a lookup-table for this matrix. How these values are generated is outside the scope of this project [8].



	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
10	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
20	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
30	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
40	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
50	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
60	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
70	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
80	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
90	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A0	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B0	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C0	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D0	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E0	E1	F8	98	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F0	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figure 2.5: The s-box in hexadecimal. The input is one byte from the 4x4 data matrix. e.g., input 0x3A ⇒ 0x80.

ShiftRow

In this operation, the rows in the matrix (4x4) are shifted to the left. Rows shift data blocks cyclically by a certain offset. The first row is left unchanged. The second row shifts its data blocks one element to the left. The third row shifts two data blocks to the left. See Figure 2.6.

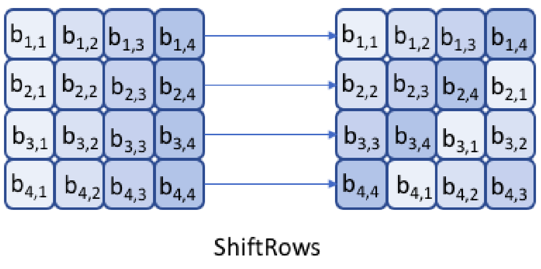

$b_{1,1}$ $b_{1,2}$ $b_{1,3}$ $b_{1,4}$
$b_{2,1}$ $b_{2,2}$ $b_{2,3}$ $b_{2,4}$
$b_{3,1}$ $b_{3,2}$ $b_{3,3}$ $b_{3,4}$
$b_{4,1}$ $b_{4,2}$ $b_{4,3}$ $b_{4,4}$
$b_{1,1}$ $b_{1,2}$ $b_{1,3}$ $b_{1,4}$
$b_{2,2}$ $b_{2,3}$ $b_{2,4}$ $b_{2,1}$
$b_{3,3}$ $b_{3,4}$ $b_{3,1}$ $b_{3,2}$
$b_{4,4}$ $b_{4,1}$ $b_{4,2}$ $b_{4,3}$
ShiftRows

Figure 2.6: The ShiftRows operation.

MixColumns

The MixColumns [8] operation uses a specific function *g(x)* and a binary multiplication. The function is part of an invertible linear transformation. Every single byte sends into MixColumns affect the four output bytes. ShiftRows and MixColumns create what is known as diffusion in cryptography.

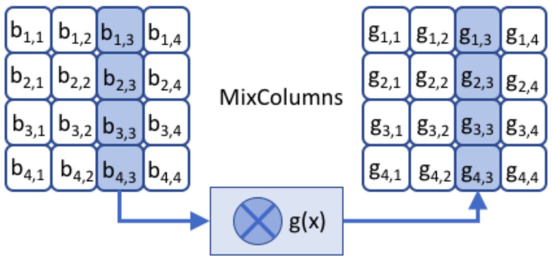

$b_{1,1}$ $b_{1,2}$ $b_{1,3}$ $b_{1,4}$
$b_{2,1}$ $b_{2,2}$ $b_{2,3}$ $b_{2,4}$
$b_{3,1}$ $b_{3,2}$ $b_{3,3}$ $b_{3,4}$
$b_{4,1}$ $b_{4,2}$ $b_{4,3}$ $b_{4,4}$
MixColumns
$g_{1,1}$ $g_{1,2}$ $g_{1,3}$ $g_{1,4}$
$g_{2,1}$ $g_{2,2}$ $g_{2,3}$ $g_{2,4}$
$g_{3,1}$ $g_{3,2}$ $g_{3,3}$ $g_{3,4}$
$g_{4,1}$ $g_{4,2}$ $g_{4,3}$ $g_{4,4}$
g(x)

Figure 2.7: The MixColumns operation.

2.3 Origin of side-channel analysis

The physics behind an electronic device creates observable phenomena. Computing tasks create heat, electromagnetic fields, power consumption and noise. These phenomena can be measured with different equipment.

The technology in an FPGA is generally low voltage complementary metal oxide semiconductor (CMOS), see Figure 2.8. The different transistors act like an opening (or closing) current gate for the circuit. When a current start flowing in the circuit, a magnetic field is generated from the wires. This magnetic field can be extracted with an antenna. Much of the theory relies on the *Power Analysis; Revealing the secrets of Smart Cards* [1]. The theory for electromagnetic analysis is the same except for the origin of information leakage. That is why the thesis will use references to literature which explain power analysis.

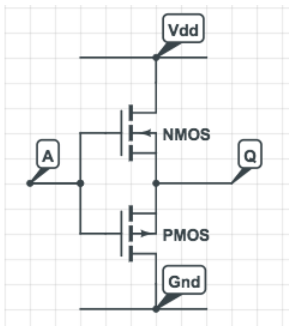


Figure 2.8: CMOS including a NMOS and a PMOS transistor.

2.4 Simple analysis

When a microprocessor or an FPGA makes computations, it sends out EM emissions. It could be a magnetic field or electric field. An example of a loop antenna extracting magnetic emission from an FPGA can be viewed in Figure 2.9.

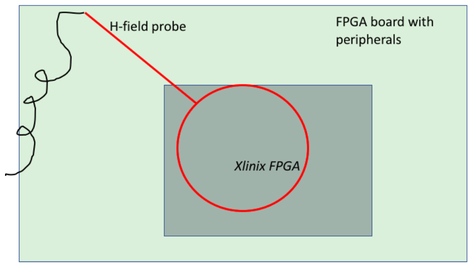


Figure 2.9: An example of electromagnetic measurement on FPGA with a loop antenna.

The attacker can observe a specific EM trace on an oscilloscope when the processor is making computations. With the use of one EM trace, an attacker can extract the key from the cryptographic device. It is a technique that directly interprets the EM emissions collected during cryptographic operations. To succeed with a simple electromagnetic analysis (SEMA), the attacker needs detailed knowledge about the implemented encryption algorithm. The SEMA is useful if the attacker only gets access to the device during a short time and can extract few EM traces. A real-world application could be the hired-by-the-hour technician, trying to retrieve information from the cryptographic device without anyone noticing it.
E.g., the technician only has access for a couple of minutes.
The attacker tries to extract the secret key while the cryptographic device is encrypting data. SEMA is divided into *Single-shot SEMA* and *Multi-shot SEMA*. *Single-shot* is an attack where the attacker only got one EM trace to analyze. *Multi-shot* is a technique that uses several EM traces. With the use of several EM traces, the attacker can extract more meta information regarding the encryption.

2.4.1 Analyze data from an electromagnetic waveform

Every encrypting algorithm executes in sequential order on a microprocessor. The instructions are using a specific architecture to conduct encryption. The microprocessor instructions set architecture will use different arithmetic operations, logical, branching and data transfer instructions. Each of these instructions uses different parts of the processor or peripherals. These components have a specific EM trace [1]. The attacker can use an EM trace from an encryption device to figure out which instruction the microprocessor is using. If the number of instructions and the sequence of them is directly related to the secret key, there might be a good chance to extract the secret key. For example: if a bit is 1, a particular operation takes place. If the bit value is 0, then another operation is made. Worth mentioning is the vital relationship between the sequence of instructions and the secret key. The SEMA will only succeed if there is a strong relationship between these two. The extraction of the key relies heavily on the executed operations and the secret key.

In this thesis, SEMA will be used to extract metadata from AES-256. There is full access to the hardware implementation which makes the SEMA more valuable. However, to be realistic, assume the attacker has no access to the netlist for the FPGA.

2.5 Differential analysis

The differential analysis does not require knowledge of the target device, which makes the attack easier to conduct. There is also a possibility to make a differential analysis on devices which generate much noise and still extract the secret key. Compared to simple analysis, differential analysis requires a large number of EM traces to reveal the key. Each trace is recorded while the cryptographic device is encrypting one block of data. Therefore it creates a requirement for the attacker to have physical access to the device for a longer time than simple analysis.

With the use of SEMA, the attacker extracts the key by visually observing and analyzing one EM trace along the time axis. When the attacker is using a differential electromagnetic analysis (DEMA), one uses statistical methods to reveal the secret key. In DEMA the attacker compares different EM traces with a given set of data, at a specific time. DEMA focuses primarily on the data dependency of the EM traces. According to [1], the definition of differential analysis is: **Differential Power Analysis attacks exploit the data dependency of the power consumption of the cryptographic devices. They use a large number of power traces to analyze the power consumption at a fixed moment of time as a function of processed data**. In this case, exchange the words "power consumption" with "EM trace". The theory is still the same.

2.5.1 How to practically conduct a differential analysis

There are several practical steps to conduct a differential analysis [1, p.119].

Step 1: Intermediate result of the encryption algorithm

The first step in a differential analysis is to decide which area of the algorithm to attack. The attacker chooses intermediate results which are written as $b = f(d, k)$, where d is the data block, and k is the not known partial subkey. d is often the plaintext or ciphertext. d is a known non-constant value. b is the intermediate result and could be from anywhere in the encryption algorithm.

Step 2: Measure EM traces

In this phase, the attacker captures EM traces from the cryptographic device when it is encrypting D data blocks. For each data block d the attacker needs to know the actual data value of d. This data value is recorded together with the corresponding EM trace. The data block is a vector $\mathbf{b} = (d_1, ..., d_D)$, where d_i is the data block of the ith encryption run. Each EM trace includes several thousand data points from the oscilloscope. The EM traces data points are collected in a vector $\mathbf{t}_i = (t_{i,1}, ..., t_{i,T})$ where T is the number of data points from the EM trace, i is the sequence number of the data block. This results in a data block d_i with corresponding EM trace vector $\mathbf{t}_i$. The number of data blocks D, and the number of EM traces data point T, create a matrix DxT. The attacker needs to measure the EM traces at precisely the same time as the EM trace before. Otherwise, the following steps (statistical model) will not work correctly.

Step 3: Hypothetical intermediate values

In the third phase the attacker creates simulated values from a specific EM model. With the use of every possible k in $\mathbf{k} = (k_1, ..., k_K)$ and the vector $\mathbf{d}$, the attacker create hypothetical values with the function $f(d, k)$. The results are stored in a matrix $\mathbf{V}$ of size DxK.

$$v_{i,j} = f(d_i, k_j) \qquad i = 1, ..., D \qquad j = 1, ..., K \tag{2.1}$$

The goal of a DEMA attack is to find which column of $\mathbf{V}$ the D encryption uses.

Step 4: Create hypothetical EM model

In this step the matrix $\mathbf{V}$ is matched with a matrix $\mathbf{H}$. Matrix $\mathbf{H}$ is matrix with hypothetical EM traces. To create the $\mathbf{H}$ matrix, the attacker is using an EM simulation model. There are several different EM simulation models. Some of these models are more adapted to a microprocessor and some to FPGA. According to [1] the Hamming distance will be a good suggestion for EM modeling. If the attacker has more knowledge about the device, e.g., the netlist, one could make better EM simulations of the algorithm. Better EM simulations will reduce the number of EM traces needed for revealing the secret key.

Step 5: Compare hypothetical EM values with EM traces

After the mapping of $\mathbf{V}$ to $\mathbf{H}$ the attacker uses matrix $\mathbf{H}$ and compares it with the collected EM traces $\mathbf{T}$. Comparing the hypothetical EM values with the measured EM values from real encryption, can reveal the secret key. Each column in $\mathbf{H}$ is compared with the EM trace vector/column $\mathbf{t}_j$ from matrix $\mathbf{T}$ in step 2. Remember that matrix $\mathbf{T} = D \ x \ T$ includes all the EM traces belonging to each encryption run where D is the number of data blocks. The final results will hold a comparison between hypothetical EM values with the key hypothesis and the measured EM traces. The comparison is stored in matrix $\mathbf{R}$, where $\mathbf{R}$ is of the size $K \ x \ T$. The comparison between h_i and t_j can be done with different statistical methods. Each element in $\mathbf{R}$ $(r_{i,j})$ contains the result between h_i and t_j. The element value of $r_{i,j}$

should be as high as possible. A high element value implies there is a correlation between the hypothetical EM trace and the real EM trace. The index value of the element $r_{i,j}$ will give the attacker the key and intermediate value. Remember that h_i is the hypothetical EM trace for a key k-hypothesis and a plaintext data block d. The overview of the steps is visualized in Figure 2.10.

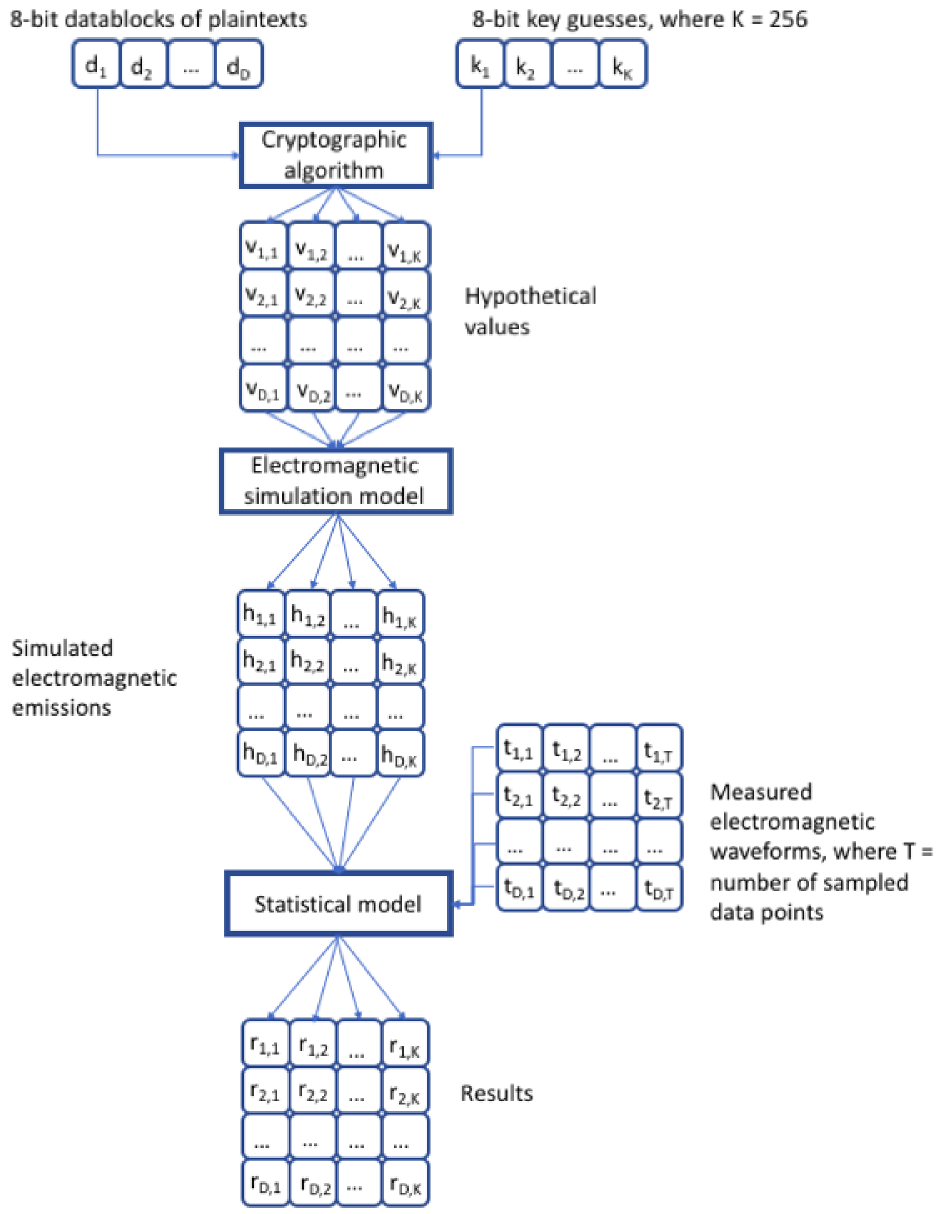

8-bit datablocks of plaintexts
d1 d2 ... dD
8-bit key guesses, where K = 256
k1 k2 ... kK
Cryptographic algorithm
Hypothetical values
Electromagnetic simulation model
Simulated electromagnetic emissions
Measured electromagnetic waveforms, where T = number of sampled data points
Statistical model
Results

Figure 2.10: The different steps in differential analysis.

2.5.2 Electromagnetic simulation models

There are different methods to model an EM trace of a circuit. If the attacker has profound knowledge about the device, one can create a good model that accurately simulates the EM traces. When an attacker makes a differential analysis, the assumption is that one does not know the circuit layout. With this starting point, there are two different models of power/EM simulations on FPGA [9][10]. There are several researched EM/power simulations models regarding FPGAs, but the assumption for this thesis is: the attacker has no, or very reduced, knowledge of the device. According to [9] and [10] there are only two models that could give an approximation of the hypothetical EM emission with little knowledge of the cryptographic device; the Hamming weight and the Hamming distance. The position of the electromagnetic simulation model in the DEMA can be viewed in Figure 2.10.

Hamming weight

The Hamming weight [1] model analyzes the processing bits of value 1. In this thesis, it will be the number of 1s in the intermediate results from Section 2.5.1 step 1. The result is dimensionless but will give a good indication of the EM emission in that operation. To give an example: Hamming weight of 1001 0110 = 4. $H(D) = 4$ where $D = 10010110$.

Hamming distance

In CMOS technology, operated at high frequency, the primary EM emissions are not in the static state of the transistor. The primary EM emission is in the dynamic state of the transistor. When currents are changing it creates a magnetic field that the attacker can extract. Hamming distance [11] is a developed Hamming weight model. First, there is an XOR-operation, and then the number of 1s is counted in the result. Let D denote an old 8-bit data value. Let R denote the new value of the 8-bit register. Then the Hamming distance can be written as:

$$HD = a \cdot H(D \oplus R) + b \tag{2.2}$$

HD represent the simulated EM emission, H is the Hamming weight function. Often $a = 1$ and $b = 0$, the reason for this is outside the scope of this thesis. $a = 1$ and $b = 0$ simplify the Hamming distance to

$$EM = H(D \oplus R) \tag{2.3}$$

Hamming distance is a better option because it takes into consideration which bits that have hanged its value. According to Ampère - Maxwell law, a change in current will create a dynamic magnetic field. A varying magnetic field induces a current in a wire. Since the experiments are using a loop antenna, the measured EM traces will become better if a dynamic magnetic field is used instead of static.

2.5.3 Statistical models

In Figure 2.10 there is a state called "Statistical model". In this state, the goal is to produce results which can reveal the secret key. There are several different models to extract the key. In this thesis two models will be introduced; Distance-of-mean [1] and correlation electromagnetic analysis [12] with Pearsons correlation coefficient. After the introduction of the methods, one will be chosen.

Distance-of-means

The distance-of-mean is a method that is based on a selection function D. With the function D the attacker decides which bit (MSB or LSB) to target. Refer to this bit as d. When the different plaintexts are encrypted the radiated EM traces are extracted and sorted based on selection function D. With 1000 different plaintexts, there would be around 500 EM traces with d=0 and 500 traces with d=1. By calculating the mean value from the collected traces, d=1 and d=0, the attacker can determine the effect d has on the EM emissions. After calculating the mean value, the two results are subtracted from each other. The largest differences reveal where in time the power consumption depends on d.

Pearson's correlation coefficient

Pearsons correlation coefficient is denoted r. One dataset is $x_1, ..., x_n$ where n is number of values, and one data set is $y_1, ..., y_n$.

$$r = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2} \cdot \sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2}} \tag{2.4}$$

$\bar{x}$ and $\bar{y}$ is the sample mean and is given by

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{2.5}$$

$$\bar{y} = \frac{1}{n}\sum_{i=1}^{n} y_i \tag{2.6}$$

Pearson correlation coefficient is a good method for detecting a linear relationship between data. It also creates better results but requires more computations. The reason why Pearson correlation method creates better results is outside the scope of this thesis. Due to the fact that Pearson correlation function produces (in general) better results, this method was chosen as the statistical model.

Chapter 3

Method

3.1 Equipment

The equipments used in the experiment are presented in the list below and the actual lab setup is presented in Figure 3.1, 3.2 and 3.3.

- - RIGOL DS1054Z Oscilloscope.
- - EM probe kit.
- - Nexys 4 FPGA-board with Artix-7 from Digilent.
- - Spectrum Analyzer 9kHz-3.5GHz Rohde & Schwarz.
- - Voltage probes
- - Powerbox3000B
- - Broadband amplifier ZHL-2-8
- - LabView 2017
- - Python 3
- - MatLab 2017b
- - PC
- - Xilinx Vivado

3.2 Experiment setup

A block schematic of the experiment setup is presented in Figure 3.1.

- PC: The computer is used to make computations and control the oscilloscope with LabView. In the LabView program, the number of encryptions is compared with the number of trigger pulses. They should match if the measurements are made without corruption.
- Powerbox 3000B: This is the power supply for the amplifier.
- Amplifier: Connected between the EM probe and the oscilloscope.
- Xilinx FPGA: The FPGA is mounted on the Digilent Nexys-4 board. From the board, a trigger signal is connected to channel two on the oscilloscope. The USB provides the power supply from the PC.
- Oscilloscope: Capture EM traces on channel one and presents it on display.
- EM probe: 1 cm, or 5 cm diameter loop antenna.

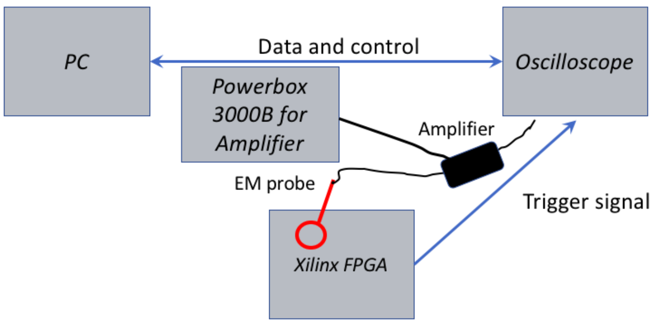


Figure 3.1: Schematic overview of the experiment setup.

In Figure 3.2 the lab environment is presented. The instrument in top left of Figure 3.2 is the spectrum analyzer. Figure 3.3 is a zoomed in version of the Digilent Nexys-4 board with Artix-7 FPGA and an EM probe.

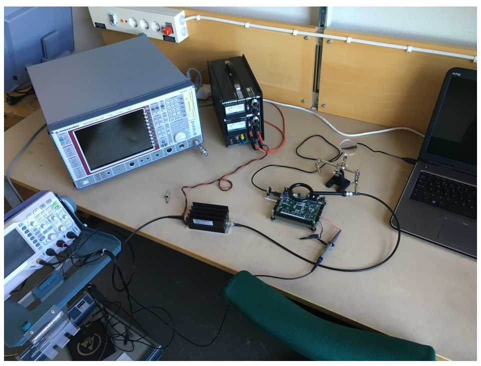


Figure 3.2: Lab setup.

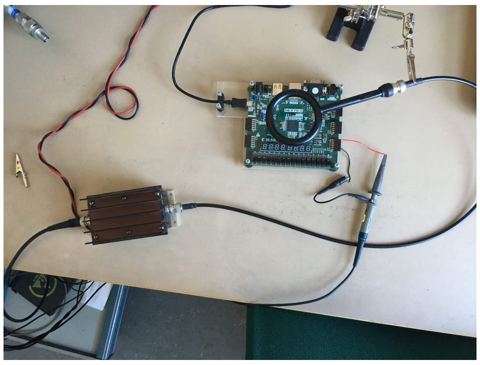


Figure 3.3: How measurements are done with a 5 cm loop antenna.

3.3 FPGA setup

The experiments are divided into two different attacks. One attack is on Advenicas IP, and one attack is on a self-designed initial and first-round AES. The self-designed round will be named "simplified AES" and the Advenicas IP as "AES-256". The purpose of the separation of attacks is to test different hardware designs when extracting the EM traces. With a more straightforward and less complicated design of the target, the experiments hope to provide an easier target for EM simulation model which can deliver results that are very close to the reality. See Figure 3.8 for system overview of simplified AES. More explanation regarding separation of attacks will be in Section 3.5.

To conduct a side-channel analysis on an FPGA, it needs to run an encryption algorithm. The AES algorithm IP from Advenica has a peripheral component interconnect express (PCI-e) as an interface. Implementation of the PCI-e interface on the FPGA is out of the scope of this thesis. Instead, an encryption loop was created on the FPGA with a linear feedback shift register (LFSR) as a pseudo-random text generator. With this type of setup, a very high speed integrated circuit hardware description language (VHDL) wrapper, encapsulated the whole AES implementation, called AES-wrapper. The wrapper will give commands to the AES implementation and send the plaintexts to the encryption algorithm. Every time an encryption starts, it sends a trigger signal to the oscilloscope. A trigger signal is not a realistic scenario. In a successful electromagnetic side-channel attack in the real-world application, the different EM traces need to be aligned along the time axis. Implementation of the aligning algorithm is outside the scope of this thesis. Instead, a trigger signal from the FPGA will solve the aligning issue.

3.3.1 AES-wrapper for AES-256

The wrapper will work as a "testbench"; feeding Advenicas AES IP with plaintexts, keys and start signals. In the wrapper, different numbers of plaintexts are generated and sent into the AES component. The plaintexts will contain the data blocks d_i from step 2 in Section 2.5.1. The encryption key is stored in the AES-wrapper. The hexadecimal value of the 256-key is 0x000102030405060708090A0B 0C0D0E0F101112131415161718191A1B1C1D1E1F. See Appendix B.1 for VHDL code for the wrapper.

Figure 3.4 presents the finite state machine (FSM) of AES-wrapper. Implementations for the different states can be found in the Appendix B.1. The main task for the FSM is to keep track of the control signals, send trigger signals at the right time in operation, feed the AES encryption with plaintexts from the LFSR at the right moment. The reason for using an LFSR in the wrapper is to generate random text strings which can symbolize different data transmission over a data link.
When the oscilloscope receives the trigger signal, it starts to measure the whole encryption run or a specified AES round. In this thesis, the initial round and the SubBytes operation in the first round was targeted. The reason is to be able to extract as many subkeys as possible from the original 256-bit key. Remember that AES-256 uses its first 128 key bits in the initial round and the SubBytes operation. The rest of the 256-bit key is expanded and spread out for the rest of the 13 rounds. Extracting the 128 bits from AES initial and first round would give 50% of the 256-bit key.

The oscilloscope shall present 14 distinct AES rounds on the oscilloscope screen, plus one memory storage cycle and one for the AES wrapper. There should be 16 distinct peaks if a whole encryption run is captured and displayed on the screen. To capture different parts of the encryption algorithm, the offset can be configured to start at different moments.

The internal system clock at 100 MHz is too fast for the oscilloscopes bandwidth at 50 MHz, according to the Nyquist theorem. Therefore, a 12.5 MHz clock was used as the encryption frequency. The 12.5 MHz is created by dividing 100 MHz with a hardware implemented counter. If the clock is too fast for the oscilloscope (which has 50 MHz bandwidth), the extracted data becomes corrupted and represents wrong values from the encryption. The oscilloscope is communicating with the PC through USB. The USB interface further reduces the data throughput from the oscilloscope.

Below 12.5 MHz, the signals from the FPGA became very weak. The magnetic coupling increases with the square of the frequency. The relationship between frequency and magnetic coupling implies that the magnetic amplitude is reduced by a factor of 4 if the frequency is reduced by 50%. Therefore 12.5 MHz was selected as the highest possible frequency for the oscilloscope with frequency divider. A higher frequency also adds extra switching noise from the transistors on the FPGA.

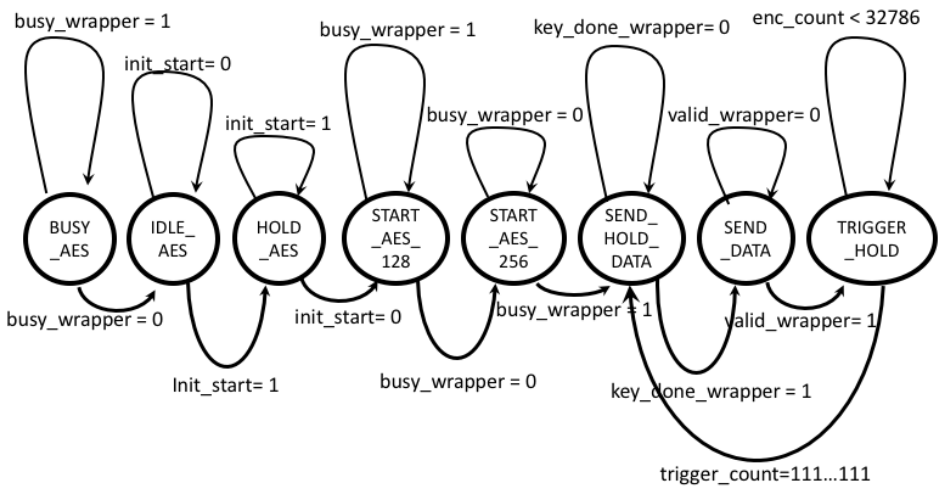


Figure 3.4: FSM of the AES-wrapper. See Appendix B for VHDL code.

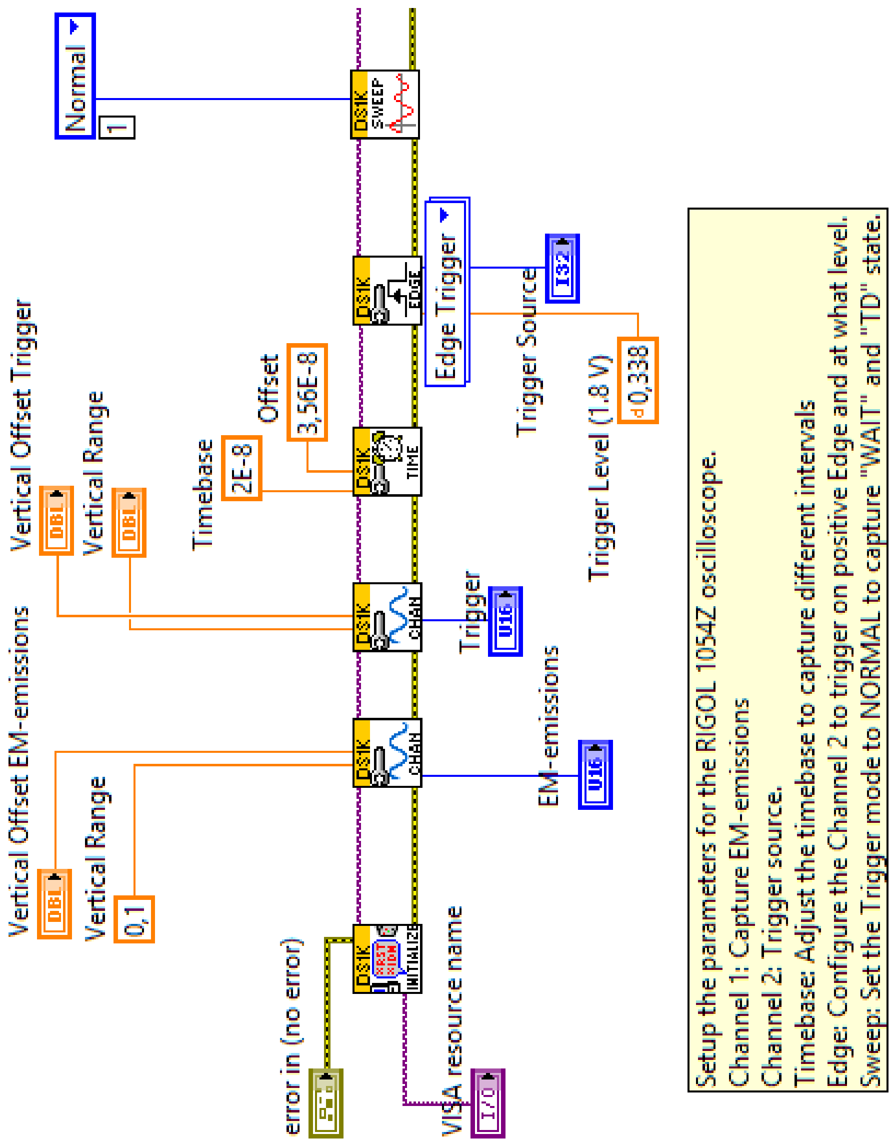
3.3.2 Oscilloscope

The control of the oscilloscope is made in detail with help from LabView. The oscilloscope is set to NORMAL trigger mode. NORMAL trigger mode presents the waveforms when the trigger condition is met otherwise it keeps the previous waveforms from last trigger pulse. See Figure 3.5 and 3.6 for LabView block diagram with oscilloscope setup and explanations.

Every EM trace contains 1200 data points. The number of sampled points is a low number of data points, but it made the EM trace extraction faster. The alternative was to put the oscilloscope in SINGLE-mode, extend the memory depth and collect more data points. If the oscilloscope is set to SINGLE-mode, it has to stop the data acquisition, fill up the internal memory and then send it to the PC. In the experiments, the screen memory buffer was used as primary storage before sending the data to the PC. The screen memory buffer only saves what it can display on the screen. Due to the screen memory buffer, the oscilloscope had to be configured to capture/display the targeted intermediate value from step 2 in Section 2.5.1; first round in AES with parts from neighboring cycles. The most important part is that the trigger pulse becomes active exactly at the same time in the encryption operations and that it encapsulates the targeted operations.

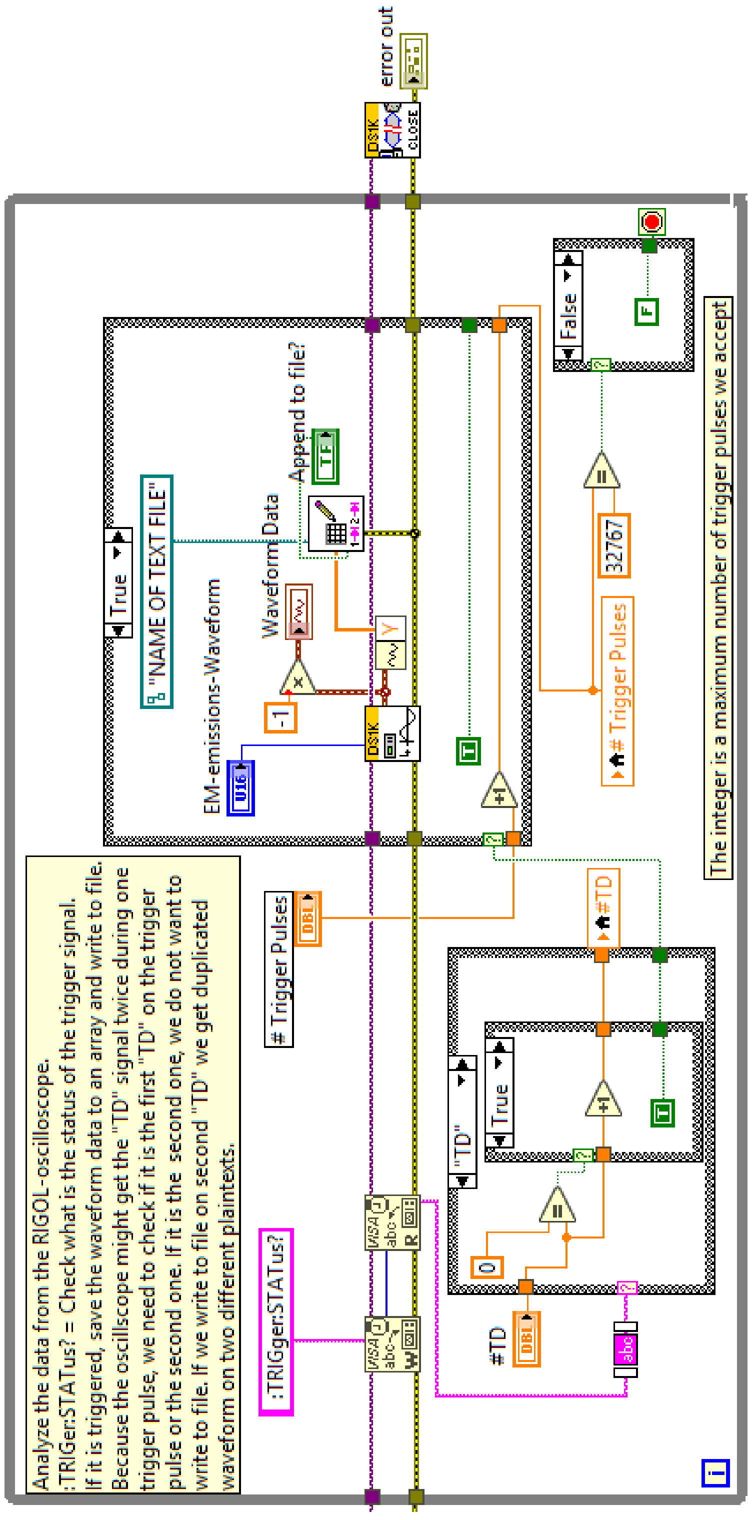
The side-channel attack will use a loop antenna according to Elke De Mulders conclusions in [6]. The FPGA will create small current loops. Extracting the magnetic field from the current loops require a loop antenna which can encapsulate the currents on the FPGA. Extending the diameter of the loop antenna will result in more extracted signals, but they will be weaker in amplitude. In the third data acquisition, the side-channels are exploited at a distance of 10 cm above the FPGA-chip with a 5 cm in diameter loop antenna.

In Figure 3.1 there is an included amplifier with supporting powerbox. The EM emission from the FPGA is very weak therefore an amplifier is needed to amplify the signals before they enter the oscilloscope. With an amplifier, there was mV as input signals. Without the amplifier, almost all of EM traces were lost due to their low amplitude. See Figure 3.5 and 3.6 for LabView setup.



Setup the parameters for the RIGOL 1054Z oscilloscope.
Channel 1: Capture EM-emissions
Channel 2: Trigger source.
Timebase: Adjust the timebase to capture different intervals
Edge: Configure the Channel 2 to trigger on positive Edge and at what level.
Sweep: Set the Trigger mode to NORMAL to capture "WAIT" and "TD" state.

Figure 3.5: First block in LabView. Setup the parameters for the oscilloscope.



Analyze the data from the RIGOL-oscilloscope.
:TRIGer:STATus? = Check what is the status of the trigger signal.
If it is triggered, save the waveform data to an array and write to file.
Because the oscillscope might get the "TD" signal twice during one
trigger pulse, we need to check if it is the first "TD" on the trigger
pulse or the second one. If it is the second one, we do not want to
write to file. If we write to file on second "TD" we get duplicated
waveform on two different plaintexts.

Figure 3.6: Second block in LabView. Collect and save the data to file.

3.3.3 Scanning the FPGA for EM emissions

When conducting an electromagnetic side-channel analysis on an FPGA chip, it is first required to locate where on the chip the emissions are radiated. Using a probe with a diameter larger than the size of the chip the attacker will collect all signals from the chip. In later attacks, the aim is to extend the distance from the chip to investigate the possibilities to extract subkeys without being physically close to the cryptographic device. However, as a first step, it is important to locate where on the chip there are distinct EM emissions. With the use of the actual floorplan of the layout, it is easy to compare the actual location of the EM emissions with the implemented logic cells. It is essential that the EM emissions be captured from the correct logic cells.

3.3.4 Building tool for collecting EM traces

It is crucial that the EM traces be captured at the same point in time when running the encryption operations. With the use of the trigger signal, the oscilloscope can capture the data from the EM emission and store it locally. The data is saved when the trigger condition is met. When the trigger signal is LOW, the oscilloscope stops collecting data. The software LabView 2017 controls the oscilloscope from the PC. To run LabView 2017 with the RIGOL oscilloscope different drivers were needed to make the correct setup.

The spectrum analyzer will analyze the variety of frequency emitted from the FPGA. If measurements are made on signals that are not on 12.5 MHz, or harmonics of it, the oscilloscope will not capture the correct electromagnetic emissions. Wrong signals will generate false results.

3.4 Simple electromagnetic analysis

With the use of different probes on top of the actual FPGA-chip, several different traces are extracted to reveal the information about the encryption. The different probes were: electric field sphere antenna and magnetic field loop antenna. However, the loop-antenna with 1cm in diameter was the best option for extracting EM emissions at 0 cm. According to [6] the small loop antennas are the best probes for measuring close to the FPGA chip. The reason for using small loop antennas as probes is because of the small current loops that are created on the FPGA chip when it is operational. To extract the EM emission, the oscilloscope manually records the received signals, with trigger mode on NORMAL. When the trigger condition is met, the oscilloscope was stopped manually, and the waveform saved as a CSV file on a USB-stick.

The goal of this attack/analysis is to collect electromagnetic traces and then analyze them visually to extract information from the FPGA regarding encryption algorithm and frequency. As a first stage, a capture is made of all 14+1+1 rounds. The reason for this is to confirm that the trigger signal is working properly and to confirm the AES structure. After the first stage, the trigger signal was changed and instead encapsulated the initial round and SubBytes in the first round in AES.

3.5 Differential electromagnetic analysis

The DEMA is divided into two subsections; *Isolated attack on first AES round* and *Advenicas IP AES-256*. The side-channel attack will follow the steps made in Section 2.5.1 to conduct a differential analysis.

3.5.1 DEMA steps

Step 1: Intermediate result

The target is the output of the SubBytes operation in the first round. SubBytes is a valid target because of the non-linearity the s-box provide. If the target is just the XOR operation in AddRoundKey, not enough EM emissions will be generated from the switching transistors. Attacking other rounds in the implementation needs a more complex model for the hypothetical values. An attack on the whole 256-bit key requires different models when targeting different rounds. The increased complexity is because each round is using values from the previous round, except for the first round. There is also a possibility to target the last round if the attacker has the ciphertext. The attacker can then take the inverse of the s-box and get the intermediate value.

The AES algorithm is using 8-bit data blocks as a standard block size. In the python simulation model, the data chunks are separated into 8 bits. The AES-256 IP process all 128 bits in one clock cycle but they are sent into XOR and s-box operation as 8 bits in parallel. The intermediate result will be written as:

$$f(d, k) = SBOX(d \oplus k) \tag{3.1}$$

In Equation 3.1, k is the unknown subkey, the attacker wants to extract and d is 8 bits of data from the plaintext.

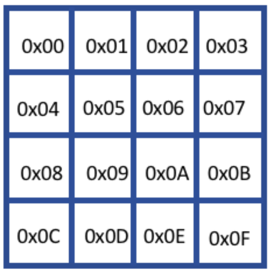
Step 2: Measure EM traces

To get correct EM traces, the FPGA sends a trigger signal to the oscilloscope. A trigger signal is not a very realistic scenario, but it is enough for this project. A more realistic scenario is a situation where an attacker collects EM traces and then afterward align them into the right time interval. The attacker needs the correct sequence of operations from the encryption algorithm to reveal the secret key. At least one trace for every specific data block is needed. Otherwise, there is no chance to find the correct key.

In this step, the EM traces are collected and stored as data points in a 2D-array (matrix). Since the order the LFSR output is known due to the seed, the VHDL wrapper can log the values and save them in a txt-file. The plaintexts can appear random, but always repeat the same pattern if the LFSR is provided with the same seed. Therefore, a sequence of 128-bit plaintexts is generated. With every trigger signal from the FPGA, the actual EM trace is stored and mapped to the corresponding 128-bit plaintext. 10000 trigger signals imply that the FPGA has encrypted 10000 plaintexts and captured 10000 EM traces.

Step 3: Hypothetical intermediate values

The AES encryption is using a 256-bit key. The key is divided into 8 bits chunks and presented as hexadecimal values. The 32-byte key used in the AES-256 encryption is
0x000102030405060708090A0B0C0D0E0F101112131415161718191A1B1C1D1E1F.
For the simplified AES version, 0x02 is the 8-bit input key. The key matrix used in the first round will according to Figure 3.7.



0x00	0x01	0x02	0x03
0x04	0x05	0x06	0x07
0x08	0x09	0x0A	0x0B
0x0C	0x0D	0x0E	0x0F

Figure 3.7: The first 16 key bytes used in first round of AES-256.

When the hypothetical values were created, the simulation used a key vector of all possible 8-bit keys, from 0x00 to 0xFF. These hypothetical values are then sent as an input into the python script, see Appendix A, which contains the intermediate value function from step 1. The key vector will create a matrix of different values according to the Table 3.1.

Table 3.1: Example of how hypothetical values are created.

$f(d,k) = SBOX(d \oplus k) = v_i$								
d_1	k_i	v_i	d_2	k_i	v_i	d_3	k_i	v_i
0x0F	0x00	0x76	0xF0	0x00	0x8C	0xCC	0x00	0x4B
0x0F	0x01	0xAB	0xF0	0x01	0xA1	0xCC	0x01	0xBD
0x0F	0x02	0xD7	0xF0	0x02	0x89	0xCC	0x02	0x8B
0x0F	0x03	0xFE	0xF0	0x03	0x0D	0xCC	0x03	0x8A
0x0F	0x04	0x67	0xF0	0x04	0xBF	0xCC	0x04	0xE8
...	...	-	...	...	-	...	...	-

The hypothetical values are now stored in a 2D array **V**. The data block d will take the value of 8-bit data chunks from the 128-bit plantext.

Step 4: Create hypothetical EM model values

The python script, with implemented Hamming distance function, accepts two inputs; 8-bit plaintext data block d and the hypothetical value v. The results from the Hamming distance is saved in a vector representing the relative EM values. When using 10000 8-bit plaintexts, 10000 hypothetical EM traces are generated for every possible key. The hypothetical EM traces will be of size 10000x256 for each 8-bit data block b. The plaintext contains 16 bytes which imply a size of 10000x256x16. In the python script, only 8 bits considered at a time and not 128 bits. The complexity of FPGA is that they process 128-bits in parallel, but the attacker tries to find a correlation with only 8 bits. Using 128 bits will generate 256^{16} different combinations instead of 256 combinations for 8 bits. The low number of combinations is the advantage of the side-channel attack compared to brute force the 128-bit key directly. 8 bits can be attacked in separate analysis and confirm the correct subkey.

Step 5: Comparison between hypothetical EM values and the measured EM values

The Pearson correlation function accepts two inputs; the measured EM values and the hypothetical EM values. The Pearson will take the first column in the hypothetical EM values matrix, $\mathbf{H}$, and loop through a comparison with all the columns in the measured EM values matrix $\mathbf{T}$. The function of Pearson method implies that one column of hypothetical EM values will be compared with 1200 sampled data points because that is the length of one row in $\mathbf{T}$. The final result is stored in a 2D-array $\mathbf{R}$, where $\mathbf{R}$ contains all the Pearsons correlation coefficients. The $\mathbf{R}$ rows are sorted according to their first elements correlation values, with the highest correlation value at top. The $\mathbf{R}$ matrix now have all the highest correlation values in the first column. The python script then selects the correct key and presents its rank. The correct key is stored in the python script to confirm if the key guess is correct or not.

The hardcoded correct key in the python script is compared with each key guess in the column of the best correlation value. When there is a match between the key guess and the correct key the rank of that key guess will be presented to the user. If the key guess gets the highest rank (**1**), then the hypothetical guess is the correct guess. If the rank is 256, then the correlation is worst among all other correlation values in the first column in $\mathbf{R}$.

These different DEMA steps can be tricky to grasp. A good way to understand the process is looking at Figure 2.10 or reading [1] from page 119.

3.5.2 Trigger signal

Due to the data speed of the oscilloscope, it was not possible to extract signals at a normal pace. Therefore a counter was used in the RTL design. This counter started at the falling edge of the trigger signal. During this period the oscilloscope saves the waveform data into an array and stores it in a CSV file on the computer.

Making the oscilloscope faster, by reducing the hold time after trigger signal, will only increase the loss of EM traces belonging to certain encryption. In this way the whole data acquisition becomes corrupt.

3.5.3 Isolated attack on first AES round

A new RTL-design was used to make the AES-attack less complex. With the use of only 8-bit key and 8-bit data, the subkey 0x02 was attacked. The hardware design of this construction can be viewed in Figure 3.8. The attack is similar to the AES-256. 10000 plaintexts are generated with an LFSR. Only the least significant 8 bits were used as a plaintext input to the simplified AES. The 8-bit data was encrypted with the 8-bit key 0x02

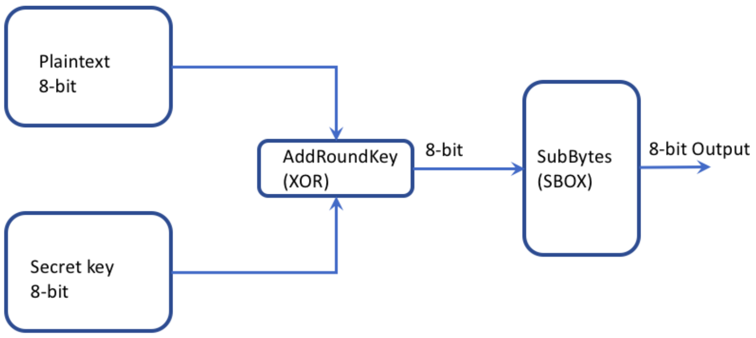


Figure 3.8: Isolated parts of AES-256.

3.5.4 Advenicas IP AES-256

The IP has optimized the computations with the use of parallelism and memory storage. The optimization will create a problem for an attacker since every computation covers the other in noise from neighboring computations. The self-designed simplified version of AES should reduce this type of issue. Advenicas IP is encapsulated by a wrapper to control the input and output, according to Section 3.3.1. The attack will be conducted two times with different trigger encapsulation and one time with a 5 cm loop antenna, at 10 cm directly above the FPGA. In the 1st data acquisition, only a single clock cycle was captured and 20% of the next one. The 2nd data acquisition used an extended timebase to the data capture. With modified trigger signal the capture now contains three cycles instead of one. The reason for extended trigger signal was to capture more data from neighboring computations in the hope of getting better correlation values. The 3rd data acquisition is exactly as the 1st data capture except for the size of the loop antenna and the distance to the FPGA. Making more data acquisitions will yield a better result but requires more time. Three attacks will provide the thesis with enough results to answer the formalized questions.

3.5.5 Computation

Doing computations are not a time-vital action. The attacker can make the statistical computations at other locations than the attacked device. All collected traces are sent as an input into a python script together with the plaintexts, see Appendix A. The python script converts the CSV-files into lists of integers, generate the hypothesis model, create a list of hypothetical electromagnetic emissions, and in the end, it creates a Pearson correlation coefficient between the hypothetical model and the measured traces from the FPGA.
With no average measurement, the attack becomes vulnerable to noise. The noise generated from the FPGA can disturb the analysis and make subkeys produce false signals. The Pearson correlation coefficient method was used to find the correct key. Distance-of-means would create results that would have been harder to analyze, compared to Pearson correlation coefficient. Pearson correlation coefficient is a mathematical stronger method to find linearity between two sets of data samples than distance-of-means. The python script will also collect the highest values of the coefficients, sort them and present them to the user. The correct key is displayed together with its rank among the other guessed keys.

Chapter 4

Results

4.1 Location of EM emissions on the FPGA

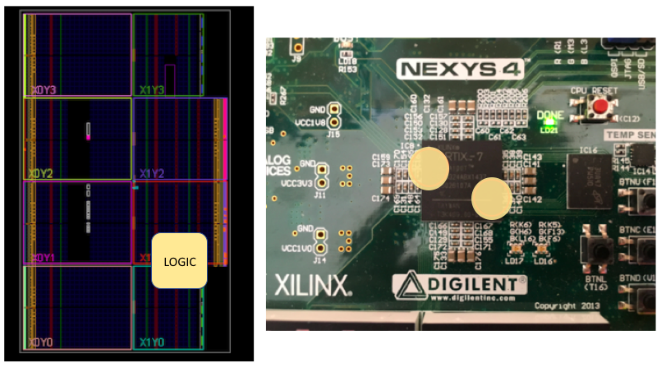


Figure 4.1: Overview of the areas which are sending out electromagnetic emissions.

The area which radiated most electromagnetic emissions can be observed in Figure 4.1. The highlighted area in the left picture is where the actual logic cells are implemented on the FPGA. The left part of Figure 4.1 is a schematic made by Xilinx Vivado; elaborated design. There are few cells spread out along the border of the FPGA. The border cells are I/O-pins. In the right part of Figure 4.1 two highlighted areas of electromagnetic emission can be observed. The results generate questions regarding the second area of EM emissions which have few logic cells. There are no logic cells there except I/O-pins.

4.2 Frequency spectrum of the FPGA

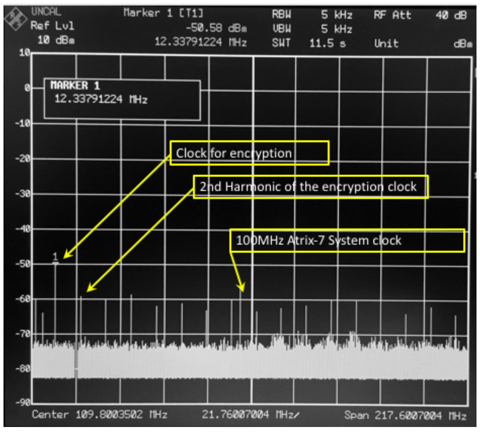


Figure 4.2: The 100 MHz internal clock is divided into a 12.5 MHz clock. The figure presents the frequency spectra of the encryption.

Before the extraction of EM traces, the spectrum analyzer gathered all the present frequency on the FPGA. Figure 4.2 clearly show the internal frequency of 100 MHz and the divided frequency 12.5 MHz. It is not exactly 12.5 MHz. There are many harmonics with +12.5 MHz for every single harmonic. If the 12.5 MHz clock became faster, the data acquisition would suffer because the oscilloscope is not fast enough when switching between state "WAIT" and "TD" (triggered). These two states indicate if the trigger condition is met or not. When the oscilloscope is in WAIT-state, the trigger condition is not met, and no data is collected. When clock becomes faster, the oscilloscope will continuously trigger, because it cannot evaluate the trigger condition correctly.

4.3 Simple electromagnetic analysis

In Figure 4.3, 4.4 and 4.5 the trigger signal with a 14-round AES encryption is presented, plus two cycles from the memory storage and AES-wrapper. The y-axis presents the amplitude in volts. The x-axis presents the sampled data points, and it is not time-dependent. Each sampled data point includes a y-value from the measurement. The screen memory buffer only contains 1200 data points which set the length of the x-axis. When the trigger condition is met, all the data between the rising edge and falling edge are captured and stored in a CSV-file as a string of integers. Depending on where the trigger signal is implemented in the register transfer level (RTL) design, different parts of the EM trace can be captured. As a first stage, all 14 rounds plus memory storage and AES-wrapper cycle are extracted and presented as 16 peaks. The peaks in the beginning and end of the AES-encryption are due to the trigger signal. It is generating noise in the FPGA due to the 3,3-volt output. The frequency between the two peaks is 12.5 MHz, which is the wanted encryption frequency.

Figure 4.3, 4.4 and 4.5 presents a whole encryption with 14 rounds and the trigger pulse. The data in these figures come from the first attack on the AES-256. When the actual DEMA attack occurred, the trigger pulse was changed to encapsulate only the first round.

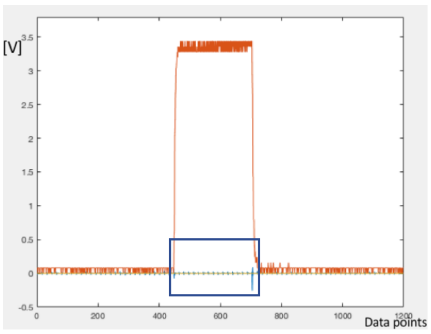


Figure 4.3: The trigger pulse for encryption.

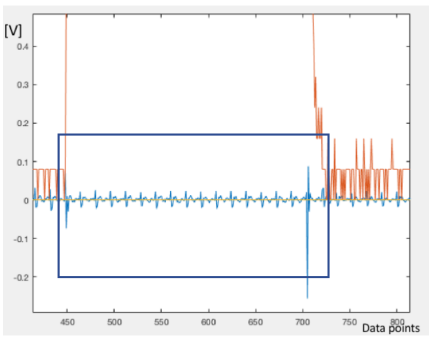


Figure 4.4: A zoomed in version of Figure 4.3.

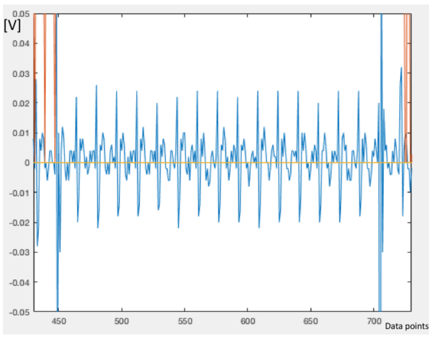


Figure 4.5: A zoomed in version of Figure 4.4. Presenting 14 rounds of AES, plus one round for memory storage, and one round for the AES-wrapper.

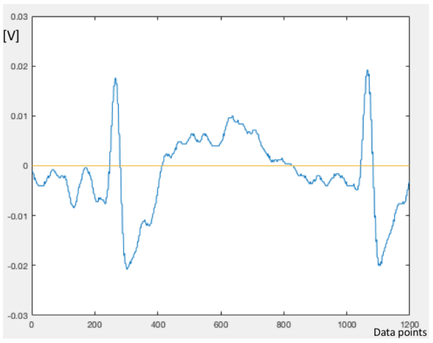


Figure 4.6: One round of AES encryption.

In Figure 4.6 the attack has captured one encryption round. The peaks have a frequency of approx. 12.5 MHz. Between these peaks is the important data that is used to extract the secret key.

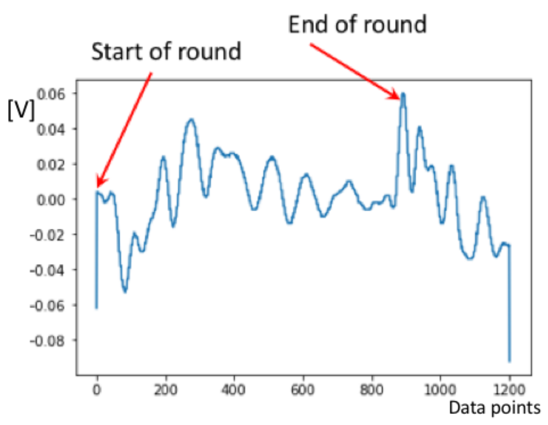


Figure 4.7: Captured EM trace at 10 cm distance with 5 cm loop antenna.

In Figure 4.7 a SEMA, at 10 cm distance, is presented from the first round of AES. This EM trace does not have much visually in common with Figure 4.6. However both represent the same AES round but at different distance from the FPGA.

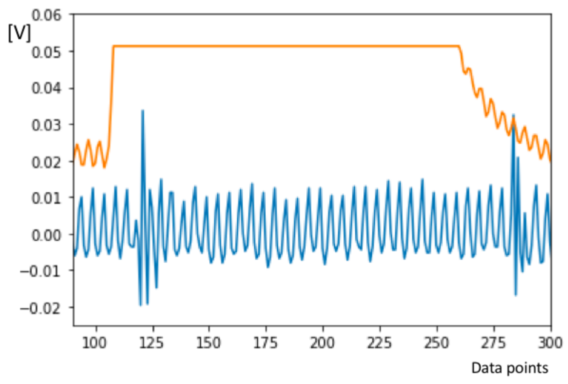


Figure 4.8: Capturing EM traces with a frequency over 50 MHz.

Figure 4.8 presents how the data acquisition is affected if the data is extracted at the same speed as the internal clock of 100 MHz.

4.4 Differential electromagnetic analysis

In Table 4.1, 4.2 and 4.3 the results from three different DEMA on AES-256, are presented. A rank of **1** implies that the key guess is 100% correct. While a rank of 256 implies that our key guess has the worst result compared to the other 255 key guesses.

In Table 4.1 the subkeys 0x03 and 0x0E are 100% extracted. In Figure 4.9 the relative correlation value is presented together with the correlation values for the other keys with 9000 captured EM traces. Figure 4.10 collects the best rank independent of the key value.

Table 4.1: Rank of key guess first data acquisition.

1st data acquisition with number of EM traces and the ranked key.										
Key	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
0x00	231	247	254	174	189	81	55	98	104	192
0x01	177	70	30	10	2	46	32	23	6	3
0x02	41	37	41	14	18	41	28	98	139	123
0x03	171	121	**1**	32	100	13	14	**1**	**1**	**1**
0x04	198	65	56	50	5	16	7	4	5	9
0x05	190	14	23	99	64	70	6	25	139	138
0x06	110	90	100	38	120	104	92	125	73	102
0x07	217	163	191	158	135	230	221	48	204	190
0x08	235	123	104	176	114	239	164	193	206	174
0x09	147	137	171	218	245	219	231	12	250	246
0x0A	163	169	195	67	2	25	82	39	102	38
0x0B	254	240	237	255	200	211	178	73	125	160
0x0C	86	120	74	46	64	11	17	21	9	11
0x0D	112	92	59	45	36	59	57	66	99	60
0x0E	48	**1**	2	2	15	6	36	2	4	5
0x0F	73	56	57	47	49	13	47	81	53	97

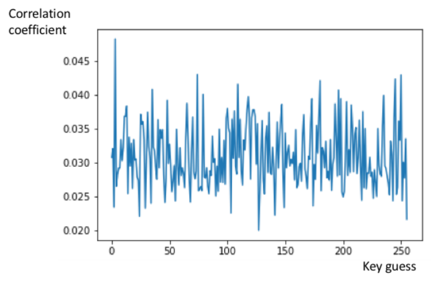


Figure 4.9: Highest correlation coefficient at key guess 0x03.

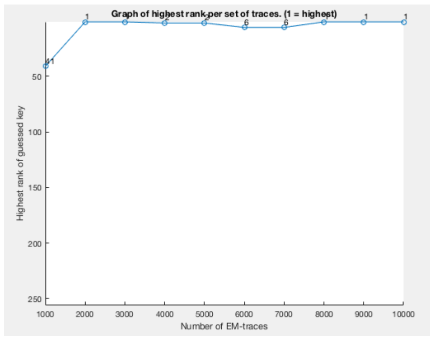


Figure 4.10: Summary of highest rank from Table 4.1.

In Table 4.2 the subkey 0x0D is fully extracted but without any repeating rank. Subkey 0x06 has an overall high rank independent of the number of EM traces. In Figure 4.11 the relative correlation value is presented for subkey 0x0D at 4000 EM traces. Figure 4.12 presents the highest rank for each $n \cdot 1000$ EM traces.

Table 4.2: 2nd data acquisition with extended oscilloscope timebase.

2nd data acquisition with number of EM traces and the ranked key.										
Key	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
0x00	141	249	180	145	241	213	249	244	172	222
0x01	222	212	110	163	117	157	136	213	236	204
0x02	161	151	188	246	250	248	256	253	249	161
0x03	178	244	78	173	19	23	94	163	36	27
0x04	139	40	21	103	50	11	48	26	23	65
0x05	222	74	188	183	192	238	243	235	212	209
0x06	82	51	15	67	30	5	7	8	4	11
0x07	62	128	177	191	162	111	117	109	231	198
0x08	86	192	123	76	66	71	123	92	133	117
0x09	238	244	216	209	209	165	229	231	223	213
0x0A	147	102	223	21	61	113	153	173	233	246
0x0B	6	194	102	75	191	125	64	86	74	137
0x0C	196	126	225	18	23	32	160	229	63	206
0x0D	229	227	161	**1**	12	26	30	39	54	40
0x0E	84	15	77	42	66	25	34	41	55	167
0x0F	223	200	123	174	178	98	95	115	138	93

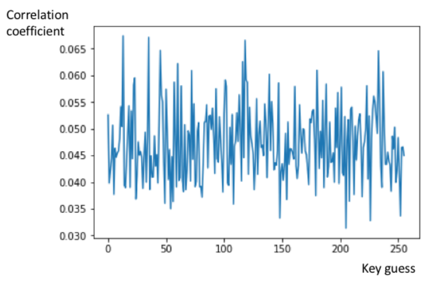


Figure 4.11: Highest correlation coefficient at key guess 0x0D.

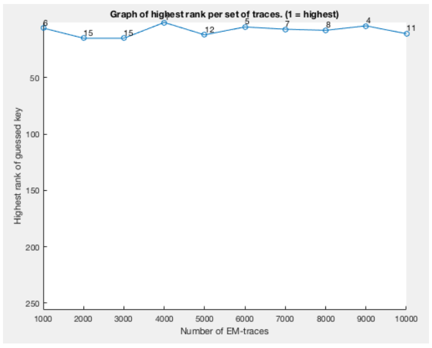


Figure 4.12: Summary of highest rank of key guesses from Table 4.2.

Table 4.3 is the results from the measurements made at 10 cm directly above the FPGA with a 5 cm diameter loop antenna. Subkey 0x05 is fully extracted. The relative correlation value, for all key guyesses at 10000 EM traces, is presented in Figure 4.13. The overall rank for each $n \cdot 1000$ is presented in Figure 4.14.

Table 4.3: Rank of key guess 3rd data acquisition at 10 cm distance.

3rd data acquisition with number of EM traces and the ranked key.										
Key	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
0x00	31	245	253	254	192	254	229	216	135	219
0x01	46	212	49	29	129	168	92	39	59	83
0x02	197	54	95	54	55	72	105	81	85	56
0x03	151	37	25	35	87	38	136	18	61	34
0x04	80	208	199	175	43	37	138	116	115	57
0x05	56	5	33	26	30	31	8	4	2	**1**
0x06	32	55	103	87	143	113	23	78	25	33
0x07	235	161	82	107	65	12	118	134	27	15
0x08	4	10	80	7	56	124	40	183	135	143
0x09	158	95	168	200	217	98	15	21	45	179
0x0A	215	144	65	37	89	65	25	50	69	67
0x0B	209	155	180	125	107	108	163	171	180	228
0x0C	39	27	62	109	22	12	5	11	44	72
0x0D	233	171	194	217	184	217	201	243	244	203
0x0E	39	75	73	223	167	170	56	81	116	145
0x0F	97	18	24	101	231	198	242	249	208	242

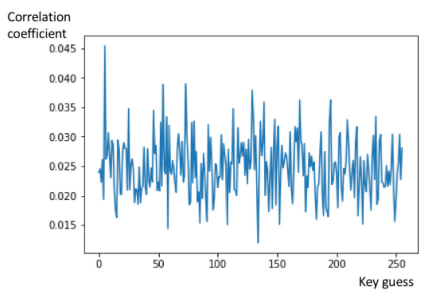


Figure 4.13: Highest correlation coefficient at key guess 0x05.

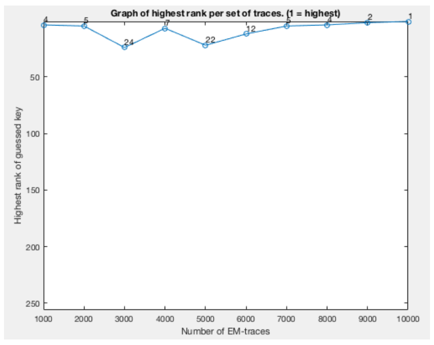


Figure 4.14: Summary of highest rank of key guesses from Table 4.3.

Table 4.4 provides the results from the DEMA of the simplified AES design. Several data acquisitions were made due to power supply interference, but only with one key; 0x02. Figure 4.15 presents the highest-ranked key for every 1000 EM trace.

Table 4.4: Rank of key guess for simplified AES. Implementation according to Figure 3.8.

Data acquisition on simplified AES, with EM traces and the ranked key.										
Key	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
0x02	216	172	55	70	112	50	42	52	38	31

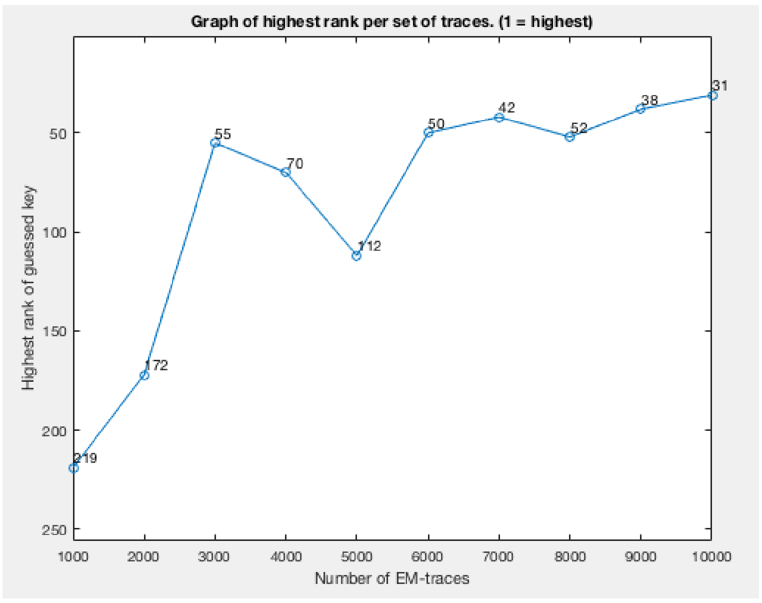


Figure 4.15: The graph presents a summary of highest rank of the key guesses from Table 4.4.

Chapter 5

Discussion

The section is divided into several subsections. First a comment on the results from Section 4, and in the end, discussion regarding the main cause of errors in the results.

5.1 Location of EM emissions on the FPGA

After examining the layout in Figure 4.1 and discussing with employees at Xilinx, it was concluded that the radiation must come from the LEDs wires. There is also a second option; there could be electronics on the other side of the FPGA board that is radiating. The second area of emissions was not investigated further, but instead, the focus was put on the actual encryption logic on the right side of the FPGA. It is important to locate the actual encryption logic to get the correct EM traces and reducing noise. In a real-world scenario, an attacker does not know where on the cryptographic device the encryption core is implemented. Therefore the attacker has to collect EM traces from larger areas which will include all the noise from, e.g., power supply, surface mounted capacitors, resistors, and processors. In this thesis, the implemented design was used as a sheet when comparing the X-Y EM emissions. Mainly to confirm that the correct EM traces were extracted.

5.2 Frequency spectrum of the FPGA

Figure 4.2 presents the frequency spectra of the EM emission from the FPGA. In the AES-wrapper, Section 3.3.1, a clock divider is implemented. The clock divider creates a 12.5 MHz clock. The 12.5 MHz clock can be observed in the spectra with its harmonics. The internal system clock at 100 MHz is also there. No other frequencies are present. The absence of unknown frequencies is good. In this way, the clock divider can be verified, and there is no larger interference taking place at other frequencies. If there are larger amplitudes at non-harmonics signals, then there is a risk that the wrong signals are extracted which generates false EM traces.
The use of a clock divider was vital to the experiment. Due to the equipment, faster extraction of the signals was not possible. If the frequency would have been higher the important data may have been lost.

5.3 Simple electromagnetic analysis

In Figure 4.6 a single round of AES is presented. The waveform of an AES round depends totally on the actual hardware implementation. A repeating pattern is sought after and wanted. By doing a SEMA, it was concluded that the encryption core is presumably using 14 rounds which in turn could imply AES-256 encryption.

Advenica makes their AES IP in such a way that it makes dummy computations while not encrypting anything. The dummy computations can visually hide the actual 14 encryption rounds. Without the trigger-signal, a mathematical algorithm would have to align and cross-correlate all the traces to find the exact start of the encryption. To create an algorithm for aligning all the traces is out of the scope of this thesis. Therefore a trigger signal is used to capture the actual start and end of one encryption round.

5.4 Differential electromagnetic analysis

5.4.1 1st data acquisition AES-256

Succeeding to extract the 128-bit subkey matrix from the first round, will automatically give the first 128 bits of the 256-bit key. According to previous results in Section 1.3 and the theory about Pearson correlation coefficient, a more significant number of EM traces should create a better correlation between the hypothetical values and the captured EM traces. Table 4.1 presents the rank of the key, where the integer **1** is the highest rank. Overall 16 bits of the 256-bit key were successfully extracted. If a rank of 11 and higher is accepted, five subkeys are within the range of brute force technique with reasonable computational time. Because no averaging method was used, the correlation value will be calculated with more signal interferences as input. Using an averaging method generates a better correlation value because the interferences are averaged.
Some rows did not improve regarding rank. The unchanged low rank indicates that the side-channel, related to that key, is very weak. To extract these subkeys, more data points are needed per waveform. The attack is using 1200 data points per waveform. Using more EM traces should also generate a higher correlation value.

In the key interval 0x06-0x0B in Table 4.1, the rank seldom passes below 100. There is also no sign of improvement with a higher number of EM traces. It is unwise to make conclusions regarding the possibilities to extract these keys. Some previous results needed, as mentioned before, millions of traces to retrieve all the subkeys.

Some keys are less complicated to extract than others. Key 0x03 is 100% extracted at 3000 EM traces, and key 0x0E is extracted at 2000 EM traces. Comparing the results from other research on side-channel analysis on a microprocessor, where they could get a correlation around 0.1-0.2, there is a clear difference between this thesis results and previous work. In the results from the thesis, the correlation value lay around 0.01-0.05. The correlation value is a reduction with a factor of 10. Which seems correct according to [1] where they attacked the AES-128 on an ASIC with power analysis. The actual value of the correlation number is not important, it is the relative correlation value compared to all other key guesses that is important. It takes around 70 minutes to collect 10000 EM traces. The amount of time may vary depending on the used equipment for extracting the EM traces. The rank, in some rows, get worse compared to the previous result which might be due to noise interference in the rest of the EM traces. Some rows got an overall high rank, e.g., 0x04 in Table 4.1 which will provide us with an excellent opportunity to brute force the last bits.

In the experiments, the correct key was used to validate the results. In a real-world application, the attacker does not have this opportunity. The guessed keys, which do not keep their high rank when the number of EM traces increases, will be excluded. Continuously high rank leaves us with only one key 100% extracted; 0x03. If the limit for accepted keys are extended, there is also a repeating key at rank 2; 0x0E.

Figure 4.10 presents how the method at least present us with high ranking keys for every thousand EM traces. In Figure 4.9 there is a clear peak at the correct key. The correlation is around 0.048 where maximum value can be 1.0. The correlation value is very low and sensitive to different kind interference. The H-probe need to be still; otherwise, different amplitude on our signals will be acquired due to the distance to the FPGA.

According to [4], there is a dependency on the actual position of the subkey and not its actual value. Some keys are harder to extract basically because of the circuits and where they are used in the encryption algorithm. The subkeys get a bad signal-to-noise ratio (SNR) value which make the extraction harder to conduct. However, in the thesis experiments, some keys gave a good correlation value in the 1st data acquisition and a worse correlation value in the 2nd data acquisition. Even if they were at the same key position.
There is no definite answer on how many electromagnetic waveforms that are needed to extract the subkeys. It all depends on the circumstances and the environment.

5.4.2 2nd data acquisition AES-256 with extended timebase

The second acquisition involves an extended time interval for acquiring data. This is done by changing the timebase of the oscilloscope. The density of the sampled data points is always 1200 for screen memory buffer on the oscilloscope. When two more cycles were included from the encryption run, a broader interval of data points was extracted. The disadvantage of this method is the loss of sampled points per encryption cycle. Instead of acquiring 1200 per cycle, the oscilloscope captured 400 points per round. The wider interval approach aimed to get a better correlation, but instead, the overall ranking of the subkey guesses became worse. Increasing the number of sampled points, will generate a better chance of getting a higher rank on the key guess. With the comparison between first data acquisition and second data acquisition, it was concluded that the number of points per EM trace is vital and directly impact the correlation coefficient.

In Table 4.2 there are only one key extracted 100%; 0x0D. However, applying this table to a real-world application, the attacker would never know it is 100% correct key guess. When compared with neighboring ranked keys it can be observed that it is not a key guess that will keep its rank when EM traces increases. As mentioned before; the attacker is looking for a key that keeps repeating its high rank. If the top five key guesses are presented for every thousand EM traces (sorted by there correlation coefficient), there might be a repeating pattern among the key guesses. The repeating pattern will present the attacker with good opportunities to brute force the correct key.

Figure 4.11 presents the key guess with its correlation value. However, it has not a significantly high correlation value compared to other surrounding peaks. The attacker is looking for a single relative high correlation value, like the one in Figure 4.9. There is no definition of a relatively good correlation value. However, many correlation peaks indicate that the side-channel is not strong for the correct key. The side-channel is strong enough to get the correct key, but there are other key guesses with a nearby correlation value which might provide the attacker with wrong key guess when number of EM traces increases.

5.4.3 3rd data acquisition AES-256 at distance of 10 cm

Table 4.3 presents the key 0x05 fully extracted. According to [1]; an increasing number of data traces should generate an increasing rank of the key. There is only a clear result on key 0x05. However, at a distance of 10 cm, it is still possible to extract subkeys with relative high correlation coefficient, see Figure 4.13. If it is possible to extract subkeys at a distance, with included noise from the power supply, resistors, and capacitors, new opportunities are created for the attacker. In this attack, the diameter of the loop antenna was increased to 5 cm. The EM trace in Figure 4.7 is visually very different from the first data acquisition in Figure 4.6 but still the program was able to find a relatively high correlation and extract one subkey. To confirm that it was the correct key, more EM traces are needed. As written before; the attacker needs a repeating pattern of ranked keys to confirm that one has the correct key. With only Table 4.3 an attacker will not be able to confirm that 0x05 is the correct key due to the lack of repeating subkey pattern.

5.4.4 Data acquisition from simplified AES

The simplified version of AES should provide results of how an attacker can extract the unknown parameter k in the function $f(d, k)$. Since it was less complicated in its design, the EM emissions should have a reduced number of interfering signals. This attack was not as successful as the attack on AES-256. There could be several reasons for this result. It could be the power supply interference making the trigger signal go out of sync with the plaintexts from the FPGA. When the FPGA was programmed to encrypt 10000 plaintexts, the LabView program only registered around 6000 encryptions which imply 4000 EM traces were not registered. The FPGA sends an "encryption done" signal to an LED on the FPGA board. Because of the lost EM traces, the test had to be retaken several times. However, there is still an improvement in the ranking system in Figure 4.15. With more traces and sample points there should be a better result.

5.5 Sources of errors

According to [1] it is a complex task to extract secret keys from hardware encryption. The complexity is primarily due to the parallelism of operations and the wiring on the SoC. Different length of wires generates various electromagnetic fields. When extracting the partial secret key, it is often needed to collect at least 100000 traces. Sometimes even up to millions of traces are needed [5]. The number of EM traces varies a lot depending on experiment setup in different research. In other research, they have used between 10000 EM traces and up to 3 million EM traces. In this thesis, the number of traces was at maximum 10000.

Based upon the previous results in Section 1.3 in [4] where they have been using around 7000 EM traces to extract the 120-bit of the 128-bit, the results in this thesis did not deliver the same success. Their results came from an SCA-FPGA-board with an oscilloscope that cost x15 times more than the used oscilloscope in this experiment. Moreover, they used an amplifier that has a wider bandwidth. On the other hand, there are examples of master thesis [13] where they never succeed to extract any subkeys at all. They only reached the rank of 2153 at best when attacking single-cycle SHA-256, two bytes, on a low power FPGA platform. They also had 143000 EM traces compared with this thesis 10000 EM traces. [13] was made on the SASEBO-GII. They used a different encryption algorithm, but the theory and method is still the same.

One way to reduce the noise and interference from the switching transistors is to lower the encryption frequency. Some of the previous results were clocked at 3 kHz. Using 3kHz as encryption frequency is not possible in this thesis due to the bandwidth of the amplifier (10MHz-1GHz). When encrypting at such a low frequency, a vast majority of the switching interference from the transistors will be drastically reduced. If a filter is added high-frequency noise can be excluded.

In some of the measurements, weird interference can be observed. This noise behavior came from the power supply jack from all the rooms on this side of the corridor. When someone was plugging in a charger, e.g., for cell phone or computer this phenomenon occurred, see Figure 5.1.

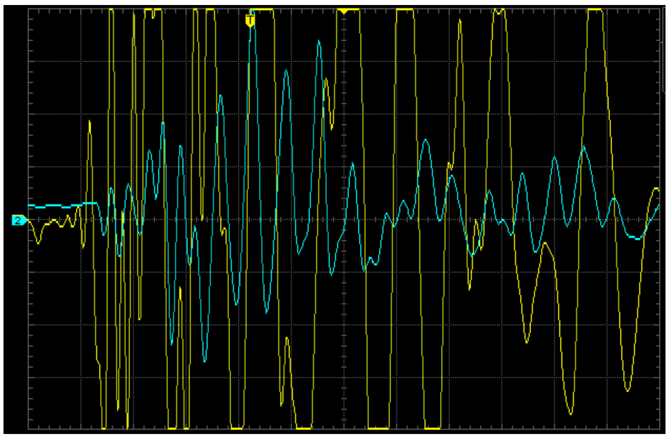


Figure 5.1: Waveforms when charger is plugged into the 230V jack in neighboring rooms or the lab room. The saturated (yellow) signal is the actual EM waveform. The signal (blue) with low amplitude, and which is alternating close around the x-axis, is the trigger signal.

This interference corrupted the data acquisition so that the next EM traces t_i where not aligned to their data block d. When this interference occurred, the data acquisition was stopped and reset. Taking a more substantial number of measurements will increase the risk of including power supply disturbance which in return made test runs corrupt. While making measurements, it is a necessity to stay present all the time to exclude the risk of interference in the data acquisition. By visually inspecting the data capture on the oscilloscope the power interference could be observed. To counter this interference a filter could be applied.

One thing worth mentioning is the temperature in the FPGA chip will affect the EM emissions. The effect of the temperature is not considered that in this thesis.

Chapter 6

Conclusions

In this thesis, one 8-bit subkey was 100% extracted at 0 cm distance from the FPGA. Including higher ranked keys, from Table 4.1, five subkeys were extracted from 10000 EM traces. At 10 cm from the FPGA, one subkey was extracted. It took around 70 minutes, without averaging, to extract the EM traces. To brute force a 256-bit key requires testing $2^{256} = 1,1579 \cdot 10^{77}$ different combinations. AES-256 would require a multiple of the lifetime of our universe, to brute force with modern digital computers. Brute forcing the first 128-bit would need to $2^{128} = 3,4028 \cdot 10^{38}$ different combinations.

The results, with high ranked keys, would give us $2^{88} = 3,0948 \cdot 10^{26}$ different combinations to try for 128-bit key encryption. With new technology in the form of a quantum computer and the promised computational power, even AES-256 is not safe anymore. A full AES-256 is considered post-quantum secure with present technology. By combining side-channel attacks with a quantum computer, it is possible to lower the number of bits to brute force the last non-extracted bits. However, more time is needed to extract all the subkeys with current equipment.

The sources of threats, within the cyberspace, can be divided into three different categories; individuals, organizations, and nations. In this thesis, the provided results confirm that it is doable, as an individual, with low-cost equipment, to conduct a side-channel attack on a post-quantum secure encryption algorithm. Not all subkeys were extracted, but the method and apparatus proved to be able to retrieve subkeys from an FPGA that is not designed for SCA. The measurement is sensitive to different interferences, but with an increased number of sampled points and a filter, the results would be better. The problem in this experiment was the simplified design of the first round in AES, see Section 3.5.3. For a none confirmed reason, the same results from AES256 could not be generated on the simplified version of AES. The simplified AES was designed to match the EM model Hamming distance. According to the theory, an attacker should be able to extract the subkeys much faster than AES-256 since the simplified AES do not have any additional logic.

6.1 Future work

Continuing and improving the experiment for the future, the following subjects need to be addressed:

6.1.1 Number of data points

In this experiment, 1200 data points were used for each EM trace. According to previous work in Section 1.3, and the results in Section 4, it was concluded that more data points are needed to create better correlation coefficients.

6.1.2 Oscilloscope

The oscilloscope used a USB to transfer data which limits the data throughput. To capture more data, a faster interface is needed between the oscilloscope and the PC. A more upgraded oscilloscope can also have a larger memory screen buffer, e.g., 3000 data points.

6.1.3 Number of EM traces

The experiment did not extend beyond 10000 EM traces. Extracting EM traces faster, should increase the total number of captured EM traces and still stay below 70 minutes for data extraction.

6.1.4 Amplifier

The amplifier had a bandwidth of 10 MHz to 1 GHz. An amplifier with a lower interval could lower the encryption frequency and in return reduce the switching noise from the transistors.

6.1.5 Low-pass filter

Using a low-pass filter could reduce the noise generated from higher frequencies, e.g., transients and other power quality reducing effects.

6.1.6 Power supply filter

A filter could be used to reduce the received power quality interference from neighboring rooms e.g., common and differential mode filter.

6.1.7 EM trace alignment software

In the experiment, a trigger signal was used to mark where the encryption starts and where it ends. In a real-world application, the attacker does not have this trigger signal support. One would need a software tool that aligns the EM traces.

6.1.8 Distance

The measurements were made at 0 cm and 10 cm. Further investigations are needed regarding the possibilities to extract subkeys at greater distances.

6.1.9 The last 128 key bits

The thesis only included an attack on the first 128 key bits in AES-256. To completely extract all the 32 subkeys, the attack needs to be modified for the last 128 bits.

Bibliography

[1] Elisabeth Oswald Stefan Mangard and Thomas Popp. *Power Analysis Attacks. Revealing the Secrets of Smart Cards*. Springer, 2007.

[2] *Defending against side-channel attacks*. `https://www.eetimes.com/document.asp?doc_id=1279920`. Accessed: 2018-02-14.

[3] *Advenica*. `https://advenica.com/en/certifications-approvals`. Accessed: 2018-02-14.

[4] A.Sasaki Y.Hori T.Katashita and Akashi Satoh. "Electromagnetic Side-channel Attack against 28-nm FPGA Device". In: *2012 IEEE 1st Global Conference on Consumer Electronics* 32.3 (Dec. 2012), pp. 657–660. ISSN: 2378-8143. DOI: `10.1109/GCCE.2012.6379944`. URL: `http://dx.doi.org/10.1109/35.267438`.

[5] Cheuk Wong. "Analysis of DPA and DEMA Attacks". MA thesis. San Jose State University, 2012.

[6] Elke De Mulder. "Electromagnetic Techniques and Probes for Side-Channel Analysis on Cryptographic Devices". PhD thesis. Katholieke Universiteit Leuven - Faculty of Engineering, 2010.

[7] Ingrid M.R. Verbauwhede (Franc¸ois-Xavier Standaert). *Secure Integrated Circuits and Systems, Chapter: Introduction to Side-Channel Attacks*. Springer US, 2010.

[8] National Institute of Standards and Technology. *FIPS PUB 197: Advanced Encryption Standard (AES)*. 2001. URL: `https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197.pdf`.

[9] Massoud Masoumi Mehdi Masoomi and Mahmoud Ahmadian. "A practical differential power analysis attack against an FPGA implementation of AES cryptosystem". In: *2010 International Conference on Information Society (i-Society)* (2010), pp. 308–312.

[10] Elisabeth Oswald Colin D. WalterÇetin K. KoçChristof Paar (Eds.) (Siddika Berna Örs and Bart Preneel). *Cryptographic Hardware and Embedded Systems - CHES 2003. Chapter: Power-Analysis Attacks on an FPGA – First Experimental Results.* Springer-Verlag Berlin Heidelberg, 2003.

[11] Weiwei Shan Jie Li and Chaoxuan Tian. "Hamming Distance Model based Power Analysis for Cryptographic Algorithms". In: *Applied Mechanics and Materials* 121-126 (Oct. 2011), pp. 867–871. ISSN: 1662-7482. DOI: `10.4028/www.scientific.net/AMM.121-126.867`.

[12] Marc Joye, Christophe Clavier Jean-Jacques Quisquater (Eds.) (Eric Brier, and Francis Olivier). *Cryptographic Hardware and Embedded Systems - CHES 2004. Chapter: Correlation Power Analysis with a Leakage Model.* Springer Berlin Heidelberg New York, 2004.

[13] Mustafa Faraj. "Side Channel Attack on Low PowerFPGA Platform". MA thesis. University of Waterloo,Ontario Canada, 2016.

Appendix A

Python

```
import numpy as np
import csv
import re
from scipy.stats.stats import pearsonr
import matplotlib.pyplot as plt
from operator import itemgetter
from tempfile import TemporaryFile



sbox = [
0x63,0x7c,0x77,0x7b,0xf2,0x6b,0x6f,0xc5,
0x30,0x01,0x67,0x2b,0xfe,0xd7,0xab,0x76,
0xca,0x82,0xc9,0x7d,0xfa,0x59,0x47,0xf0,
0xad,0xd4,0xa2,0xaf,0x9c,0xa4,0x72,0xc0,
0xb7,0xfd,0x93,0x26,0x36,0x3f,0xf7,0xcc,
0x34,0xa5,0xe5,0xf1,0x71,0xd8,0x31,0x15,
0x04,0xc7,0x23,0xc3,0x18,0x96,0x05,0x9a,
0x07,0x12,0x80,0xe2,0xeb,0x27,0xb2,0x75,
0x09,0x83,0x2c,0x1a,0x1b,0x6e,0x5a,0xa0,
0x52,0x3b,0xd6,0xb3,0x29,0xe3,0x2f,0x84,
0x53,0xd1,0x00,0xed,0x20,0xfc,0xb1,0x5b,
0x6a,0xcb,0xbe,0x39,0x4a,0x4c,0x58,0xcf,
0xd0,0xef,0xaa,0xfb,0x43,0x4d,0x33,0x85,
0x45,0xf9,0x02,0x7f,0x50,0x3c,0x9f,0xa8,
0x51,0xa3,0x40,0x8f,0x92,0x9d,0x38,0xf5,
0xbc,0xb6,0xda,0x21,0x10,0xff,0xf3,0xd2,
0xcd,0x0c,0x13,0xec,0x5f,0x97,0x44,0x17,
0xc4,0xa7,0x7e,0x3d,0x64,0x5d,0x19,0x73,
0x60,0x81,0x4f,0xdc,0x22,0x2a,0x90,0x88,
0x46,0xee,0xb8,0x14,0xde,0x5e,0x0b,0xdb,
0xe0,0x32,0x3a,0x0a,0x49,0x06,0x24,0x5c,
0xc2,0xd3,0xac,0x62,0x91,0x95,0xe4,0x79,
0xe7,0xc8,0x37,0x6d,0x8d,0xd5,0x4e,0xa9,
0x6c,0x56,0xf4,0xea,0x65,0x7a,0xae,0x08,
0xba,0x78,0x25,0x2e,0x1c,0xa6,0xb4,0xc6,
0xe8,0xdd,0x74,0x1f,0x4b,0xbd,0x8b,0x8a,
0x70,0x3e,0xb5,0x66,0x48,0x03,0xf6,0x0e,
0x61,0x35,0x57,0xb9,0x86,0xc1,0x1d,0x9e,
0xe1,0xf8,0x98,0x11,0x69,0xd9,0x8e,0x94,
0x9b,0x1e,0x87,0xe9,0xce,0x55,0x28,0xdf,
0x8c,0xa1,0x89,0x0d,0xbf,0xe6,0x42,0x68,
0x41,0x99,0x2d,0x0f,0xb0,0x54,0xbb,0x16
        ]

#-----------------------------------------------

def HYPO_AddRoundKey(key_byte,text_byte):

    XORed = bin(key_byte ^ int(text_byte,2))
    return XORed
#-----------------------------------------------
def HYPO_SubBytes(AES_byte):
    AES_byte = sbox[int(AES_byte,2)]
    return AES_byte
#-----------------------------------------------
def HYPO_HammingDistance(text_byte,AES_byte):
    count = 0
    HD_byte = bin(int(text_byte,2) ^ AES_byte)
    HD_byte.replace("0b","")
    for i in range(len(HD_byte)):
            if HD_byte[i] == '1':
                count+=1
    return count


#------------------------------------------------
VHDL_plaintexts = open("PLAINTEXTS.txt","r")

all_plaintexts = []

nbr_of_traces = 4000


for line in VHDL_plaintexts:
    all_plaintexts.append(line)

for l in range(16):
# Loop through key position from 0x00-0x0F. First 128-bits
    H = []
    for i in range(nbr_of_traces):
    #Loop through every 8 bits of all_plaintexts
        input_byte = all_plaintexts[i][(l*8):(l*8+8)]
        HD_row = []
        for k in range(256):
        # Create key hypotheses for every possible key 0-255
            #-------- Cryptographic algorithm------
            #Send in every key hypotheses and produce an output V
            XORed = HYPO_AddRoundKey(k,input_byte)
            SBOX_value = HYPO_SubBytes(XORed)
            #SBOX_value is the Hypothetical
            #Intermedia Value (V).
            HD = HYPO_HammingDistance(input_byte,SBOX_value)
            # HYPO_HammingDistance is the "POWER MODEL"
            # in the flow chart.
            HD_row.append(HD)
            # Row of hypothetical EM-points. h(1,K),

        H.append(HD_row)#Hypothetical EM-traces, size DxK
        #Add rows to an array to create a 2D-array (Matrix)




#------------CREATE T-matrix of measured values-------
    T_row = []
    T_column = []
    H_column = []


    with open("10000_2.csv") as csvfile2:
        readCSV = csv.reader(csvfile2, delimiter='␣')
        for row in readCSV:
            T_row.append(row[0])
            #Fetch all the values from CSV-file

    for i in range(nbr_of_traces):
        T_row[i] = T_row[i].replace(",",".")
        T_row[i] = re.split(r'\t+',T_row[i])

         #Create a row in T-matrix with length 1200 data points
        for k in range(1200):
            T_row[i][k] = float(T_row[i][k])
         #Measured values are written in strings so we
         #need to convert them into floats.

    for i in range(1200):
        #For every data point in every EM-trace add them
        #in a new array to make it easier to compute
        # Pearson correlation Coefficient.
        T_temp = []
        for k in range(nbr_of_traces):
            T_temp.append(T_row[k][i])

        T_column.append(T_temp)
        #Now we have Colums of data from T-matrix
        #in rows.


    for i in range(256):
        H_column_temp = []
        for k in range(nbr_of_traces):
           H_column_temp.append(H[k][i])

        H_column.append(H_column_temp)

#------CREATE Result-matrix. The last step in the flow-chart-------
        #-------------Statistical Analysis-----------
    r_matrix = []
    r_temp_matrix= []
    #Loop through every column of Matrix T and compare them with columns
    #in H. Create a correlation value "r" for every T_column vs H_column.
    for i in range(256):
        r_temp_matrix= []
        for k in range(1200):
            r = pearsonr(T_column[k], H_column[i])[0]
            r_temp_matrix.append(abs(r))
            #We dont care if there is a negative linear relationship.
            #Only if there is a linear relationship at all.
        r_matrix.append((r_temp_matrix))

    #---Create an array with maximum r-values to rank the key guess.
    max_r_matrix_tuple =[]
    max_r_matrix =[]
    for i in range(len(r_matrix)):
        max_r_matrix_tuple.append((max(r_matrix[i]),i))
        max_r_matrix.append(max(r_matrix[i]))

    max_r_matrix_tuple.sort(key=itemgetter(0),reverse=True)

    print("What correlation had the correct key among the key guesses:")
    print([item for item in max_r_matrix_tuple if item[1] == 1])
    #Change last digit to search for key
    indices = [i for i,tu in enumerate(max_r_matrix_tuple) if tu[1]==1]
    #Change last digit to search for key
    print("What was the rank of the correct key among our key-guesses"
         ,indices)
    plt.plot(max_r_matrix)

    csvfile2.close()
VHDL_plaintexts.close()
```

Appendix B

VHDL

B.1 AES-wrapper

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
library std;
use std.textio.all;
use ieee.std_logic_textio.all;


entity AES_Wrapper is
  Port (

                rst  : in std_logic;
                init_start  : in std_logic;
    -- Click on button to start the N encryption runs
                fast_clk  : in std_logic;
                trigger  : out std_logic;
                key_done_LED  : out std_logic;
                cipher_wrapper_OUT  :out unsigned(7 downto 0)
    -- Vivado optimize the code and remove the ciphertext signals.
    -- Therefore we force it to keep them by giving them as output.

  );
end AES_Wrapper;

architecture rtl of AES_Wrapper is
    type state_type is (TRIGGER_HOLD,BUSY_AES,IDLE_AES,HOLD_AES,
                         START_AES_128,START_AES_256,
                         SEND_HOLD_DATA,SEND_DATA);
    signal state,next_state : state_type;
```

```
component aes_256_top
-- Advenicas AES-256 IP. The code for this IP will not be included.
    port(
        sys_rst  : in std_logic;
        sys_clk  : in std_logic;
        clear    : in std_logic;
        new_key   : in  std_logic;
        key       : in  unsigned(127 downto 0);
        key_done  : out std_logic;
        start   : in  std_logic;
        plain   : in  unsigned(127 downto 0);
        cipher  : out unsigned(127 downto 0);
        busy    : out std_logic;
        valid   : out std_logic

    );
end component;

component LFSR -- Used to generate psudo-random plaintexts.
    port(         --We could use a counter but that would not be realistic.
        reset  : in std_logic;
        clk  : in std_logic;
        enable  : in std_logic;
        q   : out std_logic_vector(127 downto 0)
        -- 128 bits of plaintext. Sent as an input to aes_256_top

    );
end component;


signal busy_wrapper, valid_wrapper, key_done_wrapper,
new_key_wrapper : std_logic;

signal cipher_wrapper,plain_wrapper : unsigned(127 downto 0);
signal q_plain_text : std_logic_vector(127 downto 0);
signal cipher_q,cipher_next : unsigned(127 downto 0);
signal key_256: unsigned(255 downto 0); -- Complete key of 256 bits.
signal key_128: unsigned(127 downto 0);
-- The 256-bit key is divided into two 128-bits and sent into aes_256_top

signal init_start_q: std_logic;
-- Start signal generated from the init_start from the FPGA-board

signal clear_wrapper: std_logic:='0';
--Signal to clear the memories. Not used in the experiment.

signal counter: unsigned (1 downto 0):="00";
```

```
-- Counter is used to create a slow clock.
signal clk : std_logic := '0';    -- Slow clock
signal enc_counter: unsigned (13 downto 0):="00000000000000";
--Set the number of encryption to run by adjusting the binaries.

signal round_counter: unsigned(3 downto 0) :="0000";
-- Used to set the trigger at correct time intervall.

signal SEL: unsigned(3 downto 0):="0000";
-- A selector in a multiplexer for cipher_wrapper_OUT

signal enable: std_logic; -- Control the LFSR
signal trigger_count: unsigned(21 downto 0)
:="0000000000000000000000";
-- Delay time after the trigger pulse, to give the
-- oscillscope time to reconfigure for the next EM-trace.

--file file_RESULTS : text;
-- Used in simulations to log the plaintexts from LFSR.

begin
aes_256_top_COMPONENT:aes_256_top
    port map (
      sys_rst    => rst,
      sys_clk    => clk,
          key => key_128,
          new_key => new_key_wrapper,
          key_done => key_done_wrapper,
          plain => unsigned(q_plain_text),
          busy => busy_wrapper,
          valid => valid_wrapper,
          start => init_start_q,
          clear => clear_wrapper,
      cipher => cipher_wrapper
      );

random_text:LFSR
    port map (
    reset => rst,
    clk => clk,
    enable => enable,
    q => q_plain_text
    );

clock_divider:process(fast_clk,rst)
begin
```

```
            if(rst='1') then

            elsif(fast_clk'event and fast_clk ='1') then
              counter<=counter +1;


              if(counter = "11") then
                clk <= not clk;
                counter<=(others =>'0');
              end if;

            end if;

end process;


sequential: process(clk,rst)
 -- variable plaintext     : line;
 begin
           if(rst='1') then
          --On reset set the initial state
          --for the FSM and other initial values.
                state <= BUSY_AES;
                plain_wrapper<=(others =>'0');

                SEL<="0000";
                trigger <='0';
             --  file_open(file_RESULTS,
             --  "C:\Users\oskar\Desktop\PLAINTEXTS.txt", write_mode);
            elsif(clk'event and clk ='1') then

                if(state=TRIGGER_HOLD) then
                -- State where a counter delay the
                -- amount of time after a trigger pulse.
                    trigger_count<=trigger_count+1;

                elsif(valid_wrapper = '1' ) then
                -- When an encryption is done the plaintext
                --register is loaded with new data.
                    SEL<=SEL+1;
                    plain_wrapper<=unsigned(q_plain_text);
                    -- Load new plaintext from LFSR
                   -- write(plaintext, q_plain_text);
                   -- writeline(file_RESULTS, plaintext);
                    enc_counter<=enc_counter+1;
                    -- Increment the encryption counter.
                    elsif(next_state=SEND_DATA and
                          round_counter="0000") then
                   -- An operation to force the trigger to occur at a
                   -- specific point in time.
                   -- Send a trigger signal to oscilloscope

                          round_counter<=round_counter+1;
                          trigger <='1';

                    elsif(next_state=SEND_DATA
                   and round_counter/="0000") then
                              round_counter<=round_counter+1;
                              trigger <='0';

                    else
                    trigger <='0';
                   end if;

                     cipher_q<=cipher_next;
                     state<= next_state;

              end if;

 end process;


 key_done_LED<=key_done_wrapper;


 combinational: process(enc_counter,trigger_count,
 cipher_wrapper,cipher_q,busy_wrapper,init_start,s
 tate,key_done_wrapper,valid_wrapper,key_256)
 begin
    --default values
    key_256<=X"000102030405060708090a0b
0c0d0e0f101112131415161718191a1b1c1d1e1f";
    -- Encryption key for the AES-256

    next_state<=state;
    cipher_next<=cipher_q;
    init_start_q<='0';
    enable<='0';
    new_key_wrapper<='0';
    key_128<=(others =>'0');

 case (state) is
    when BUSY_AES => if (busy_wrapper ='0') then
```

```
                      next_state<= IDLE_AES;
                      end if;

when IDLE_AES => if (init_start ='1') then
                     next_state<=HOLD_AES;
                     end if;

when HOLD_AES => if (init_start='0') then
                      next_state<=START_AES_128;
                      end if;

when START_AES_128 =>if(busy_wrapper='0')then
                       new_key_wrapper <='1';
                       key_128<=key_256(255 downto 128);
                       next_state<=START_AES_256;
                       end if;


when START_AES_256 => if(busy_wrapper='1') then
                           key_128<=key_256(127 downto 0);
                           new_key_wrapper<='0';
                           next_state<= SEND_HOLD_DATA;
                           end if;

when SEND_HOLD_DATA => if(key_done_wrapper ='1') then
                       init_start_q<='1';
                       next_state<=SEND_DATA;
                       enable<='1';
                       end if;

when SEND_DATA =>if(valid_wrapper = '1') then
                              cipher_next<=cipher_wrapper(127 downto 0);
                              next_state<=TRIGGER_HOLD;
                              enable<='0';
                     end if;


 when TRIGGER_HOLD => if(enc_counter="11111111111111") then
                              next_state<=IDLE_AES;

                           else
                                 if(trigger_count="1111111111111111111111"
                                 then
                                 next_state<=SEND_HOLD_DATA;
```

```
                                end if;
                            end if;

    end case;

    end process;



cipher_OUTPUT: process(SEL, cipher_wrapper)
    begin
    case SEL is
        when "0000" => cipher_wrapper_OUT <=
        cipher_wrapper(127 downto 120);

        when "0001" => cipher_wrapper_OUT <=
        cipher_wrapper(119 downto 112);

        when "0010" => cipher_wrapper_OUT <=
        cipher_wrapper(111 downto 104);

        when "0011" => cipher_wrapper_OUT <=
        cipher_wrapper(103 downto 96);

        when "0100" => cipher_wrapper_OUT <=
        cipher_wrapper(95 downto 88);

        when "0101" => cipher_wrapper_OUT <=
        cipher_wrapper(87 downto 80);

        when "0110" => cipher_wrapper_OUT <=
        cipher_wrapper(79 downto 72);

        when "0111" => cipher_wrapper_OUT <=
        cipher_wrapper(71 downto 64);

        when "1000" => cipher_wrapper_OUT <=
        cipher_wrapper(63 downto 56);

        when "1001" => cipher_wrapper_OUT <=
        cipher_wrapper(55 downto 48);

        when "1010" => cipher_wrapper_OUT <=
        cipher_wrapper(47 downto 40);

        when "1011" => cipher_wrapper_OUT <=
        cipher_wrapper(39 downto 32);
        when "1100" => cipher_wrapper_OUT <=
        cipher_wrapper(31 downto 24);

        when "1101" => cipher_wrapper_OUT <=
        cipher_wrapper(23 downto 16);

        when "1110" => cipher_wrapper_OUT <=
        cipher_wrapper(15 downto 8);

        when "1111" => cipher_wrapper_OUT <=
        cipher_wrapper(7 downto 0);
        when others =>
        end case;
         end process;
end rtl;
```

B.2 Simplified AES

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.all;
library std;
use std.textio.all;
use ieee.std_logic_textio.all;


entity top_simple_aes is
    Port ( start : in STD_LOGIC;
           reset : in STD_LOGIC;
           fast_clk : in STD_LOGIC;
           sbox_out : out std_logic_vector(7 downto 0);
           trigger : out std_logic
           );
end top_simple_aes;

architecture Behavioral of top_simple_aes is
type state_type is(IDLE_AES,HOLD_AES,
SEND_HOLD_DATA,XOR_SBOX,TRIGGER_HOLD);

signal state,next_state : state_type;


signal enable,clk: std_logic :='0';
signal q_plain_text: std_logic_vector(127 downto 0);
signal sbox_input: std_logic_vector(7 downto 0);
signal counter: unsigned(1 downto 0);
--Clock divider. Make the 100MHz a 12,5MHz clock.

signal enc_counter: unsigned (13 downto 0):="00000000000000";
signal trigger_count: unsigned(21 downto 0)
:="0000000000000000000000";

--file file_RESULTS : text;

component LFSR
   port(
      reset : in std_logic;
      clk : in std_logic;
      enable : in std_logic;
      q  : out std_logic_vector(127 downto 0)

    );
```

```
end component;

component sbox
     port(
     clk: in std_logic;
     rst: in std_logic;
     bytein: in std_logic_vector(7 downto 0);
     byteout: out std_logic_vector(7 downto 0)
     );

     end component;
begin

random_text:LFSR
     port map (
     reset => reset ,
     clk => clk ,
     enable => enable ,
     q => q_plain_text
     );

sbox_component: sbox
     port map (
     clk => clk ,
     rst => reset ,
     bytein => sbox_input ,
     byteout => sbox_out

     );

clock_divider:process(fast_clk ,reset)
begin

               if(reset='1') then
               counter<="00";


               elsif(fast_clk'event and fast_clk ='1') then
                 counter<=counter +1;


                 if(counter = "11") then
                   clk <= not clk;
                   counter<=(others =>'0');
                 end if;

               end if;
end process;


sequential: process(clk,reset)
 variable plaintext      : line;
 begin
             if(reset='1') then
             trigger <='0';
             state<=IDLE_AES;


             elsif(clk'event and clk ='1') then
                 if(next_state=XOR_SBOX) then
                     enc_counter<=enc_counter+1;
                     trigger <='1';
                 elsif(state=TRIGGER_HOLD) then
                      trigger_count<=trigger_count+1;
                      trigger <='0';
                 elsif(state=SEND_HOLD_DATA) then



                 else
                      trigger <='0';
                 end if;
              state<=next_state;

             end if;

 end process;

combinational: process(state,start,enc_counter,
trigger_count,q_plain_text)

  begin
   next_state<=state;
   enable <='0';
   sbox_input<= (q_plain_text(7 downto 0) XOR X"03");
  case (state) is
     when IDLE_AES => if (start ='1') then
                      next_state<=HOLD_AES;

                      end if;

     when HOLD_AES => if (start='0') then
                       next_state<=SEND_HOLD_DATA;
```

```
                        end if;
    when SEND_HOLD_DATA => enable<='1';
                           next_state <=XOR_SBOX;

    when XOR_SBOX =>
                        sbox_input<= (q_plain_text(7 downto 0)
                       XOR X"03");
                        next_state<=TRIGGER_HOLD;

    when TRIGGER_HOLD => if(enc_counter="11111111111111") then
                          next_state<=IDLE_AES;

                          elsif(trigger_count=
                          "1111111111111111111111")then
                          next_state<=HOLD_AES;
                          else

                          end if;

    end case;
    end process;

end Behavioral;
```