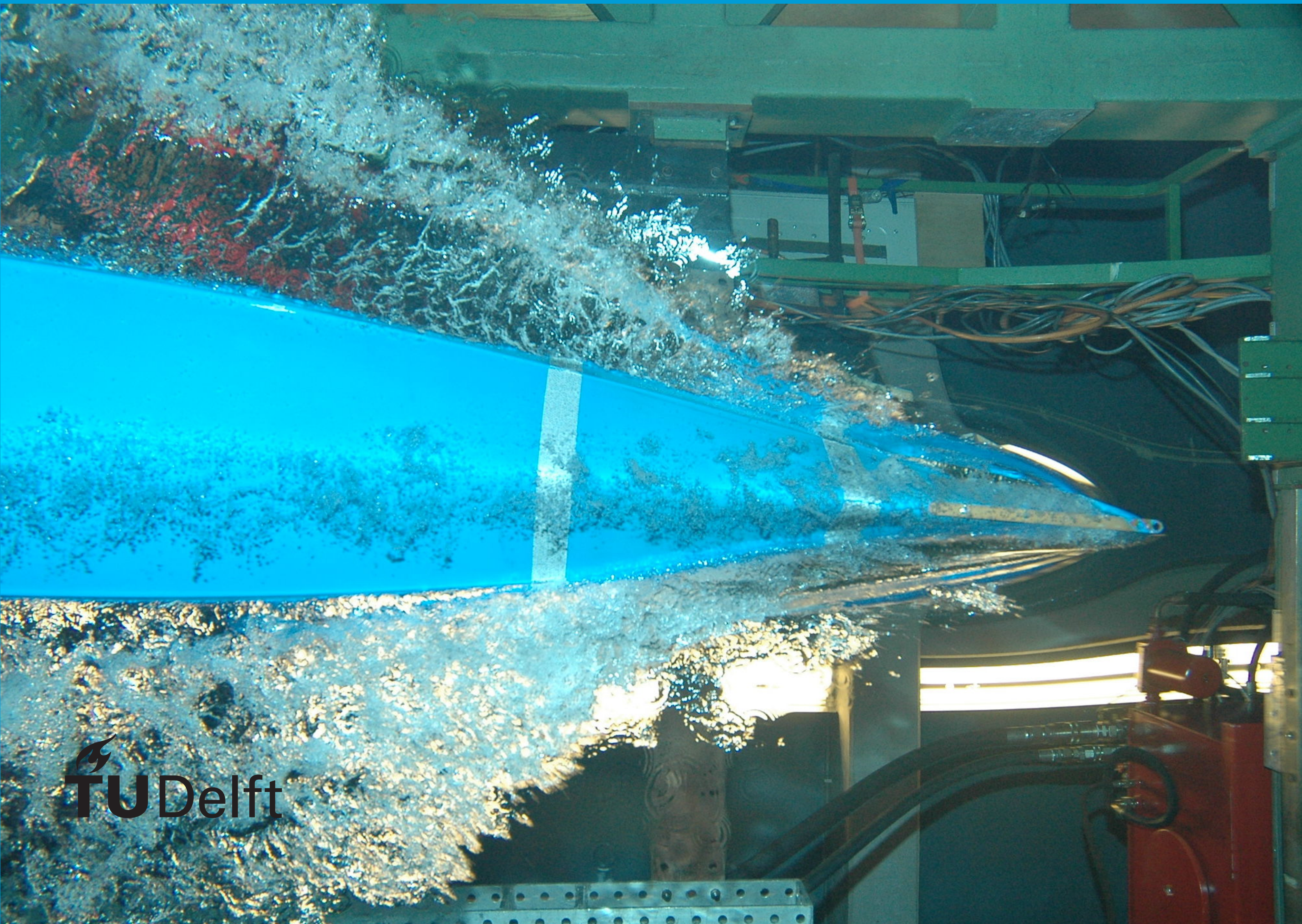
Thermal Side Channel Analysis

Marc Zwalua



Thermal Side Channel Analysis

by

Marc Zwalua

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Monday October 28 2020 at 13:00.

Student Number:	4137671	
QCE Thesis Number:	Q&CE-CE-MS-2020-12	
Project duration:	September 1, 2019 – October 28, 2020	
Thesis committee:	Prof. dr. S. Hamdioui,	TU Delft
	Dr. ir. M. Taouil	TU Delft, supervisor
	Dr. C. R. Wedig Reinbrecht	TU Delft, supervisor
	Dr. S. Picek,	TU Delft

TU Delft

Preface

After all of these great years in Delft, it is time to wrap up my studies. Although I enjoyed most of my studies, I do have to admit it was a turbulent ride. With lots of highs but of course also some inevitable lows. I can't say I enjoyed every part of writing my thesis during this global pandemic, but I can say that finishing this research was the one of my many pinnacles of my time in Delft. In no particular order I am very thank full for, my parents (Anske and Sippy Zwalua), my fellow board members of the 142^{nd} board of the Electrotechnische Vereeniging, my girlfriend Caroline Mackenbach, Richelle van Capelleveen, Mottaqiallah Taouil and Cezar Wedig Reinbrecht of the Quantum Computing Engineering department and other friends that have supported me during this long journey.

Last but not least, I want to thank you, the reader of this thesis. This almost 100 page book might seem daunting at first, but I did my best to make it as enjoyable as possible.

With ever innovating greetings,

Marc Zwalua
Delft, October 28 2020

Contents

Acronyms

ADC Analog to Digital Converter. 40, 51

AES Advanced Encryption Standard [34]. 17, 18, 21–24, 31, 33, 73, 74

ASCII American Standard Code for Information Interchange [2]. 18, 19

BINEX Binary Expansion [72], the naive square and multiply algorithm for modular exponentiation. 26, 29, 30, 37, 44, 45, 48, 55–59, 61, 63, 64, 69, 71, 76

CNN Convolutional Neural Network [45]. 5, 13, 14, 32, 46–49, 55, 63–71, 73, 74, 80

CPA Correlation Power Analysis [60]. 13, 32, 44, 46, 55, 58, 71, 73, 79

CPU Central Processing Unit. 13

CRT Chinese Remainder Theorem [83]. 12, 13, 33, 34, 37

CTA Correlation Thermal Analysis. 14, 44, 45, 56–61, 68–71, 73

DES Data Encryption Standard [76]. 18, 21, 30, 31

DH Diffie-Helman [35]. 12

DPA Differential Power Analysis [55]. 13, 32, 46, 55, 73

ECC Elliptic-Curve Cryptography [53]. 18, 74

FPGA Field Programmable Gate Array. 51–53, 57, 71

GCD Greatest Common Divisor. 25, 26, 34, 36

GPU Graphical Processing Unit. 46

HD Hamming Distance [7]. 28, 41, 42, 44

HTTPS Hypertext Transfer Protocol Secure [8]. 12, 18, 24

HW Hamming Weight [9]. 13, 28, 31, 38, 41, 42, 44–46, 56, 57, 59, 60, 74

IOT Internet of Things. 11

LSB Least Significant Bit. 21

LUT Look Up Table. 21

M2M Machine to Machine. 11

MD5 Message-Digest algorithm [70]. 18

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor [51]. 27, 40–42

MSB Most Significant Bit. 21

OpenSSL Opensource Transport Layer Security (TLS) and Secure Sockets Layer (SSL) toolkit [13]. 33, 76

PCPA Progressive Correlation Power Analysis. 28, 46, 57, 63, 64, 71, 73, 74

PCTA Progressive Correlation Thermal Analysis. 14, 45, 46, 57, 60–63, 71, 73, 74

PL Programmable Logic. 53

QR Quarter Rounds [29]. 24

RSA Rivest–Shamir–Adleman encryption algorithm [69]. 12, 13, 18, 19, 24–26, 32–34, 36, 37, 40, 43–45, 48, 55, 57–65, 68, 69, 73, 74

SHA Secure Hash Algorithm [37]. 18, 20, 21, 73

SHA-256 Secure Hash Algorithm 256 bit [41]. 18

SPA Simple Power Analysis [55]. 13, 32, 73

SSH Secure Shell [16]. 12, 18, 24

STA Simple Thermal Analysis. 14, 44, 73

TiA Timing Attack [56]. 33, 73

VCC Voltage Common Collector. 40, 42

XADC Xilinx Analog Digital Converter [19]. 42, 43, 51–54

List of Figures

List of Tables

Abstract

Within a world that increasingly relies on connected devices, security and reliability have become more important then ever. Whereas failures in digital components used to have a limited effect, nowadays an attack on a critical digital infrastructure impacts our daily lives on a huge scale. Due to these impacts it is estimated that the financial burden of cybercrime is going to increase to 6 trillion dollars by 2021. These costs are estimated to increase even more with the ever increasing amount of connected devices to the internet. To limit this problem, new ways of protecting digital systems must be developed. However, to be able to protect something, it is first necessary to identify the possible threats. In order to do so, vulnerabilities in digital systems have to be found. Preferably before a potential attacker finds them.

This thesis looks for such vulnerabilities with respect to Thermal Side Channel Attacks. Whilst power side channel analysis has been around since the mid 90s, the amount of research on thermal side channel analysis has been very limited. However, when it comes to non-invasive side channel analysis, thermals can be of great use. Compared to power side channel analysis, it is much easier to do a non invasive thermal measurement. In some cases, the sensors for thermals are already included and available in the chip itself. This makes them a very suitable target.

To perform Thermal Side Channel Attacks, three passive side channel attacks were developed based on existing power attacks. These attacks, Simple Thermal Analysis, Correlation Thermal Analysis and Convolutional Neural Network Thermal Analysis aimed at retrieving a private key from a RSA decryption. Because the thermal traces show a very different behaviour compared to the power traces, some extensive pre-processing techniques had to be developed. This pre-processing consists of removing a drifting offset and filtering unwanted frequencies in the collected trace. To further improve the attack success, a novel attack was created to retrieve the private key from a Montgommery Ladder based RSA implementation. This protected RSA algorithm is used as a counter measure against many side channel attacks. Using Progressive Correlation Thermal Analysis (PCTA), it was possible to attack this algorithm. As this attack does not exist in the power domain, it was also converted to work with power traces referred to as Progressive Correlation Power Analysis (PCPA). Using this technique, the existence of a Thermal Side Channel Attack was proven by successfully attacking multiple temperature traces and retrieving the private key.

1 Introduction

This thesis starts with the introduction chapter. In Section 1.1 the motivation for thesis will be given. After this, the State of Art analysis is showed in Section 1.2. The contribution is showed in Section 1.3 and this chapter ends with Section 1.4.

1.1. Motivation

In the current data driven world, the amount of devices is growing every day. It is estimated that at the end of 2020, there are over 25 billion Internet of Things (IOT) devices (as seen in Figure 1.1) and that the market is still growing rapidly. One aspect of IOT devices is that they often are open and unprotected in the field. This makes this growing group of well connected devices a perfect target for side channel attacks. The reason that these devices are vulnerable

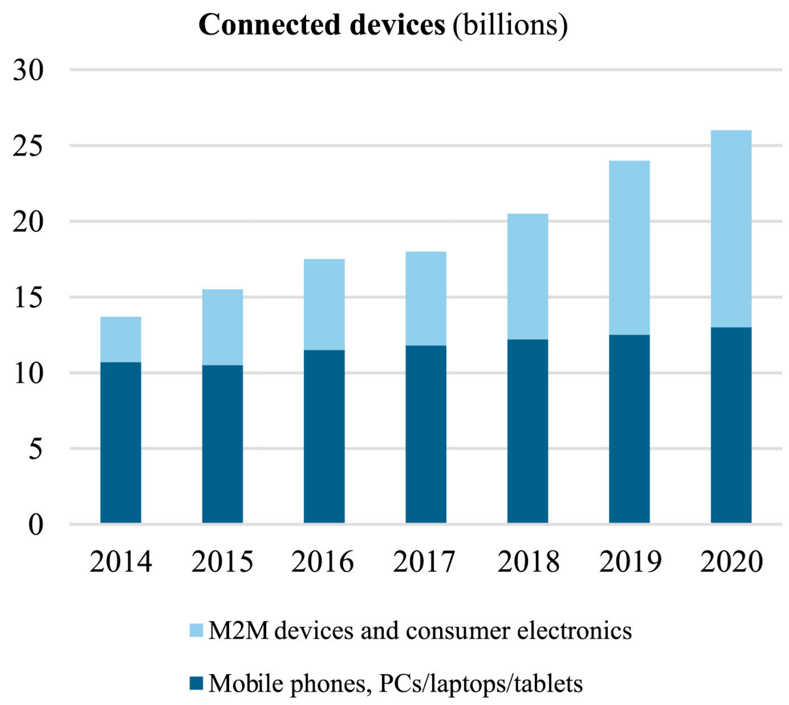


Figure 1.1: This figure shows the growing market of IOT devices. It clearly shows that the Machine to Machine (M2M) market grows fast [75].

On top of that, it is also predicted that the damage due cyber-attacks is going to increase to 6 Trillion dollars by 2021 [79]. These cyber attacks are not only limited to the virtual world. With an ever increasing digital society they can also attack physical systems like cars [43] or traffic lights [64] (more

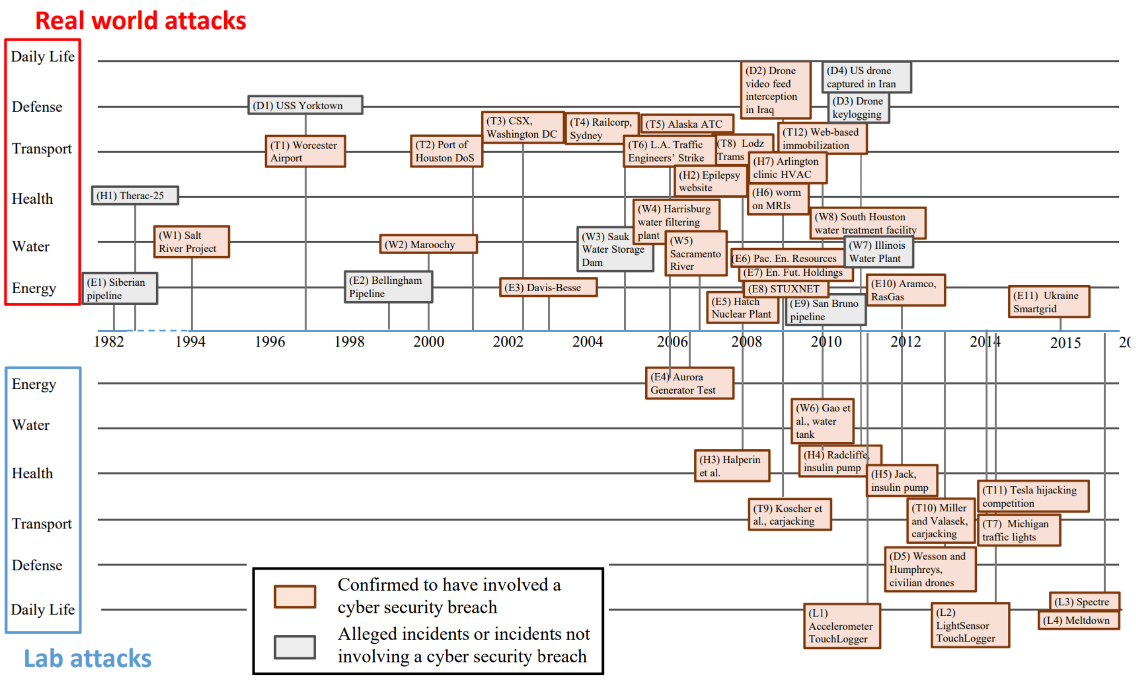

Real world attacks
Daily Life
Defense
Transport
Health
Water
Energy
(D1) USS Yorktown
(D2) Drone video feed interception in Iraq
(D3) Drone keylogging
(D4) US drone captured in Iran
(T1) Worcester Airport
(T2) Port of Houston DoS
(T3) CSX, Washington DC
(T4) Railcorp, Sydney
(T5) Alaska ATC
(T6) L.A. Traffic Engineers' Strike
(T8) Lodz Trams
(T12) Web-based immobilization
(H1) Therac-25
(H2) Epilepsy website
(H6) worm on MRIs
(H7) Arlington clinic HVAC
(W1) Salt River Project
(W2) Maroochy
(W3) Sauk Water Storage Dam
(W4) Harrisburg water filtering plant
(W5) Sacramento River
(W7) Illinois Water Plant
(W8) South Houston water treatment facility
(E1) Siberian pipeline
(E2) Bellingham Pipeline
(E3) Davis-Besse
(E5) Hatch Nuclear Plant
(E6) Pac. En. Resources
(E7) En. Fut. Holdings
(E8) STUXNET
(E9) San Bruno pipeline
(E10) Aramco, RasGas
(E11) Ukraine Smartgrid
1982
1994
1996
1998
2000
2002
2004
2006
2008
2010
2012
2014
2015
Energy
Water
Health
Transport
Defense
Daily Life
(E4) Aurora Generator Test
(W6) Gao et al., water tank
(H3) Halperin et al.
(H4) Radcliffe, insulin pump
(H5) Jack, insulin pump
(T9) Koscher et al., carjacking
(T10) Miller and Valasek, carjacking
(T11) Tesla hijacking competition
(T7) Michigan traffic lights
(D5) Wesson and Humphreys, civilian drones
(L1) Accelerometer TouchLogger
(L2) LightSensor TouchLogger
(L3) Spectre
(L4) Meltdown
Confirmed to have involved a cyber security breach
Alleged incidents or incidents not involving a cyber security breach
Lab attacks

Figure 1.2: This overview shows various cyber physical attacks. The attacks on the top are considered real attacks while the bottom ones are attacks that only happened in the lab. [80]

can be seen in Figure 1.2). This combined with the vulnerability for side channel analysis, makes this research topic very interesting.

Since the publication of the first physical side channel attack in 1998 [54], side channel attacks have been focusing on the power consumption of crypto-devices. As a result of this, a lot of counter-measures have been focusing on obscuring the power consumption for an attacker. For example by shielding [48] or measuring inconsistencies on the power lines. As a result of these countermeasures and the newly developed techniques, researchers have been looking into other sources for information leakage. Electromagnetic [22], optical [81] and even acoustic side channel attack [40] were suggested and tested. Although the thermal domain has been suggested, it has barely been researched.

The development of a practical thermal side channel attack would open a new research segment and uncover new vulnerabilities of current integrated circuits. Additionally, the exploitation of temperature as a side channel has other benefits. The thermal leakage has the advantage compared to other sources to be measured cheap and even non-invasive. Especially for more complex systems where the power is heavily regulated this can be advanced. Due to the amount of buck-boost converters it can be hard to get a proper power trace. Therefore adding non-invasive an external sensor can be very useful. In some cases it is not every required to place a sensor. A lot of microprocessors already have a temperature sensor build in. Not only can these sensors give more directional estimation of the power consumption, in most cases they're open and accessible for everyone. These properties make investigating thermal side channel analysis very interesting.

The algorithm that has been attacked in this thesis is Rivest–Shamir–Adleman encryption algorithm [69] (RSA). This is one of the most used asymmetric encryption algorithms today and that is critical for the functionality of the world wide web. It is used during the key-exchange for Hypertext Transfer Protocol Secure [8] (HTTPS) with Diffie-Helman [35] (DH) [35] to make sure both parties have the same key without sharing it with the outside world but also for the keys of Secure Shell [16] (SSH). In addition to that, most attacks on RSA have been focusing on using fault injection [50] on RSA Chinese Remainder Theorem [83] (CRT) . This makes new discoveries possible when performing observing attacks like correlation analysis.

1.2. State of Art Analysis

Like mentioned in the introduction, the first physical side channel attack was published in 1998 by Kocher [54]. Here two methods for power side channel analysis was shows. Simple Power Analysis [55] (SPA) and Differential Power Analysis [55] (DPA). The first one consisted of analyzing a power trace by hand while the latter one showed a method with statics to retrieve secret information. Although this paper is considered as the first real physical side channel attack, it wasn't exactly the first side channel attack. In 1996 Kocher also published the concept of a timing side channel attack [57]. This was the first concept of a non physical side channel attack. The timing attack suggested that it was possible to retrieve secret information by analyzing the response time of a system.

Since 1998 these techniques have been further improved. In 2002 a new method called Correlation Power Analysis [60] (CPA) was published [61]. This method, just like DPA, used statistics to retrieve secret information. In the mean time also other techniques likes higher-order DPA was developed [55] to further improve on statistical models. Another powerful method that has been proven many times, it a machine learning side channel attack [46]. This method first published in 2011, uses machine learning to analyze traces. This method has since then been expanded in many ways, for example by using CNN [23].

Although observing a system can result in retrieving secret information, it was also discovered that injecting a system can work as a side channel attack. One method of doing this, is a Fault Injection [25]. By creating an error during a critical operation, it is possible to trick a system into spilling secret information.

Whereas power side channel attacks have been researched for a long time, the current research on thermal side channel attacks has been very limited. Until now, there has not been an observing thermal side channel attack published. There has been a suggestion that there is a relationship between of Hamming Weight [9] (HW) of data transferred to memory and thermal behaviour[47], but the article itself does not prove the existence of an observing attack. It only shows that is possible to retrieve the private key of RSA CRT while performing an heat induced fault injection. Another paper proves that it is possible to leak information through the fan speed of a Central Processing Unit (CPU) [31], however the bandwidth of this is very low and the information that is leaked is created by the attacker. It can be used to transfer data, but it not suitable for attack a crypto-algorithm.

1.3. Contribution

The main goal of this thesis is to further improve the knowledge on thermal side channel analysis. Due to the research gap on thermal side channel analysis, first an investigation had to be to prove the possibility of an attack. After an elaborate research with different micro-processors, the research continued looking into the attacks themselves. With the practical knowledge on thermal side channel analysis the main contributions of this thesis are:

- **Proposal for a novel attack on an protected RSA algorithm:** in order to perform a side channel attack on a Montgommery implementation of RSA, a new method was developed. This new method, Progressive Correlation Analysis, is an improved variant of Correlation Analysis. Due to the novelty, a version for thermal, Progressive Thermal Correlation Analysis and a version for power, Progressive Power Correlation Analysis were created.

- **Proposal of three new techniques inspired on power side channel analysis:** to perform a thermal side channel attack, three new methods were developed. These methods, Simple Thermal Analysis, Correlation Thermal Analysis and Convolutional Neural Network Thermal Analysis were all inspired on existing attacks for power side channel analysis.

- **Successful evaluation of these new techniques for different RSA implementations:** during this elaborate evaluation, the newly developed thermal and power side channel analysis were tested on traces of a RSA decryption. The target of each attack, was the 1024 bit private key. In all cases, the newly developed methods were able to recover this 1024 bit key.

- **New methodology to exploit thermal leakage from micro-processors:** during the research it was discovered that there was a relationship between Hamming Weight of the processed data and the temperature. This relationship was proven before for memory operations [32], but not yet for values within an algorithm and not yet for these frequencies.

To summarize these contributions, this thesis proves the existence of thermal side channel attack. It contributed a new group of attacks and also added a new attack method for power side channel analysis.

1.4. Thesis Organization

This thesis is divided in five parts and an Appendix. The organisation of this thesis can be seen in the following summary:

- **Chapter 1:** this chapter will give the motivation, state of art analysis and the contribution of this research.
- **Chapter 2:** this chapter describes the necessary background in two main subjects, encryption and side channel analysis. The encryption part will describe a classification and an example of every type. During the other part, side channel analysis, the basics of side channel analysis are described. Also a classification system is given on how the various attacks can be labeled.
- **Chapter 3:** this chapter describes the methodology of this research. It also further explains how various side channel attacks work in practice. This chapter also explains the newly developed techniques such as Simple Thermal Analysis (STA), Correlation Thermal Analysis (CTA), Progressive Correlation Thermal Analysis (PCTA) and the use of CNN on thermal side channal attacks. The details of the implementation can be seen in the next chapter.
- **Chapter 4:** in this chapter the implementation, validation and results are given. During this chapter, also the various challenges and solutions are described. This chapter ends with a discussion on the results.
- **Chapter 5:** in the last chapter, the summary and the future work are described.
- **Appendix:** this part of this thesis shows a list of all the traces and a overview of all the histograms.

2 Secure Systems and Hardware Attacks

This chapter will give background information that is required for Chapter 3 and Chapter 4. To begin with, an introduction with some history on secure systems is given. After this, basic cryptography and side channels are discussed. The most important part of this chapter is understanding the current state of cryptography and side channel analysis. This is required for understanding the methodology, implementation and the results

2.1. Need for Secure Systems

The need for hiding a message from one another has been around for a very long time. As early as 500-600 BC, evidence of an early cipher system has been found [65]. This simple substitution cipher consisted of replacing the letter with a reverse alphabet, for example a becomes z, b becomes y etc. Although the exact goal was not clear, it was considerd one of the first cipher systems.

Another early cipher system was the famous Ceasar Cipher. This simple substitution cipher was used to encrypt messages for Ceasars generals. Only someone with the correct substitution pattern was able to read the messages. The substitution consisted of right or left shift with n positions [24]. In Figure 2.1 you can seen an example with $n = 3$. As the figure shows, an A becomes a D and an E becomes an H.

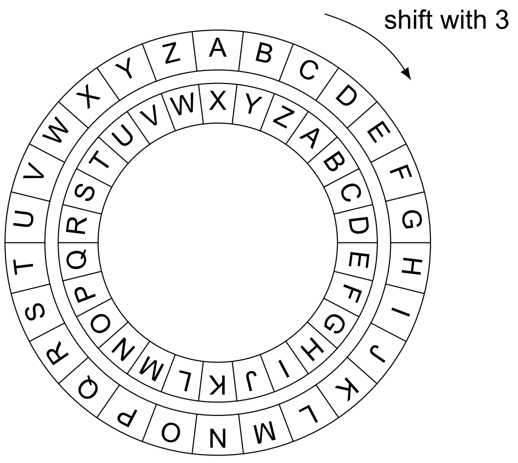


Figure 2.1: The outer wheel is shifted 3 times to the right. The inner wheel is static. With this, A becomes D, B becomes E, etc [3].

Since then, encryption has evolved to many complex. To get an overview of all the properties, the CIA-triad [80] was constructed. This CIA-triad proposes three requirements (as seen in in Figure 2.2) that make a secure system secure. Later on, a fourth requirement was added to complete the picture. These requirements are:

- **Confidentiality**: the need for someone to hide it's message for anyone that is not the intended reader. This can be done by hiding the message through encryption.
- **Integrity**: the requirement that guarantees that the message is not altered by a third party. When a message is integer it is not altered.
- **Availability**: being able to communicate whenever this is needed.
- **Authenticity**: the confirmation that the send message is really send by the intended sender and not by a third party.

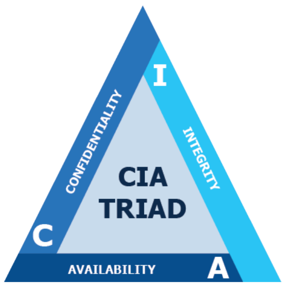


Figure 2.2: CIA Triad [4]

With the extended CIA-triad, it is possible to create a secure system. In order to fulfill these requirements, encryption can be applied in three out of the four properties. For example, to keep data confidential a message could be convert to cipher text with the aid of encryption and if only the sender and the receiver have the key, the data is hidden. This same goes for integrity. If a message is encrypted, it is not possible to alter it without having the key. It is of course possible to inject some characters, but when the cipher text is decrypted, it will end up in garbage data. Also authenticity can be guaranteed with encryption. If Alice and Bob both have the same key and a symmetric encryption scheme is used, Bob knows Alice is the only one that is able to send the message. The same goes for Alice. With the aid of asymmetric encryption this guarantee can even be stronger [63]. The only thing that can not be guaranteed by encryption is availability. To solve this, redundant communication lines or backups should be used.

2.2. Cryptographic Algorithms

In order to fully understand side channel analysis, it is necessary to have some background in various encryption algorithms. This section starts with an introduction on encryption. After this a classification of different encryption algorithms is shown and with that a few elaborate examples of encryption algorithms.

As seen in Section 2.1, encryption plays a very important role in secure systems. To keep track of all the algorithms, encryption algorithms can be separated in three categories. These categories are based on the amount of keys they require. The three categories are:

- **Asymmetric**: this type of encryption uses two different keys, one for encryption and one for decryption. For example, message A gets encrypted with key_{public} and the cipher text gets decrypted by $key_{private}$. It is not possible to use key_{public} for decryption. This type of encryption is often used as key-exchange.
- **Symmetric**: unlike asymmetric encryption, this type only uses one key. This key is the same for encryption and decryption. Because of this property, it is very important to keep this key hidden. This type of encryption is often used to encrypt large data sets.
- **Hash**: although these types of algorithms are not considered encryption at all (because they don't use a key), there are commonly used in encryption schemes and share a lot of similarities. Therefore it was added to this classification. A hash function obscures information without using

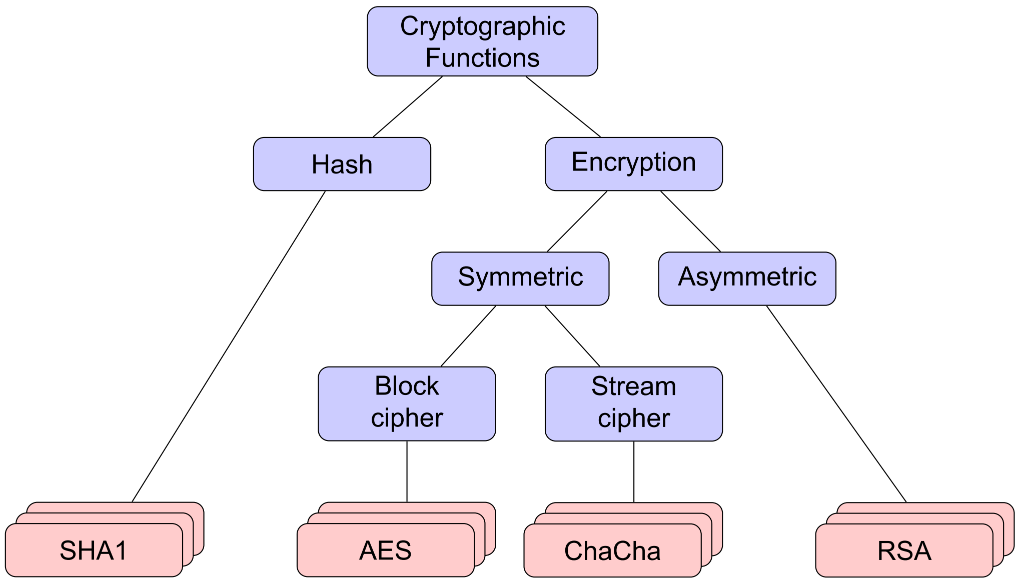


Figure 2.3: The classification of cryptographic algorithms

a key. One property of this type of functions is that it's a one way function. Meaning, that it is impossible to retrieve the original message after it has been hashed. A popular purpose for hash function is for password verification.

Within symmetric encryption, there are two types of ciphers. The first one is block cipher. This encryption method encrypts data in blocks of bytes. In the case of Advanced Encryption Standard [34] (AES), this means that encryption done in blocks of 128 bits. When a message only has 14 bits, the other 114 bits are padded with zeros. This padding can lead to some unwanted situations. Therefore, stream ciphers were developed. These ciphers work on bit level instead of blocks of bits. An overview of these methods can be seen in Figure 2.3. Some other examples and their applications can be seen in Table 2.1.

For encryption schemes there are two important properties, confusion and diffusion. Confusion is the property that indicates on how many bits of the key a single bit of the cipher text is depended on [79]. An example of confusion can be seen in Example 2.2.1.

Example 2.2.1: Confusion

key=0110111	cipher=010110	original
key=0110011	cipher=010110	low confusion
key=0111011	cipher=010110	high confusion

The red bits indicate the bits that influence each other. In the case of low confusion, one bit in the key, changes only one bit in the cipher. In the case of high confusion, many key bits are involved for only a single bit in the cipher. This makes it very hard to predict if the cipher is going to be a one or a zero.

Diffusion on the other hand, is the amount of bits that are altered when a single bit changes in the plain text. When this property is high, a single bit in the message will change a high amount of bits in the cipher text [79]. An example of this can be seen in Example 2.2.2.

Example 2.2.2: Diffusion

message=100011	cipher=0110101	original
message=100001	cipher=0110101	low diffusion
message=100001	cipher=1101001	high diffusion

The red bit indicate the bits that have changed. In the case of low diffusion, if one bit in the message changes, one bit in the cipher changes. On the other hand in the case of high diffusion, one toggled bit in the message changes most bits in the cipher.

In the following sections, four encryption algorithms are explained. The background starts with AES and then follows with ChaCha, Message-Digest algorithm [70] (MD5) and RSA. Because of the importance of RSA, multiple implementations of this algorithm are also added.

Table 2.1: Overview of cryptographic algorithms and their application

Type	Algorithms	Applications
Block cipher	AES, Data Encryption Standard [76] (DES), 3DES	Encrypting wide variety of data like wireless communication, HTTPS traffic and mary others
Stream cipher	ChaCha,RC4	Encrypting data when there is low computing power,
Asymmteric encryption	RSA, Elliptic-Curve Cryptography [53] (ECC)	Key exchange between parties in for example HTTPS and SSH
Hash	MD5, Secure Hash Algorithm 256 bit [41] (SHA-256),	Hashing of credentials, fingerprints of keys, checking data integrity

2.2.1. SHA1

Secure Hash Algorithm [37] (SHA)1 has been designed by the National Security Agency and was published in 1995 [37]. It was used throughout the mid 2000s until it was no longer considered secure. Although SHA1 is not used anymore (at least is should not), it's successors (like SHA-256) are still very popular today. Although SHA1 is not being used anymore, it's algorithm does show the basics for hashing. Just like any other hashing algorithm, SHA was developed as a one way function and converts a variable length input to a 40 hexadecimal string (160 bits).

These properties of hashing, are commonly used in the credential checking. The reason this is done, is that the server does not have to save the password of a user in plain text. If the database of that server is taken, the attacker only has a database with hashes. These hashed values are useless for two reasons. One, the attacker is not able login with credential from the hacked database and two because the attacker can't see what the password of a user is. Because humans often re-use password, this is also a popular target.

Another application of hashing, is during the signing of a message. RSA is often used for signing. However, to ensure the input message stay between healthy bounders (too small is unsecure while too long take too much time to compute), a hashing algorithm is used. An overview of this can be seen in Figure 2.4. Although each hashing algorithm has it's own characterises. Most of them use the following steps:

Padding

Let's say the goal is to calculate the hash of the message ”test142”. The first step would be to convert this string to 8 bit American Standard Code for Information Interchange [2] (ASCII) code. In other words ”test142” would become $[116, 101, 115, 116, 049, 052, 050]$ in binary with 8 bits. Next, this 56 bit example is joined and padded with a one. After this the message is padded with zeros until the length

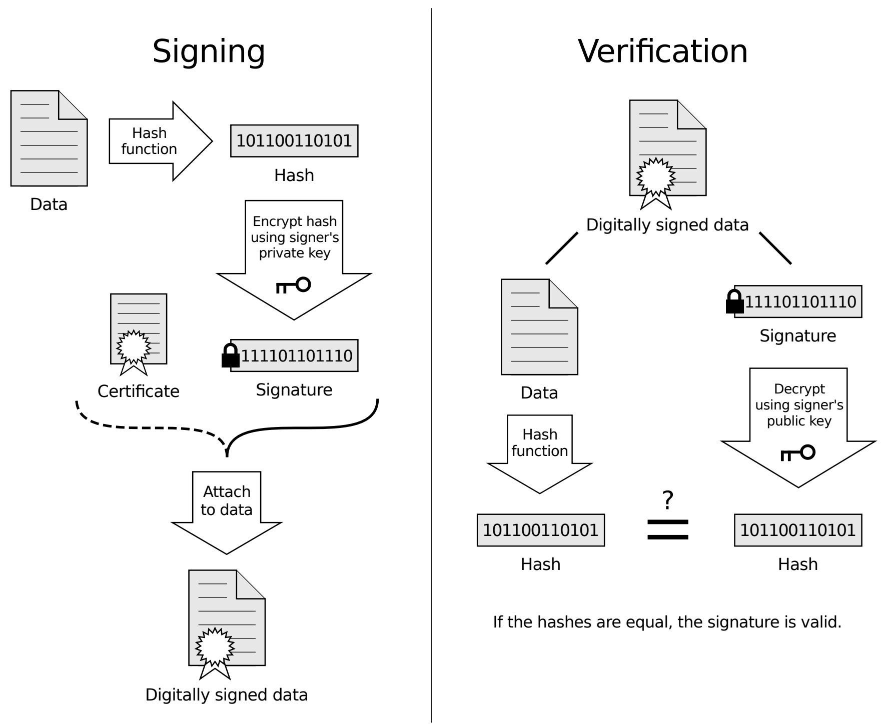


Figure 2.4: This example [14] shows how hashing can be used in combination with a RSA signature. By first hashing data, the input is bounded. This has the advantage that the message isn't too short (not secure) or too long (too much computing time). Another advantage is, that the data that is hashed is always random. In that way it is much harder to forge the data that is used for the signature.

is $512 \mod 448$. In the example, this results in 391 zeros to make it 448 bit long. The next step in the padding, is accounting for the length of the input message (7 characters equals 56 bits). This is done by convert 56 to binary and padding zeros in front of it until it's length is 64 bit long. These steps can be seen in Example 2.2.3.

Example 2.2.3: Padding

input:	"test142"
ASCII:	$[116, 101, 115, 116, 049, 052, 050]$
binary:	$01110100011001010111001101110100001100010011010000110$
padding:	binary("test142") $+1+$ zeros(391)
length input:	56
padded length:	zeros$(58) + 111000$
output:	binary("test142") $+1+$ zeros$(391) +$ zeros$(58) + 111000$

Expanding the words

After the padded string has been created, it can be separated in 16 32 bit words. With the aid of some XOR operations the 16 words get expanded to 80 words. The pseudo code for doing this, can be seen in Algorithm 1.

Initialize variables

Now that the words are expanded, five variables need to be initialized. These five variables are changed every round until the algorithm is finished. Then they form the output of the algorithm. Because these five variables always contain 32 bits and are initialized according to Table 2.2, the output is always 160 bit long (5 times 32).

Algorithm 1 Expand 16 words to 80 words [37]

```
for i := 16 to 79 do
    word[i] = (word[i-3] xor word[i-8] xor word[i-14] xor word[i-16]) leftrotate 1
end for
```

Table 2.2: Initial values of the five SHA1 variables

h0:	0x67452301
h1:	0xEFCDAB89
h2:	0x98BADCFE
h3:	0x10325476
h4:	0xC3D2E1F0

Looping through the words

The biggest step in the SHA1 algorithm, is looping through the variables. There are 80 rounds. During these rounds, there are four possible operations.These operations are bitwise operations that manipulate the values in a,b,c and d. After each round, the variables are shifted and the word of that round (hence the amount of words) is added in the result. How this exactly works can be seen in Algorithm 2.

Algorithm 2 Main loop SHA1

```
a ←h0, b ←h1, c ←h2, d ←h3, e ←h4
for counter := 0 to 79 do
    if $0 \leq i \leq 19$ then
        f ←(b and c) or ((not b) and d)
        k ←0x5A827999
    else if $20 \leq i \leq 39$ then
        f ←b xor c xor d
        k ←0x6ED9EBA1
    else if $40 \leq i \leq 69$ then
        f ←(b and c) or (b and d) or (c and d)
        k ←0x8F1BBCDC
    else if $60 \leq i \leq 79$ then
        f ←b xor c xor d
        k ←0xCA62C1D6
    end if
    temp ←(a leftrotate 5) + f + e + k + word[i]
    e ←d
    d ←c
    c ←b leftrotate 30
    b ←a
    a ←temp
end for
```

Creating the hash

In order to finalize the hash, the results of the variables a,b,c,d and e have to be added to h0 to h4. With these values in the h register, the hash can be created by concatenating the registers from h0 until h4. This can be seen in Example 2.2.4.

Example 2.2.4: Creating the hash

a = 0x032A58D4D
b = 0xF47D7B7D5
c = 0xF7CE8EE74
d = 0xF7D41F84F
e = 0xF987DE831

h0 = h0 + a = 0x99EAB04E
h1 = h1 + b = 0x37A5635E
h2 = h2 + c = 0x15A3CB72
h3 = h3 + d = 0x92D05C27
h4 = h4 + e = 0x5C50CA21

Hash = join(h0,h1,h2,h3,h4) = 99EAB04E37A5635E15A3CB7292F05C275C50CA21

By adding a,b,c,d,e to the previous values of h0,h1,h2,h3,h4 and joining them together, the output Hash is calculated

One of the things that make SHA unsecure is the problem of collisions [78]. A collision is a hash that remains the same for multiple inputs. This can be problem since hashes are sometimes used to verify the authenticity. If this verification can be faked by using another input, the verification can't be trusted. This one of the reasons, SHA1 was upgraded.

2.2.2. AES

AES was developped by Vincent Rijmen, Joan Daemen in 1998 [34]. The main reason for the development of the algorithm was the need for a more secure symmetric encryption algorithm then DES. Just like DES and 3DES [74], AES is a symmetric block cipher algorithm. That means it encrypts in blocks and uses the same key for decryption as for encryption.

The AES implementation is based on four steps. SubByte, ShiftRows, MixColumns and AddRoundKey. The algorithm works in blocks of 128 bits (often represented in a 4 by 4 matrix). The key can be 128, 192 or 256 bit long.

The algorithm starts with key expansion. This means that the original 128 bit key gets expanded to $11 \cdot 128 = 1408$ bit, $13 \cdot 192 = 2496$ bit or $15 \cdot 256 = 3840$ bit. This is done with the key schedule. Although the process of expanding the key is very interesting, it is out of the scope for this research.

The sequence of AES can be seen in Figure 2.5. First it starts with Key Expansion and AddRoundkey, then it does 9, 11 or 13 rounds (for 128, 192 or 256 bit key) and then it ends with a final SubByte round. The individual rounds are explained in the following paragraphs.

SubByte

During this step, bytes of the 4 by 4 matrix, are getting translated to new bytes according to a 16 by 16 table (as seen in Table 2.3). The first four bits (Most Significant Bit (MSB)) are used for looking up y value, while the last 4 bits (Least Significant Bit (LSB)) are used to look up the x value. An example of this can be seen in Example 2.2.5. This step is used for creating confusion. For side channel attacks, the SubByte step is usually the most interesting step because it consumes the most power [84]. The Look Up Table (LUT) is sometimes also referred to as the S-Box.

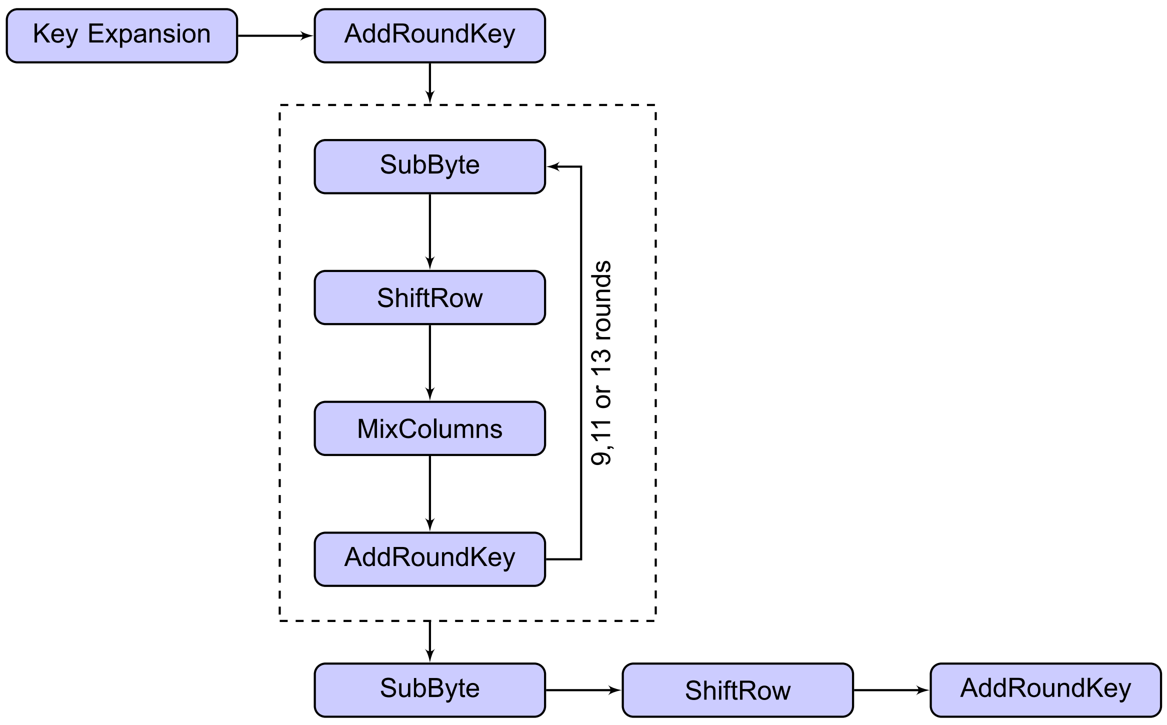


Figure 2.5: The AES sequence. First the key is expanded, AddRoundkey is performed. Then 9, 11 or 13 rounds are performed depending on the key size. To finish, SubByte, ShiftRow and AddRoundKey are performed one last time.

Example 2.2.5: S-Box look up method

Input = 0x6A
Horizontal = 60
Vertical = A0

S-box = 0x02
By taking 60 on the horizontal axes and A0 on the vertical axes, 0x02 can found in Table 2.3

ShiftRows
This step shifts rows in the 4 by 4 matrix. Depending on the row number, that amount gets shifted to the left (as seen Equation 2.1). This means R_0 shifts 0 positions to the left, R_1 shifts 1 position to the left (so the first becomes the last byte in the row), R_2 shifts 2 positions and R_3 shifts 3 positions to the left. The reason this is done, is create diffusion.

$$\begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \to \begin{bmatrix} S'_{00} & S'_{01} & S'_{02} & S'_{03} \\ S'_{11} & S'_{12} & S'_{13} & S'_{10} \\ S'_{22} & S'_{23} & S'_{20} & S'_{21} \\ S'_{33} & S'_{30} & S'_{31} & S'_{32} \end{bmatrix} \tag{2.1}$$

MixColumns
The next step is to MixColumns. This is actually a multiplication and just like the previous step it is meant to create diffusion. It works by multiplying each column with a fixed matrix. It is however not a regular multiplication. It uses Galois Field multiplication. An example of Galois Field multiplication can be seen in Algorithm 3. The MixColumns step can be seen in Equation 2.2.

$$\begin{bmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{bmatrix} \begin{bmatrix} S_{0j} \\ S_{1j} \\ S_{2j} \\ S_{3j} \end{bmatrix} \to \begin{bmatrix} S'_{0j} \\ S'_{1j} \\ S'_{2j} \\ S'_{3j} \end{bmatrix} \text{ for } 0 \le j \le 3 \tag{2.2}$$

	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

Table 2.3: AES SubByte transform table [79]

Algorithm 3 $res = GF(a, b)$ [1]

```
for counter := 0 to 8 do
    res ←0
    if b and 1 then
        res ←(res xor a)
    end if
    carry ←(a and 0x80)
    a ←a ≪ 1
    if carry = 0x80 then
        a ←a xor 0x1B
    end if
    b ≪ 1
end for
return res
```

AddRoundKey

During this step, the key gets integrated with the data and therefore creating confusion. The operation is relative easy. Every byte from the 4 by 4 matrix gets XOR'ed with the RoundKey. This can be seen in Equation 2.3. K is the RoundKey matrix and r is the round.

$$\begin{bmatrix} S_{00} & S_{01} & S_{02} & S_{03} \\ S_{10} & S_{11} & S_{12} & S_{13} \\ S_{20} & S_{21} & S_{22} & S_{23} \\ S_{30} & S_{31} & S_{32} & S_{33} \end{bmatrix} \oplus \begin{bmatrix} K_{r00} & K_{r01} & K_{r02} & K_{r03} \\ K_{r10} & K_{r11} & K_{r12} & K_{r13} \\ K_{r20} & K_{r21} & K_{r22} & K_{r23} \\ K_{r30} & K_{r31} & K_{r32} & K_{r33} \end{bmatrix} \rightarrow \begin{bmatrix} S'_{00} & S'_{01} & S'_{02} & S'_{03} \\ S'_{11} & S'_{12} & S'_{13} & S'_{10} \\ S'_{22} & S'_{23} & S'_{20} & S'_{21} \\ S'_{33} & S'_{30} & S'_{31} & S'_{32} \end{bmatrix} \tag{2.3}$$

2.2.3. ChaCha

This stream cipher was proposed by Bernstein in 2008 [29]. The reason for the development was a low power replacement of AES. In contrary to AES, is ChaCha a stream cipher. This means that the encryption happens on bit level instead of in blocks. Compared to AES, ChaCha is relative simple. It all starts with the initialisation matrix that can be seen in Equation 2.4.

$$\text{init matrix} = \begin{bmatrix} const & const & const & const \\ key & key & key & key \\ key & key & key & key \\ input & input & nonce & nonce \end{bmatrix} \tag{2.4}$$

With this matrix, the 8 Quarter Rounds [29] (QR) are performed. The first 4 QR perform an operation on each column while the last 4 QR perform an operation on the diagonal. The QR function can be seen in Algorithm 4. A graphical overview of the ChaCha algorithm can be seen in Equation 2.5.

Algorithm 4 QR(a_i,b_i,c_i,d_i)

```
a += b; d ^= a; $d <<<= 16$;
c += d; b ^= c; $b <<<= 12$;
a += b; d ^= a; $d <<<= 8$;
c += d; b ^= c; $b <<<= 7$;
```

As seen in the graphical representation of the algorithm, the encryption takes place in 8 steps. 4 column operations and 4 diagonal operations. Four steps of QR are considered a single round. To complete the encryption, ChaCha performs 20 of these rounds. Or in other words, 10 times the sequence that is shown in Equation 2.5.

$$\begin{bmatrix} a_0 & a_1 & a_2 & a_3 \\ b_0 & b_1 & b_2 & b_3 \\ c_0 & c_1 & c_2 & c_3 \\ d_0 & d_1 & d_2 & d_3 \end{bmatrix} \rightarrow QR(a,b,c,d) \rightarrow \begin{bmatrix} a_0 & b_3 & c_2 & d_1 \\ a_1 & b_0 & c_3 & d_2 \\ a_2 & b_1 & c_0 & d_3 \\ a_3 & b_2 & c_1 & d_0 \end{bmatrix} \rightarrow QR(a_i,b_i,c_i,d_i) \tag{2.5}$$

for i is 0 to 3

The key that is used for ChaCha is a 256-bit key. With this key and a 64 bit nonce (unique message id), the input is directly converted to cipher. In contrary to for example AES, the key does not have to be pre-processed. It is directly integrated in the cipher [29]. Because it is directly integrated, the data that enters the algorithm, does not have to be separated in blocks.

2.2.4. RSA

RSA was invented by Rivest–Shamir–Adleman and published for the first time in 1977 [69]. RSA is an asymmetric encryption algorithm. That means that the encryption key is different from the decryption key. This results in a private key and a public key. Although it was published in 1977, it is still very popular today. Applications such as SSH and HTTPS rely on the security of this encryption scheme.

In order understand the algorithm, first a few variables need to be declared. These variables will be used through out this thesis and can be found in Table 2.4.

Variable	**Meaning**
m	message
c	cipher text
e	public key
d	private key
N	modulus
p	prime number different then q
q	prime number different then p
ϕ	$(p-1)(q-1)$
r	positive non zero random integer

Table 2.4: Variable declaration for RSA

Key generation

The first step in encrypting with RSA is the generation of the key [21]. To do this, two prime numbers (p and q) should be generated. These prime numbers should be large and different from each other. Also there length should differ a few digits from each other. p and q should be kept secret and are used to calculate N (see Equation: 2.6).

$$N = pq \tag{2.6}$$

Next $\phi(N)$ should be calculated. This is with Equation 2.7.

$$\phi(N) = (p-1)(q-1) \tag{2.7}$$

With the aid of $\phi(N)$ it's possible to calculate the public key. The public key e should be between one and $\phi(N)$ and both numbers should be coprime. In other words, the Greatest Common Divisor (GCD) between $\phi(N)$ and e should be one (Equation 2.8).

$$\begin{gathered} 1 < e < \phi(N) \\ GCD(e, \phi(N)) = 1 \end{gathered} \tag{2.8}$$

However, since e is public, it is not necessary to keep this key secret. To speed things up, e is almost always chosen as 65537. But in the following example a smaller number is chosen to make calculations more simple.

Last but not least, the private key d should be calculated. This can by solving Equation 2.9.

$$ed = 1(\mod \phi(N)) \tag{2.9}$$

Encrypting data

Although the key generation looks a bit complicated, the encryption of data is much easier. This is done by Equation 2.10. By raising the message to the power of the public and taking the modulo N, the message is encrypted. This might feel odd since e as well as N are both publicly known. However, this is the power of asymmetric encryption. Because the encryption key and decryption key are not equal, it is possible to share this key without sharing details. This property can then be used in other algorithm like Diffie Hellman [63] in order to exchange keys without sharing information.

$$c = m^e \mod (N) \tag{2.10}$$

Decrypting data

The next step would decrypting the data. This is done by Equation 2.11.

$$m = c^d \mod (N) \tag{2.11}$$

To explain the concept of RSA ever better, an example is showed in Example 2.2.6. Even with relative small numbers for p,q and m, the decryption still requires a massive operation. The intuitive way of calculating the decryption step would be, first 140^{461} and then take the modulo operation. However, 140^{461} would already result in a massive overflow error for most calculators.

Example 2.2.6: RSA decryption

$$\begin{aligned}
&\textbf{Variable declaration:}\\
m &= 4\\
p &= 13\\
q &= 17\\
N &= pq = 221\\
\\
&\textbf{Key generation:}\\
\phi(N) &= (p-1)(q-1) = 12 \cdot 16 = 192\\
GCD(e,\phi(N)) &= 1 \rightarrow GCD(5,192) = 1\\
ed &= 1(\mod \phi(N))\\
5d &= 1(\mod 192) \rightarrow 5 \cdot 461 = 1(\mod 192)\\
\\
&\textbf{Encrypting:}\\
c &= m^e \mod N = 4^5 \mod 221 = 1024 \mod 221 = 140\\
\\
&\textbf{Decrypting:}\\
m &= c^d \mod N = 140^{461} \mod 221 = 4
\end{aligned} \tag{2.12}$$

To make this problem even bigger, in order for RSA to be secure, large keys (minimum of 1024 bits up to 4096 bits) are used. Even with small numbers for c and m, raising something with the power of a 1024 bit number (around 330 digits long) results in an gigantic number. Not only does this take a very long time to compute, it takes even more time to compute the modulus operation of that gigantic number. Therefore, the calculation is done in multiple steps. The easiest implementation is called Binary Expansion [72], the naive square and multiply algorithm for modular exponentiation (BINEX) or also known as Square Multiply.

Implementation

The direct method of calculating an RSA encryption has two problems. First, it requires to raise the power with a huge number (more then 1000 bits). One way of calculating this, would be by multiplying with the base with it self for the exponent amount of times. In the case of 3^10 this would result in 10 multiplications and after that a modulo operation. With a 1000 bit number this would of course take a lot of time. The second problem however lays in the modulo operation. The calculation time of the modulo operation increases with the length of the input. Therefore it is faster to this operation in smaller pieces.

The backbone of the square and multiply algorithm lies in two properties. The power rule (Equation 2.13) and the distributive property (Equation 2.14).

$$a^{m*n} = (a^m)^n \tag{2.13}$$

$$a \cdot b \mod n = [(a \mod n)(b \mod n)] \mod n \tag{2.14}$$

The algorithm works by splitting the exponent in powers. So for example 10 (in binary 1010) has 4 bits. Therefor it only requires 4 operations. The first operation always starts with writing down the base number, in this case a. The next step is reading every bit from left to right with the exception of the first one. In case the bit is a zero, the previous result should be squared. If the bit is a one, the previous result should also be squared but also multiplied with base, hence the name, square multiply. In case a^10 has to be calculated, it looks as follows Equation 2.15.

$$\begin{aligned} a &= a^1 \\ a^2 &= a^2 \\ (a^2)^2 \cdot a &= a^4 \cdot a^1 = a^5 \\ ((a^2)^2 \cdot a)^2 &= (a^5)^2 = a^{10} \end{aligned} \tag{2.15}$$

This reduces the amount of steps required to calculate the power. However, the modulo of this result still has to be calculated. Luckily, it is possible to also do this in the same steps due to Equation 2.14. This results in Equation 2.16. Although the left side of the equation looks a lot more complicated, for a computer this is much faster to calculate. N limits the size of the results and this results in a faster computation.

$$(((a^2 \mod N)^2 \cdot a) \mod N)^2) \mod N = a^{10} \mod N \tag{2.16}$$

The translation of this algorithm to pseudo code is visible in Algorithm 5.

Algorithm 5 Calculate $c = m^e \mod N$

```
for i := length(e) - 2 to 0 do
    $c \leftarrow c^2 \mod N$
    if e[i] = 1 then
        $c \leftarrow (c \cdot m) \mod N$
    end if
end for
```

2.3. Side Channel Attacks

In this section, the basics of different types of side channel attacks are explained. The most important types are Simple Power Analysis, Correlation Power Analysis and side channel analysis by neural network. These types of attacks are the basis form the basis for the Thermal Side Channel Analysis. The other types of of side channel analysis are there to give a proper view of the possibilities.

The first concepts of side channel analysis have been around since the mid nine-tees [55]. The idea of a side channel analysis is by analyzing the surroundings of a crypto-system, information can be secret information can be discovered. An example of a side channel attack would be looking at the power consumption of a chip while performing cryptographic operations.

The reason a chip might leak information during use has to do with the fact that a Metal-Oxide-Semiconductor Field-Effect Transistor [51] (MOSFET) consumes power, especially while switching [51]. As a result, a processor will consume more power when it calculates something complex compared to a simple calculation. If this power is measured and analyzed in a clever way, secret information can be recovered. An example of this can be seen in Figure 2.6

To get an idea of the possible types of data side channel attacks, a classification has been made. The classification starts with two types of data attacks:

- **Passive data attack**: retrieving secret information from a crypto-system without manipulating it. Another word for this could be observing attack. An example for this type of attack would be performing a simple power analysis on a crypto-system. The main goal of the attack is measuring data.

- **Active data attacks**: active data attacks focus on injecting or manipulation the crypto-system. This could be done for example by using a fault injection, but it is also possible to manipulate in spilling data by injecting light on a chip [48].

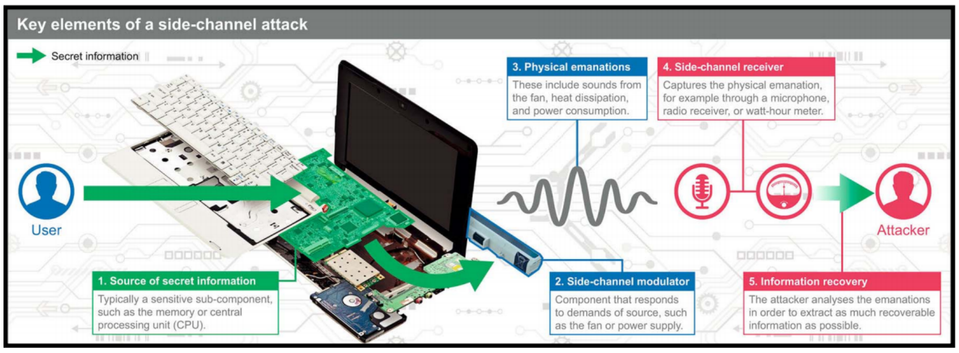

Key elements of a side-channel attack
Secret information
User
Attacker
3. Physical emanations
These include sounds from the fan, heat dissipation, and power consumption.
4. Side-channel receiver
Captures the physical emanation, for example through a microphone, radio receiver, or watt-hour meter.
1. Source of secret information
Typically a sensitive sub-component, such as the memory or central processing unit (CPU).
2. Side-channel modulator
Component that responds to demands of source, such as the fan or power supply.
5. Information recovery
The attacker analyses the emanations in order to extract as much recoverable information as possible.

Figure 2.6: Example of a Side Channel Attack [80]

.

For this classification is doesn't matter if the attack is invasive, semi invasive or non invasive. The classification is based on the main goal of the attack. An overview can be seen in Figure 2.7. From power side channel analysis, it is possible to further zoom in to the different kind of analysis. This can be seen in Figure 2.8. One of these types, Progressive Correlation Power Analysis (PCPA) is developed during this research. The other types have been around for some time.

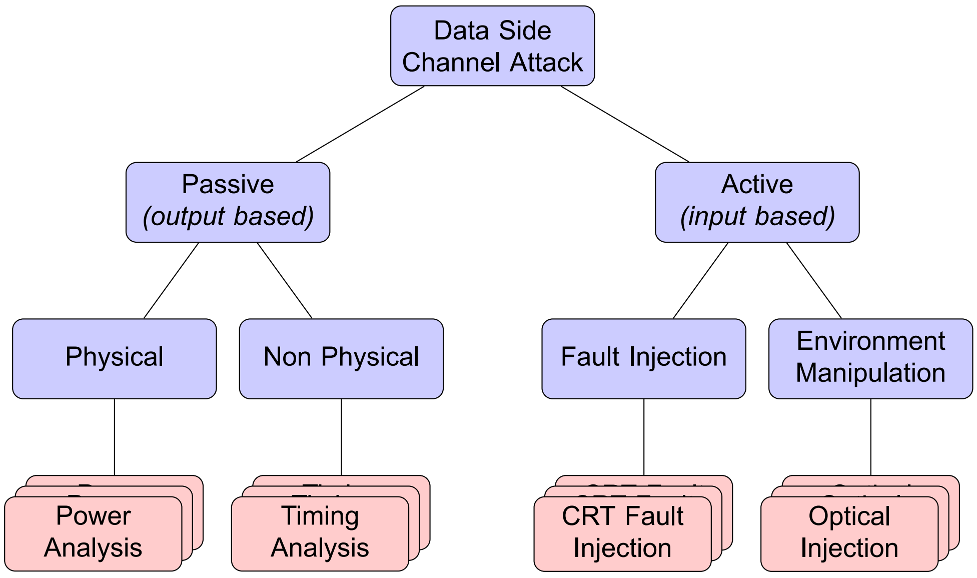


Figure 2.7: The classification of different side channel attacks

Leakage models

In order to make a proper estimation about what the crypto-system is performing, a leakage model should be created. These models are the base of side channel analysis and tell the attacker what to look for in his gathered data. Some places to look for leakage are:

- Different power per operation
- Different timing per operation
- HW of (intermediate) data
- Hamming Distance [7] (HD) of (intermediate) data

For example, a crypto-system uses two operations based on the bit of the key in order to encrypt data. When the bit of the key is zero, the crypto-system uses operation A and in case the bit is one, operation B is used. Operation A is a very complex calculation while operation B is just a simple subtraction.

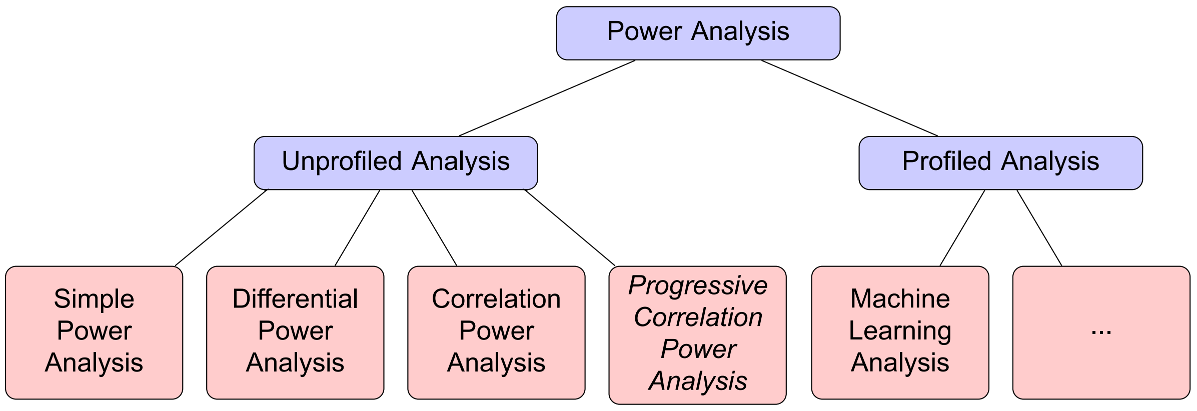

Power Analysis
Unprofiled Analysis
Profiled Analysis
Simple Power Analysis
Differential Power Analysis
Correlation Power Analysis
Progressive Correlation Power Analysis
Machine Learning Analysis
...

Figure 2.8: The classification of different power side channel Analysis. Progressive Correlation Power Analysis has been marked because it was specifically developed in this research.

If an attacker will gather a power trace during the encryption it could retrieve the key used in this system. Operation A is much more complex, so the power consumption is probably higher. But also, since it's more complex, it will take more time. Therefore there are two possible leakage models. One based on power and one based on timing.

2.3.1. Simple Power Analysis

The concept of Simple Power Analysis is like the name suggest rather simple. It works as follows. While a system is performing a cryptographic operation like encrypting, it leaks information about that operation in it's power consumption [55]. To give an example, while performing a BINEX algorithm, it is possible to see the differences between square and multiply operations. When measuring this in a laboratory, this could look like Figure 2.9. However to better explain the concept, this plot was generated.

It clearly shows that the voltage drops when a square operation is performed. When the multiply operation follows, the voltage rises again. This has to do with the fact that the multiply operation requires less power and therefore the voltage drop is smaller. When only the square operation is performed, the voltage remains low.

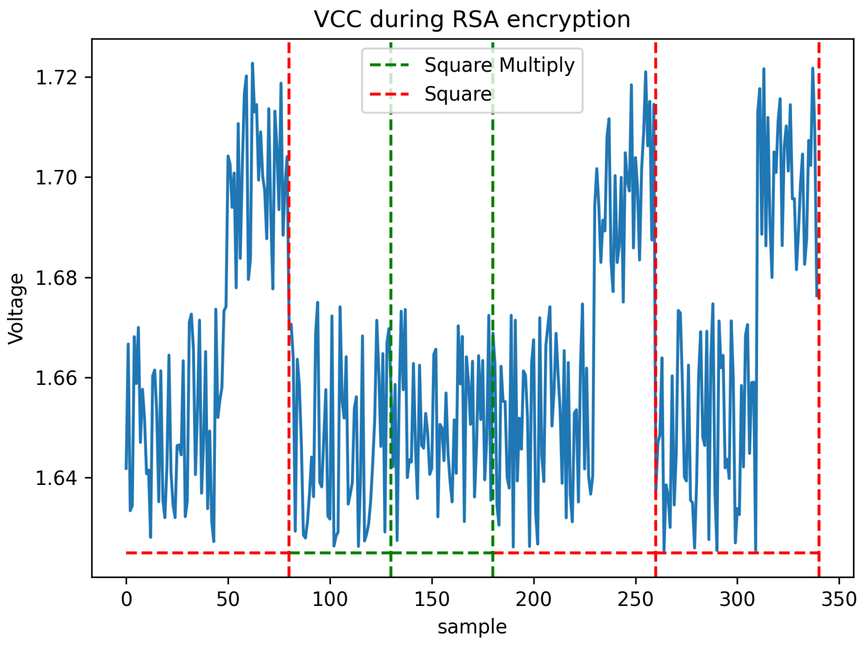


Figure 2.9: Simple Power Analysis on BINEX algorithm. It shows that the square multiply operation is very different from the square operation.

In real world conditions however, the power trace usually is not as clear as the one in this example. Therefore, other methods were developed.

2.3.2. Differential Power Analysis

One of the methods to improve accuracy is Differential Power Analysis [55]. It was first described by Kocher et al in 1998. The basic idea behind is using statistics to improve the results, hence it class statistical power analysis. The attacker needs to have access to the power traces, but also to the cipher text. This method was described for DES, but can be applied for other algorithms.

Step 1
The attacker has to start with gathering the power traces and cipher texts. The original paper suggests that at least 1000 traces are required. However, more traces will lead to a better result.

Step 2
Next, the attacker has to create a selection function called D. The input of D is the sub key and the cipher text. The idea is, that the selection functions splits the traces in two groups. The first group for which the output of the selection function is equal to the target bit and the second group where the output of the selection function is unequal to the target bit. The selection function mimics the behaviour of the crypto algorithm, in this case DES. In other words, all ciphers will be grouped based on the hypothetical sub key.

Step 3
Now that the groups are created, they have to be averaged and subtracted from each other. In other words, $Differentrial\_trace = \overline{group1} - \overline{group0}$. An overview of this step can be seen in Figure 2.10.

Step 4
If the differential trace is more or less flat, the hypothetical sub key is wrong. It means that the groups are sorted in a random way. However, if there is a peak visible, the guess is correct. The peak means that there is a difference in the power consumption of group 1 compared to group 0. The attacker has to test every hypothetical sub key. In the case of DES, the 6 bit sub key results in 64 hypothetical keys.

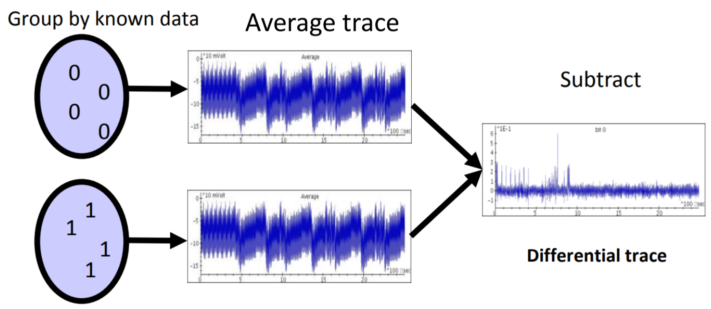


Figure 2.10: This image shows that the traces are grouped by the selection function. After they have been grouped and averaged, they are subtracted to create the differential trace [71].

The differential trace with the highest peak, has the correct guess for the sub key. Because there are only 64 possible sub keys, there is always a correct guess.

Step 5
Because DES consist of multiple rounds, the sub key itself is not enough to reconstruct the key. Therefore, the next round has to be attacker. This means that the process starts over at Step 2. When the attacker has completed all 9 rounds, the complete 56 bit key is retrieved.

2.3.3. Correlation Power Analysis

Another type of statistical power analysis is, is Correlation Power Analysis [30]. It works as follows. This method is based on the correlation of predicted power and the power trace. In order to perform this, four steps are required. Before the analysis can start, first the attacker has to find the place to attack. This might sound very obvious, but not all algorithms or places in a algorithm are suitable for this type of analysis. One of the requirements is that the amount of possible (sub)key is limited to a reasonable amount. Meaning that a computer must be able to process that amount of possible keys in a reasonable amount of time. The attack does not make sense if it is required to process 2^1024 potential keys in order to find the key.

Step 1
The first step is gathering the power traces of a target. If there are d traces that are all N long, the matrix T will have size N by d. This can be seen in Equation 2.17

$$T = \begin{bmatrix} t_{00} & t_{01} & .. & t_{0N} \\ t_{10} & t_{11} & .. & t_{1N} \\ .. & .. & .. & .. \\ t_{d0} & t_{d1} & .. & t_{dN} \end{bmatrix} \tag{2.17}$$

Step 2
During this step, the potential keys are generated. These potential keys are usually sub keys for the intermediate result. In case of the S-box in AES, the potential keys will be all the numbers between 0 and 255. This results in 2^8 potential keys. This key matrix is called S and is 1 by the amount of potential keys (Equation 2.18).

$$S = \begin{bmatrix} k_0 & k_1 & .. & k_{amountofpotentialkeys-1} \end{bmatrix} \tag{2.18}$$

Step 3
In this step, the potential keys are converted to something that is representing the power usage. This can be done for example by using the HW of the intermediate result (the S-matrix). This matrix H has the same size as S and can be seen in Equation 2.19.

$$H = \begin{bmatrix} h_0 & h_1 & .. & h_{amountofpotentialkeys-1} \end{bmatrix} \tag{2.19}$$

Step 4
Now that all the matrices are computed, the correlation analysis can start. The idea is to compare H with T in order to compute r (this correlation matrix). This can be done with Pearson's Correlation Equation (Equation 2.20). In practise this looks like Equation 2.21. It works as follows. One entry of the H matrix gets subtracted by the average of all H.

$$\rho_{x,y} = \frac{cov(X,Y)}{\sigma_x \sigma_y} = \frac{E[(X-\mu_x)(Y-\mu_y)]}{\sqrt{E[(X-\mu_x)^2(Y-\mu_y)^2]}} \tag{2.20}$$

$$G_{i,j} = \frac{(h_{d,i}-\overline{h_i})(t_{d,j}-\overline{t_j})}{\sqrt{(h_{d,i}-\overline{h_i})^2(t_{d,j}-\overline{t_j})^2}} \tag{2.21}$$

2.3.4. Side Channel Analysis by Neural Network

Another way of analyzing traces for a side channel attack is using a neural network. The basic idea of this analysis is learn a neural network the shape of certain keys. In order to do this, it is required to gather a lot of traces with known keys. When the neural network is trained with the known traces and keys, it can be used to find the key in unknown traces.

Although the training of a network requires a lot of computing power, it can be very useful when the information about a crypto system is limited. A neural network usually requires less information to fine tune it's operation then methods like SPA, DPA or CPA.

In order to apply deep learning for a side channel attack, one could use a CNN. In most cases, a CNN used in a profiled attack. That means that the attacker has access to a copy of the crypto system. The copied system should have similar properties and should be controllable by the attacker.

Because the attacker has access, getting traces is relative easy. To make the attack work properly, the attacker should feed the copied system a wide variety of messages and keys to the system. The more variety the traces have, the easier it is for a CNN to generalise.

After the traces have been generated, the traces have to be sliced. Every slice should have a label that is related to the key. For example, a trace of a RSA calculation is sliced to individual square and multiply operations. This slicing is required, but every slice needs to have a label that can relate to the key. In the case of a square and multiply operation, the label is a 1.

Now that the traces and labels are created, the CNN can start to train. When the training has been successfully verified, the attacker can go to the next step, the gathering of the traces of the live crypto system. When this process is finished, the attacker can use the CNN to find out which label belongs to which trace and recover the key.

2.3.5. Timing Analysis

One example for a non physical side channel analysis, is timing analysis. The goal of this attack is to find a relation between the time and the data that is processed. For example, the time a certain operation takes, can spill information about the key. In Algorithm 6 an example of a timing vulnerability is shown. To retrieve the password, the attacker does the following. First, the attacker tries all combinations from aa to az (depending on the character set). If one of the combination contains a character matching with the secret (which is always since all characters are tested), the time it takes to retrieve a access false takes a slight bit longer. This has to do with the break within the check statement. If a character is false, the loop breaks. However, if the first character is correct, the second character is checked. This however requires one more operation.

After the first character is found, the second character can be attacked. This is done by varying the second character until one option is clearly faster then the others. This continues until all the characters are known.

It is also possible to vary more characters at the same time. Although the amount of options grows rapid, the different timings also become bigger between the correct and the incorrect one.

Algorithm 6 Check password with secret

```
for i := 0 to length(secret) do
    if password[i] != secret[i] then
        return false
    end if
end for
return true
```

This example looks very silly and can be easily fixed, for example by first checking every character before returning a value. However, constant timing is something not always that easy to achieve. In 2005 it turned out that an implementation of AES in Opensource Transport Layer Security (TLS) and Secure Sockets Layer (SSL) toolkit [13] (OpenSSL) was vulnerable to a Timing Attack [56] (TiA) [28].

2.3.6. Fault Injection

Another type of side channel attacks are active side channel attacks. These types of attacks focus on injecting or manipulating a crypto system in spilling secret information. A famous example of this is Fault Injection for RSA CRT.

The naive implementation of RSA requires at least the amount of bits in the key. Because this square and multiply operation can take some time, it is considered an expensive computation. There is however a possible shortcut. This requires the CRT. The basic idea of the algorithm is splitting the calculation in two smaller portions. This reduces the amount of memory required and the complexity. By doing this, the algorithm can become almost four times faster [83].

Variable	Meaning
d_P	$d \mod p-1$
d_Q	$d \mod q-1$
q_{inv}	$q^{-1} \mod p$
p_{inv}	$p^{-1} \mod q$
m_1	$c^{d_P} \mod p$
m_2	$c_{d_Q} \mod q$
h	$q_{inv}(m_1 - m_2) \mod p$
m	$m_2 + hq \mod p \cdot q$

Table 2.5: Variable declaration for RSA CRT

The encryption algorithm of RSA CRT is equal to the encryption algorithm of regular RSA. The speedup only happens at the decryption algorithm. In order to do this, some values are pre-computed. These can be found in Table 2.5. With these values the decryption can be done with Equation 2.22. Although this looks more complicated (and it requires more steps), it is a major speedup since the values are much smaller. An example can be seen in Example 2.3.1.

$$\begin{aligned} m_1 &= c^{d_P} \mod p \\ m_2 &= c^{d_Q} \mod q \\ h &= q_{inv}(m_1 - m_2) \mod p \\ m &= m_2 + hq \end{aligned} \tag{2.22}$$

Example 2.3.1: Decrypting a message with RSA CRT

Variable declaration:

$$
\begin{aligned}
p &= 137\\
q &= 131\\
N &= pq = 17947\\
d &= 11787\\
e &= 3\\
m &= 513
\end{aligned}
$$

Encrypting with regular RSA:

$$c = m^e \mod N = 513^3 \mod 17947 = 8363$$

Pre-calculating values for RSA CRT:

$$
\begin{aligned}
d_P &= e^{-1} \mod p-1 = d \mod p-1 = 11787 \mod 136 = 91\\
d_Q &= e^{-1} \mod q-1 = d \mod q-1 = 11787 \mod 130 = 87\\
q_{inv} &= q^{-1} \mod p = 131-1 \mod 137 = 114
\end{aligned}
$$

Decryption of the message:

$$
\begin{aligned}
m_1 &= c^{d_P} \mod p = 836391 \mod 137 = 102\\
m_2 &= c^{d_Q} \mod q = 836387 \mod 131 = 120\\
h &= q_{inv}(m_1 - m_2) \mod p = 114 \cdot (102 - 120 + 137) \mod 137 = 3\\
m &= m_2 + hq = 120 + 3 \cdot 131 = 513
\end{aligned}
$$

Because the numbers involved are much smaller, decrypting with RSA CRT, like in this example [15], is much faster.

$$s = m^d \mod N \tag{2.23}$$

$$
\begin{aligned}
s_1 &= m^{d_P} \mod p\\
s_2 &= m^{d_Q} \mod q\\
s &= (s_1 \cdot q \cdot q_{inv}) + (s_2 \cdot p \cdot p_{inv}) \mod N
\end{aligned}
\tag{2.24}
$$

There is however a downside to RSA CRT. It is much more vulnerable to fault injection [25]. This type of side channel attack is used to change the behaviour of an algorithm. In the case of RSA CRT it works in a rather clever way. The vulnerability in RSA CRT occurs during the signing process. A normal RSA signing happens with Equation 2.23. However, in case of the RSA CRT, it happens with Equation 2.24.

The vulnerability works as follows. When a fault occurs during the calculation of either s_1 or s_2, the signature is incorrect. If this faulty signature is compared by a correct one by subtracting it from the correct signature, p is can be calculated (Equation 2.25). This is done by taking the GCD of N and the difference. How this works, can be seen in Example 2.3.3 [25]. With half of the factorization retrieved, the other variables and finally private key d can be retrieved (as seen in Equation 2.26).

$$
\begin{aligned}
\Delta &= s_{correct} - s_{faulty}\\
GCD(\Delta, N) &= p
\end{aligned}
\tag{2.25}
$$

$$
\begin{aligned}
q &= \frac{N}{p} \\
\phi(N) &= (p-1)(q-1) \\
1 < e &< \phi(N)\text{: e is already public} \\
d &= e^{-1} \mod \phi(N)
\end{aligned}
\tag{2.26}
$$

Example 2.3.2: Signing a message with RSA and RSA CRT

Variable declaration:

$$
\begin{aligned}
p &= 137 \\
q &= 131 \\
N &= pq = 17947 \\
d &= 11787 \\
m &= 513 \\
d_p &= 91 \\
d_q &= 87 \\
q_{inv} &= 114 \\
p_{inv} &= 22
\end{aligned}
$$

Regular RSA signing

$$s = m^d \mod N = 513^{11787} \mod 17947 = 5977$$

RSA CRT signing

$$
\begin{aligned}
s_1 &= m^{d_p} \mod p = 513^{91} \mod 137 \\
s_2 &= m^{d_q} \mod q = 513^{87} \mod 131 \\
s &= (s_1 \cdot q \cdot q_{inv}) + (s_2 \cdot p \cdot p_{inv}) \mod N \\
&= (86 \cdot 131 \cdot 114) + (82 \cdot 137 \cdot 22) \mod 11787 = 5977
\end{aligned}
$$

Example 2.3.3: RSA CRT Fault Injection

Variable declaration:

$$\begin{aligned}
p &= 137\\
q &= 131\\
N &= pq = 17947\\
d &= 11787\\
m &= 513\\
d_p &= 91\\
d_q &= 87\\
q_{inv} &= 114\\
p_{inv} &= 22
\end{aligned}$$

Calculate correct s

$$\begin{aligned}
s &= (s_1 \cdot q \cdot q_{inv}) + (s_2 \cdot p \cdot p_{inv}) \mod N\\
&= (86 \cdot 131 \cdot 114) + (82 \cdot 137 \cdot 22) \mod 11787 = 5977
\end{aligned}$$

Calculate faulty s due to incorrect p_{inv}

$$\begin{aligned}
s &= (s_1 \cdot q \cdot q_{inv}) + (s_2 \cdot p \cdot p_{inv}) \mod N\\
&= (86 \cdot 131 \cdot 114) + (82 \cdot 137 \cdot 5) \mod 11787 = 12416
\end{aligned}$$

Calculate p

$$\begin{aligned}
\Delta &= s_{correct} - s_{faulty} = 5977 - 12416 = -6439\\
GCD(\Delta, N) &= GCD(-6439, 17947) = 137 = p
\end{aligned}$$

Calculate other variables

$$\begin{aligned}
q &= \frac{N}{p} = \frac{17947}{137} = 131\\
\phi(N) &= (p-1)(q-1) = (130)(136) = 17680\\
d &= e^{-1} \mod \phi(N) = 11787
\end{aligned}$$

Both signature calculations result in 5977. However the calculations for RSA CRT are much faster because the exponent is way smaller.

2.3.7. Countermeasures

In order to protect against side channel attacks, countermeasures have been developed. In the case of RSA the most popular public ones are the Montgommery Ladder and the use of blinding. Both methods protect against different types of attacks and can be combined.

Montgommery Ladder

One way of protecting RSA to side channel analysis is using a Montgommery Ladder [52] to encrypt and decrypt the data. The basic idea of the Montgommery Ladder is that both a zero and a one in the key result in a calculation of roughly the same complexity. This makes it much harder to perform a side channel attack. The drawback of this algorithm however, is that is does require more memory and more computing time. Instead of one register with the result, two registers are required. Also, it always requires two multiplications (one multiply and one square) instead of only two when the key bit is one.

The Montgommery Ladder works as follows. If the key bit is a one, first a multiplication of R0 and R1 is done for R0 and then a square operation for R1 with R1 is performed. If the key bit is a zero however, a multiplication is performed with R0 and R1 but this time for R1. Then a square operation is performed for R0 with R0. This can be seen in Algorithm 7. A short example can be seen in Example 2.3.4.

Algorithm 7 Calculate $c = m^e \mod N$

```
R0 ←1
R1 ←m
for i := 0 to length(e) do
   if e[i]=1 then
      R0 ←(R0 * R1) mod N
      R1 ←(R1 * R1) mod N
   else
      R1 ←(R0 * R1) mod N
      R0 ←(R0 * R0) mod N
   end if
end for
c ←R0
```

Just like in a BINEX implementation, it is possible to use the CRT speedup with a Montgommery Ladder. The Montgommery Ladder then is used to calculate m1 and m2 (for example in Equation 2.22) or s1 and s2 in Equation 2.23. However, just as in the case of a BINEX implementation of RSA, the implementation remains vulnerable for Fault Injection [39].

Example 2.3.4: Calculating RSA Montgommery

Variable declaration:

$$m = 4$$
$$e = 5 = 0b101$$
$$N = 221$$

Register declaration:

$$R0 = 1$$
$$R1 = m = 4$$

Encrypting:

$$e[0] = 1 \left| \begin{array}{l} R0 = R0 * R1 \mod N = 1 * 4 \mod (221) = 4 \\ R1 = R1 * R1 \mod N = 4 * 4 \mod (221) = 16 \end{array} \right.$$

$$e[1] = 0 \left| \begin{array}{l} R1 = R0 * R1 \mod N = 4 * 16 \mod (221) = 64 \\ R0 = R0 * R0 \mod N = 4 * 4 \mod (221) = 16 \end{array} \right.$$

$$e[2] = 1 \left| \begin{array}{l} R0 = R0 * R1 \mod N = 16 * 64 \mod (221) = 140 \\ R1 = R1 * R1 \mod N = 64 * 64 \mod (221) = 118 \end{array} \right.$$

Result:

$$c = R0 = 140$$

Just like in the case of regular RSA, the cipher turns out to be 140. To decrypt this, the same procedure can be used with $R1 = c$ and $e = d$:

Blinding

Another possible countermeasure is the use of blinding. Blinding can be applied to two things. It can be used on the message, but it can also be used on the exponent. Blinding the message can be useful if an attacker tries to use a specific message attack [36]. The idea behind this attack is, that by using a specific message, the differences in HW become very large. More on this attack can be seen in Section 3.5. Blinding the exponent however, has another reason. Because the attacker is usually after the private key, the exponent is the main target. If this value is altered, the attacker can only retrieve the altered value.

Blinding works as follows. First a random value r is created. After this, v_f and v_i are created. With these values it is possible to blind the key and the message. An encryption with this technique can be seen in Example 2.3.5 [59]. Although this example shows an encryption, this technique also works for decryption if the public key is replaced by the private key.

As seen in the example, it is possible to use an altered private key. The only thing that has to happen after the encryption, is another multiplication and a modulo. With this technique, the real key is hidden but the result remains equal. To make it even harder, it is possible to use new values for v_f and v_i every time. This is however an expensive operation and therefore it is preferred to pre-calculate these values.

Example 2.3.5: Key and Message blinding

Variable declaration:

$$\begin{aligned} m &= \text{message} \\ c &= \text{cipher} \\ N &= \text{modulus} \\ e &= \text{public key} \\ r &= \text{random value} \\ \phi(N) &= (p-1)(q-1) \\ v_f &= \text{relative prime to N} \\ v_i &= ({v_i}^{-1})^e \mod N \end{aligned}$$

Blinding

$$\begin{aligned} m' &= v_i \cdot m \mod N \\ e' &= e + r \cdot \phi(N) \\ c' &= {m'}^{e'} \mod N \end{aligned}$$

Unblinding

$$c = v_f \cdot c' \mod N$$

3 Attack Methodology

This chapters explains the attack methodology of a thermal side channel attack. During this chapter, four different methods are described. Three of these methods are based on their power side channel analysis counterpart, but have been modified to work with temperature traces. One method however, progressive correlation thermal analysis, has been specifically developed for thermal side channel analysis. It does however also work for power side channel analysis. The newly developed attacks can be also be classified. This can be seen in Figure 3.1

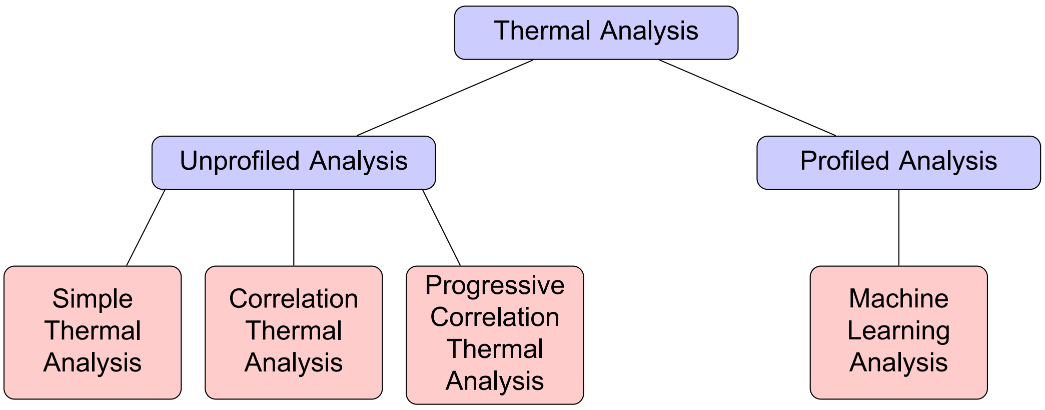

Thermal Analysis
Unprofiled Analysis
Profiled Analysis
Simple Thermal Analysis
Correlation Thermal Analysis
Progressive Correlation Thermal Analysis
Machine Learning Analysis

Figure 3.1: The classification of different thermal side channel Analysis

3.1. Threat Model

For this research, two attack scenarios are considered in order to proof the existence of a thermal side channel attack. The first one considers an unprotected implementation while the second one assumes a protected setup. In both cases it is assumed that the attacker has access to a version of the crypto-device.

Unprotected

The attacker's goal is retrieve the private key that is used in a RSA-decryption. The attacker has eavesdropped the communication between sender (called Alice) and the receiver (called Bob) and has captured the encrypted data. To do this, the attacker places a thermal sensor on Bob's crypto-device. In addition, the adversary controls the clock source of the target device. It means, there is a component that can interrupt the clock of Bob. When Alice sends an unknown message to Bob, the attacker measures the temperature and pauses the clock periodically. After the message has been decrypted, the attacker starts analyzing the gathered traces in order to find the key. This scenario is used for the un-profiled attacks. To summarize this, the unprotected threat model assumes:

- Attacker has access to the messages being decrypted

- Attacker has access to a temperature sensor on the crypto-device
- Attacker can control the clock of the crypto-device

Protected

However, if Bob uses a Montgommery Ladder for the RSA implementation, the scenario changes. In this case, it is assumed that the attacker is also able to send messages to Bob and it is aware of the value of N (the modulo). This is possible because N is considered public. Just like in the previous case, the attacker has captured some encrypted messages from Alice to Bob and wants to decrypt them. However, in this case the attacker measures the temperature while Bob tries to decrypt the attacker's message. This scenario is used for the profiled attacks. To summarize this, the protected threat model assumes:

- Attacker can request decryptions
- Attacker has access to a temperature sensor on the crypto-device
- Attacker can control the clock of the crypto-device
- Attacker has access to a similar device

3.2. Leakage Model

The first step in the side channel attack is to find a place where information about the target leaks. Since the private key is the target, the attack should be focused on the decryption algorithm. Next, the implementation of this algorithm should be analyzed.

The crypto-system runs it's software on a processor. Like any modern day computers, these are MOSFET based. Because the gate is isolated from the source and drain, the MOSFET should in theory only consume power while it's switching (because there is not current flowing from the gate to either the source or drain). However, due to the sub-threshold current between source and drain and the gate leakages, there is always some static power consumption [51]. As a result, the power consumption is the total of the static power consumption and the dynamic power consumption.

For a side channel analysis, the dynamic power consumption is the interesting part. This tells the attacker something about activity. In order to measure this, the attacker could for example measure the current entering the cryptodevice. This tells the attacker something about the activity. This could be done by measuring the voltage over a very well known low ohmic resistor. Not only does this require a resistor in front of the crypto-device but it also requires an accurate and fast way to measure very small voltages.

Another way of measuring the power consumption, is by measuring the Voltage Common Collector (VCC). When a transistors draws current from an infinite source, the current rises while the voltage remains the same. However, in the real world an infinite source does not exist. Especially not when there are a lot of switching transistor at the same time. As a result, the voltage drops. This on the other hand is very easy to measure. It only requires an Analog to Digital Converter (ADC) on the supply line (VCC). Although it is more difficult to get an accurate estimate of the power consumption, it is possible to observe differences between different power loads. When the supply voltage drops by a lot, the processor consumes a lot of current. When the supply voltage drops by a little, the current consumption is low.

Power model

In order to make a proper estimation of the dynamic power consumption during an algorithm, it is necessary to have some kind of power model. If the power model is sufficient, it is possible to give some kind of approximation of the dynamic power consumption and therefore the activity. There are several ways of doing this. One way to approach this, is by looking at the Hamming Weight [60]. The idea is that data with only ones is harder to compute then data with mostly zeros. This concept can be best explained by looking at a serial multiplier.

Listing 1 Serial divider, k bits q divided by k bits d [67]

```
# Registers
#     Rc for counter
#     Rd for divisor
#     Rs for high and remainder
#     Rq for low and quotient

# Init
init:     load      bits(Rd) into Rc
          load      q into Rs and Rq
          load      d into Rd

# Division loop
d_loop:   shift     Rq left 1      # zero to LSB. MSB to carry
          rotate    Rs left 1      # carry to LSB, MOB to carry
          skip      if carry = 1
          branch    no_sub if Rs < Rd
          sub       Rd from Rs
          incr      Rq             # set quotient digit to 1

no_sub:   decr      Rc              # decrement counter by 1
          branch    d_loop if Rc ≠ 0

# Store
store:    store     Rs in remainder
          store     Rq in result
```

The serial multiplier [66] (also known as shift-add multiplier) works in the same way humans multiply numbers. In case the bit is zero, do nothing. In case the bit is one, add it the previous answer. The pseudo code can be seen in Algorithm 8. It is clearly visible that ones in b take more steps then zeros. In this case, $operations = bits(b) \cdot (HW(b) + 2)$ Therefore, if the HW of b is high, it takes more energy then when the HW is low.

To improve this assumption, lets take a look at pseudo assembly code of a serial divider (also known as shift-add division) [67]. This example is extra useful because not all processors have hardware that support division. Especially very large numbers are often handled by software instead of hardware. In Listing 1 the pseudo assembly of the serial divider can be seen. In this implementation is shows that the amount of instructions is very dependend on the HW of q. To be precise, $instructions = 6 \cdot bits(q) + 2 \cdot HW(q)$ excluding the init and store instructions.

Algorithm 8 $c = a \cdot b$

```
for bits(b) do
    if $b_{LSB}$ = 1 then
        c += a
    end if
    a « 1
    b » 1
end for
```

Another way to tell something about the data, is to look at the Hamming Distance. The idea behind this is, that if the HD changes, a MOSFET switches and therefore consumes energy. This power model can be useful when performing side channel analysis on memory. However, for analyzing algorithms, HD is less preferable compared to the HW.

Thermal model

This consumed power has to go somewhere. And since a MOSFET does not light up (in the right conditions), does not move and the EM radiation is also limited, most of the consumed energy is converted to heat. This heat can be measured in the form of temperature and will vary depending on the cooling and load of the processor. If the processor is doing a complex calculation, the processor will produce more heat then when it's doing only a fast and simple operation.

Just like in the power model, the HW and HD can be used to estimate the activity of a processor. If the HW of data entering a serial multiplier is high, it will produce more heat then when the HW is low and thus the temperature will rise (the opposite of the VCC).

There are however a few differences compared to power. First of all, the effect is much slower. This makes measuring on very high frequency useless since the temperature does not change fast enough in order to measure the difference. Another problem, with temperature is the offset. Whilst the VCC is regulated by a voltage controller, the temperature of the environment and the processor are not that strictly controlled.

One way of modeling thermal behaviour is by analyzing the system as a RC-network [44]. This network behaves like a low pass filter with a cut-off frequency somewhere in the kHz range. This poses a problem since most computers tend to run in the gHz range. With a low pass filter (even if it's first order), it is very hard to measure any useful when a system is running at 800 mHz.

Luckily, for a side channel analysis, it is not required to capture every clock cycle. The only thing that has be acquired is a difference between certain operations. If these operations take long enough cycles, they should be visible in the thermal traces.

Another problem that comes with a RC-network is that it functions like an integrator. In other words, temperature accumulates when running a crypto-system. This means that energy from a previous operations will still be visible later in time.

In order to solve this, the crypto-system should stop accumulating energy or cool down very rapidly after each operation. The first can be achieve by periodically stop the clock. Another option could be introducing pauses after each operation. The accomplish the second potential solution, external cooling can be used. For example a high speed fan to cool down the system but also to replace the heated air around the crypto-system.

3.3. Simple Thermal Analysis

In order to find our if a thermal side channel analysis is still possible, first a small test was created to see if it was possible to distinguish two operations.

The code on the Zynq-7 platform (on one of the ARM A9 cores) consisted of two loops without any code inside. After each loop a paused was introduced. This was required for two reasons, first to get rid of the accumalated energy and second to give the logic analyzer some time to transfer the data to the computer. The pseudo code can be seen in Algorithm 9.

During these loops, the temperature was measured by the Xilinx Analog Digital Converter [19] (XADC). Since this test was only done to verify the concept, the XADC was directly connected to a computer with the aid of Vivado [18]. The disadvantage of this setup is that the measuring frequency is greatly reduced (to around 200 Hz). However, for testing the concept this frequency was sufficient.

The differences in temperature of the Zynq-7 processing system whilst performing different operations can clearly be identified (as seen in Figure 3.2). The short loop (blue) is indeed much shorter then the long loop (red). Another thing that is visible from the measurement, is the temperature difference

Algorithm 9 Simple thermal side channel analysis test

```
for i := 0 to N do
   i++
end for
pause
for i := 0 to 2N do
   i++
end for
pause
```

between the short loop and the long loop. Although not very clear, the long loops seem to have a peak temperature approximately 0.5 degree higher then the short loops peak temperature.

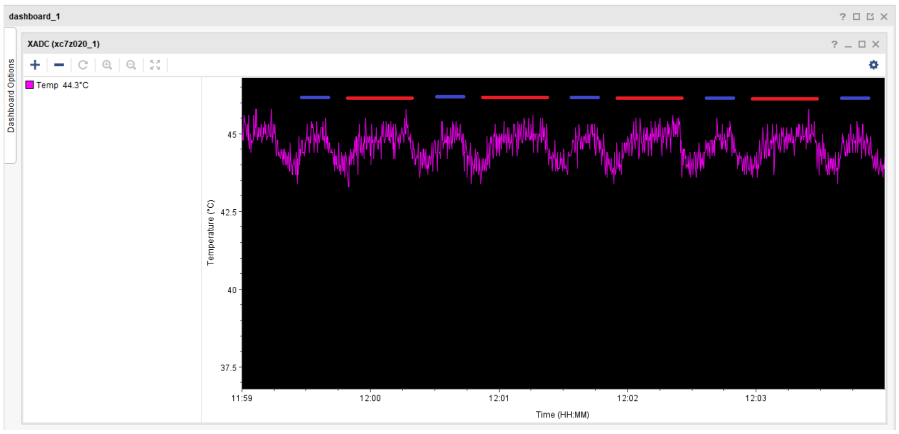


Figure 3.2: Thermals of different for loops. It shows that the longer for loops are longer and become a little hotter.

Binary expansion is known as the most simple implementation of RSA. However, it is very prone to side channel analysis as seen in in Section 2.2.4. This makes it a very suitable target for analyzing the possibilities for the thermal side channel analysis. The goal of the attack is to find the private key d. The public key, like the name suggest is public (and usually 65537) and therefore not an interesting target. As a result of this, the algorithm that is attacked is the decryption algorithm of RSA. The pseudo code that is used for this algorithm can be seen in Algorithm 10.

Algorithm 10 Calculate $m = c^d \mod N$

```
c ←m
for i := length(d) - 2 to 0 do
   $m \leftarrow m^2 \mod N$
   pause
   if d[i] = 1 then
      $m \leftarrow (m \cdot c) \mod N$
      pause
   end if
end for
```

Just like in the example of Figure 3.2, a pause is required to cool down the system and to let the Logic Analyzer transfer data to the computer. The logic analyzer is connected to the pins of the XADC and the sample rate is 50 kHz. This value was acquired by testing different sample frequencies. The logic analyzer captures 32000 samples and then it transfers them to a computer.

3.4. Correlation Thermal Analysis

After all the traces are acquired, they need to be analyzed. It is possible to do this by hand, but there is also a possibility to automate this. This could be done by borrowing the Correlation Power Analysis [60] algorithm and converting this for thermal traces and making it suitable for RSA. The idea behind a correlation analysis is that a predicted energy consumption can be correlated to the real consumption (more information can be found in Section 2.3.3). Another benefit of this method is that it is more robust then STA.

The first step in CTA is finding a place that is suitable for the method. In the case of a BINEX method, this could be in the step were either a square or a multiply is chosen. The advantage of this place, is that there are only two options that can be used. It is either a square or a multiply. The next step would be choosing a thermal model for this operation. Since the attack looks at an algorithm, it makes sense to use the HW instead of the HD.

Next, the HW have to be generated. At first this poses a problem. In order to find the fifth bit, all the previous bits have to be known. Therefore, it is required to test all 2^5 possible bits. This is not a problem if the key is small. However, most RSA keys have a size of at least 1024 bits. This means that it is required to test all these 2^{1024} key combinations (if there is no check if the key is valid). If it would be possible to use every atom in the universe to test a key every second, it would still require $5.95 \cdot 10^{211}$ years [20].

Luckily, the BINEX algorithm has a vulnerability, that removes this problem. One aspect of the BINEX is that is performs a modulo operation at the end of a square or a multiply. This means that the result of $c \cdot c \mod N$ and $c \cdot m \mod N$ is never bigger then N. Also, because N is very big, the change that the result is a small number is pretty low. It is probably somewhere in the neighbourhood of N. Therefore, the modulo acts like a sort of reset. This is very useful, because if the result is always around N before it is either squared or multiplied, the differences in HW are also very limited. The only two possible HW in this case are the one of the square and the one of the multiply. This means that there are only two possible HW.

With this solution, it is possible to perform CTA. Now that the H-matrix is created, it is necessary to gather the traces and pre-process them for correlation. The first thing that is necessary, is the average of all the traces to retrieve $\bar{t}$. Next, it is necessary to calculate the average on all traces individual to retrieve t_j.

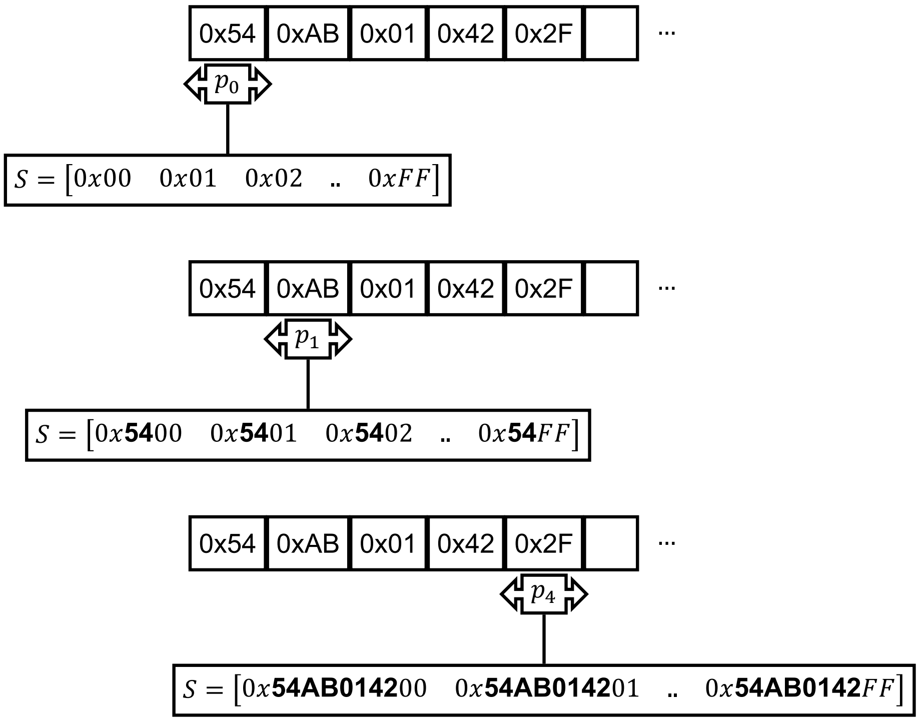
With these values it is possible to calculate the correlation between the two values in H (one for square and one for multiply) and the traces. If the correlation is high, the guess from the value from H is probably correct.

This method works very similar to the correlation power analysis. The only major difference is, is the fact that correlation thermal analysis has a opposite relation with the HW compared to voltage. If the HW is high, the temperature rises, but the voltage drops.

3.5. Progressive Correlation Thermal Analysis

Progressive Correlation Thermal Analysis is based on CPA. The main reason for it's development in the fact that it isn't very useful in most RSA implementations. In most cases, the search space and with that the H-matrix is simply too large.

In order to solve this, the amount of possible options has to be reduced dramatically. One possible way, would be to split the 1024 key in n-bit segments. However, this approach has a problem. The intermediate result is very dependent on the previous intermediate result. In order to solve this, progressive correlation thermal analysis attacks every group of n-bits progressive (as the name suggest).



0x54
0xAB
0x01
0x42
0x2F
···
p_0
$S = [0x00 \quad 0x01 \quad 0x02 \quad .. \quad 0xFF]$
p_1
$S = [0x\mathbf{54}00 \quad 0x\mathbf{54}01 \quad 0x\mathbf{54}02 \quad .. \quad 0x\mathbf{54}FF]$
p_4
$S = [0x\mathbf{54AB0142}00 \quad 0x\mathbf{54AB0142}01 \quad .. \quad 0x\mathbf{54AB0142}FF]$

Figure 3.3: Step 0, Step 1 and Step 4 in Progressive Correlation Thermal Analysis. First the first byte is compared with CTA. Then for the next byte, the first byte is taken into account to create the H-matrix. This new H-matrix is then compared for the second byte with CTA. This continues until the key is completely recovered.

This works as follows. First, the first n-bits are generated to find the HW that fit the used operation. This means that there are 2^{n-bits} of possibilities to check. If there is a HW that matches, the next possible HW are generated with the key of the previous step. This still requires 2^{n-bits} possible HW. For the next n-bits, the next 2^{n-bits} possible HW with the information of the first 2n bits until the key is found. In Figure 3.3 an example is showed with $n = 8$. One thing that should be emphasised is that the amount of entries in the S-matrix remains the same in all the steps. The values inside the S-box will get bigger with each step, but the amount of entries to compare remains equal

The major advantage of PCTA is that it is also suitable for a Montgommery Ladder implementation of RSA (as seen in Section 2.3.7). The biggest difference in the Montgommery implemtation compared to BINEX is the fact that every step results in more or less equal amount of power consumption and therefore temperature difference. This makes a side channel analysis much more complex.

However, more or less the same means that there is some variation. This means, that it should be possible to use PCTA. Although the differences in each intermediate result are small, it might be enough to correlate.

To increase the differences, it is possible to add another concept. The concept of using specific messages in the form of $m = N - 1$ [36]. An example of this can be seen in Equation 3.1.

At first glance, it looks like every round is exactly the same. The answer is always $N - 1 = 76$ in the first round and 1 in the second. However on a closer look, it turns out that the multiplications of every round are slightly different. There are only three possible multiplications.

- $1 \cdot 1$
- $1 \cdot (N - 1) = (N - 1) \cdot 1$
- $(N - 1) \cdot (N - 1)$

This means that there are also only three different HW and that they differ a lot from each other. It turns out that if the key bit toggles, the multiplication remains the same. However, if the next key bit remains equal, the multiplications switch. This not only effects the multiplication, but more important also the modulo operation. Even greater then the multiplication is this operation influenced by the HW of the result (as seen in Section 3.2).

Due to this large difference in HW, it is very suitable for thermal side channel analysis as well as power side channel analysis. The only thing that makes the side channel analysis more difficult, is the fact that it does require information about the previous bit. This is however not a problem for PCTA and PCPA since they were developed for these kind of challenges.

Example 3.5.1: Calculating RSA Montgommery with specific message

Variable declaration:

$$d = 43 = 0b101011$$
$$N = 77$$
$$c = 76$$

Register declaration:

$$R0 = 1$$
$$R1 = c = 76$$

Decrypting:

$$\begin{aligned}
e[0] = 1 &\left| \begin{array}{l} R0 = R0 * R1 \mod N = 1 * 76 \mod (77) = 76 \\ R1 = R1 * R1 \mod N = 76 * 76 \mod (77) = 1 \end{array} \right. \\
e[1] = 1 &\left| \begin{array}{l} R0 = R0 * R1 \mod N = 1 * 76 \mod (77) = 76 \\ R1 = R1 * R1 \mod N = 1 * 1 \mod (77) = 1 \end{array} \right. \\
e[2] = 0 &\left| \begin{array}{l} R1 = R0 * R1 \mod N = 76 * 1 \mod (77) = 76 \\ R0 = R0 * R0 \mod N = 76 * 76 \mod (77) = 1 \end{array} \right. \\
e[3] = 1 &\left| \begin{array}{l} R0 = R0 * R1 \mod N = 1 * 76 \mod (77) = 76 \\ R1 = R1 * R1 \mod N = 76 * 76 \mod (77) = 1 \end{array} \right. \\
e[4] = 0 &\left| \begin{array}{l} R1 = R0 * R1 \mod N = 76 * 1 \mod (77) = 76 \\ R0 = R0 * R0 \mod N = 76 * 76 \mod (77) = 1 \end{array} \right. \\
e[5] = 1 &\left| \begin{array}{l} R0 = R0 * R1 \mod N = 1 * 76 \mod (77) = 76 \\ R1 = R1 * R1 \mod N = 76 * 76 \mod (77) = 1 \end{array} \right.
\end{aligned} \tag{3.1}$$

3.6. Side Channel Analysis by Neural Network

Another way of analyzing would be by using machine learning. The goal of a side channel attack is to classify parts of a trace in such way that the lead to the key. This makes this kind of attack very suitable for supervised learning to solve a classification problem. The model that fits very well to this kind of problem, is a CNN[27]. It is a popular and effective way of (image) classification and recognition [42]. Although machine and deep learning can be very complex, they have one advantage over the techniques like DPA and CPA. They require less manual work and tuning to solve the problem. However, there is no such thing as free lunch. This automation comes at a price. It requires a lot of computing power. Luckily, with the aid of Graphical Processing Unit (GPU)s, this less of a problem nowadays.

A CNN consists of multiple layers and therefore is considered a Deep Learning method. Just like any other deep learning network, it consists of an input layer, hidden layers and an output layer. This can be seen in Figure 3.4. The description of the possible layers can be seen below

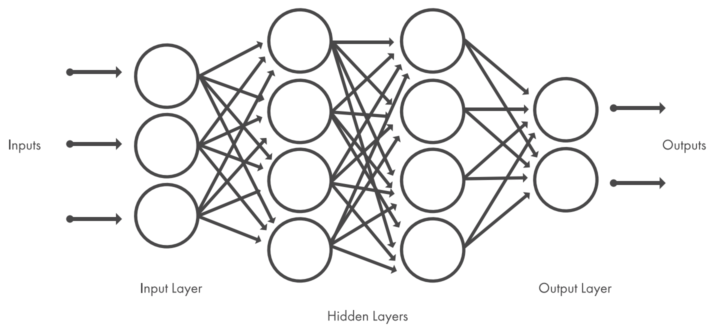


Figure 3.4: The structure of a general Deep Learning Network [42]

- Convolutional layer: like the name suggest, it is the back bone of the CNN. The name of the layer comes from the convolutional filters that are applied here. With these filters, it is possible to extract information. For example, by fitting a number on an image to see how much it is a match [42].
- Activation layer: the layer contains the activation function. This layer is always applied after the convolutional layer and converts the information from the neurons into a single number. A popular activation function is the Rectified linear unit (or ReLu). This function is zero for all negative values but maintains the positive values. The equation can be seen in Equation 3.2 [23].
- Pooling layer: this layer is also known as the down sampling layer. If reduces the pixels of an image (or elements of a trace) to make processing more easy while still maintaining the necessary information. An example of reducing the amount of pixels is by taking the maximum value of a group of pixels. This is called Max Pooling [62].
- Flatten: this layer simply converts the multidimensional data into one dimension. This is usually done right before the Fully Connected Layer [6].
- Fully connected layer: this layer reduces the one dimensional data from the flatten layer down the amount of classifiers [23].
- SoftMax: This layer is converts all the values from the previous layer to a value between zero and one. It simply normalises the data from the previous layer. The equation can be seen in Equation 3.3. Here, x_i is the output of the neuron of the previous layer and N is the total amount of neurons in that layer.

With these layers it is possible to construct a CNN. An example of such a network can be seen in Figure 3.5. However, this is not the complete story yet. One common problem with neural network is over-fitting. Over fitting the problem that a neural network trains so well on a certain data set, that it is not able to recognise anything other then that data set. As a result, it is not able to classify items in a real world environment. To fix this, the neural network has to generalise it's learning capabilities. A few possible solutions for this are:

- Batch normalization layer: originally these layer was introduced to reduce the effect of the randomness of the parameter initialization and input data [49]. However, it turned out Batch Normalization also has other advantages like improving generalization. In this research it is mainly used to reduce over fitting and therefore to better generalize the CNN.
- Added noise layer: this layer, adds noise to the data before it is entering the neuron. As a result, the input data is has added random noise. Usually added noise makes processing data harder, but in this case that is the desired effect. As a result, the neural network is able to classify slight variations of one image instead of just one image [11].
- Dropout: this technique randomly disables neurons in a layer. This prevents the network of completely relying on a single neuron. By also using the neurons around the most important

neuron, the network is better able to generalize data [77]. It is however, not recommended to combine this technique with Batch Normalization as the improvement of Dropout is mitigated or even removed [49].

- L1 and L2 Norm: these values limit the size of the input weights. This is done because large weights leads to poor performance as large weights make a neural network very dependent on a limited amount of nodes. By limiting the weights, the neural network dependents on more nodes to classify [23].

$$f(x) = \begin{cases} \text{x} & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

$$\sigma(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{N} e^{x_i}} \tag{3.3}$$

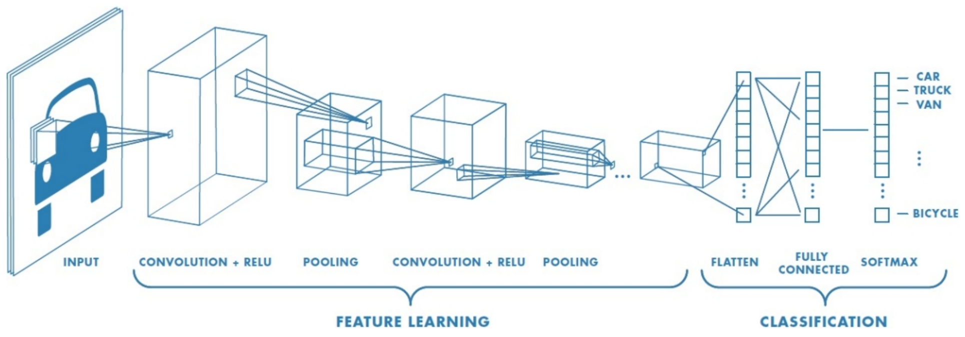


Figure 3.5: The structure a possible convolutional neural network [42]. It shows how an image of a car is classified.

The first step in using a CNN for a side channel attack, is gathering traces. Because the CNN needs to be trained on known traces, it is required to have traces with a known key and message. This makes using a CNN a profiled attack. One way of gathering these traces, would be by using the same device and software as the target to recreate the environment. For simplicity in this research, the target and the copied target (also known as the template device) are the same device.

After the traces have been gathered, they will be pre-processed to remove noise. This can be done in various ways. For this research a high pass filter and a Fourier Transform have been used. More details about this can be seen in Section 4.6.

The next step in the process is determining the labels. Because of the use of a CNN as a classifier, it is necessary to have a finite set of known labels. In the BINEX implementation for RSA it makes sense to look into two possible label structures:

- Labels: *square* and *multiply*
- Labels: *square square*, *square multiply* and *multiply square*

The first method has as advantage that it is the most simple approach. It just requires the have traces with only one operation. The second method however has a statistical advantage. The first method will test every trace for a zero and a one, even if this is not possible. For example, there will never be two multiply operations in a row. This is not the case for the second method. This automatically eliminates impossible possible at the cost of some complexity.

After the labels have been created, the training should be gathered. This can be done by running RSA on the crypto system with known keys. Another option would be to gather the traces from Bob it's crypto-system while it's performing encryption operations. This also generates a known trace label combination because the key and the modulo are both public. However, this does make it harder to get a uniform training set. Therefore it is better to use a template device.

For this research, the template platform and the attack platform were the same device. To train the network, first a lot of temperature traces were captured from the platform. Next another data set was created with new traces and a unknown key. This was used to validate the system. In order to get a correct key, multiple measurements were done on the unknown key. This makes it possible to perform a vertical attack [26].

The network that was used was based on an existing network [23] and modified to fit the purpose. This CNN used three convolutions and used batch normalization to reduce over fitting. More details about this can be seen in Section 4.6.

4 Validation and Results

This chapter validates the techniques from the previous chapter and explains all the challenges that have occurred. The techniques that are validated in this chapter are, Simple Thermal Analysis, Correlation Thermal Analysis, Progressive Correlation Thermal Analysis and Deep Learning Analysis. Progressive thermal analysis was also converted to power to validate it's behaviour in the power domain.

4.1. Setup

The main platform to test if a thermal side channel analysis is possible is the PYNQ-Z1 Field Programmable Gate Array (FPGA). The board can be seen in Figure 4.1 and the features that make this platform suitable are:

- Build in sensors and a fast ADC
- Soft and hard core processors available for testing
- Flexible platform for running code and for developing hardware
- Cheaply available

The PYNQ board creates a lot of flexibility. It allows to run bare metal c(++) code on one or both ARM-A9 cores, but it also allows to run a soft processor core like the MicroBlaze [10]. Another advantage is the XADC [19]. This ADC is connected to a network of sensors and is able to measure up to 1 MSPS. The most important sensors for this research are:

- Temp (0x00), on die temperature
- VCCint (0x01), voltage of power supply of Programmable Logic
- VCCPint (0x0D), voltage of power supply of the ARM-core

The XADC writes raw data. It is possible to convert this value to for example degrees Celsius with Equation 4.1 [19] (Equation 4.2 [19] is used for converting raw values to voltage). However, the conversion will introduce some noise due to rounding and makes the resolution harder to see. Therefore, all side channel analysis were performed on the raw value of the XADC. An overview of the FPGA can be seen in Figure 4.2

$$T = \frac{Raw\,Temperature \cdot 503.975}{4096} - 273.15 \tag{4.1}$$

$$v = \frac{Raw\,Voltage \cdot 3}{4096} \tag{4.2}$$

Although the XADC measures at a high frequency, it's power consumption compared to the rest of the system is relative low. Figure 4.3 shows the relative energy consumption compared to the rest of the system in case of a system with an ARM-core and in case of a MicroBlaze soft core processor. It shows that in both cases, the energy consumption of the XADC negligible compared to the other consumers.

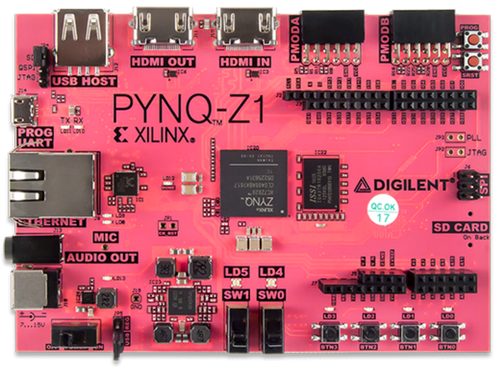


Figure 4.1: PYNQ-Z1 board [85]

From Figure 4.3a is clearly shows that the Microblaze soft core requires much less power then the ARM-core (Figure 4.3b). As a result of this, the temperature differences that were generated by the Microblaze softcore were much smaller compared to the temperature differences from the ARM-core. Another explanation for this is the fact the dynamic power consumption of the ARM-core is much higher then that of the Microblaze. This due to the fact that the Microblaze runs on the Programmable Logic of the FPGA. Due to the lower energy consumption and the lower dynamic energy consumption, it was much harder and in some cases impossible to perform a thermal side channel attack. Therefore, it was chosen to only perform the thermal side channel attack on the ARM-core.

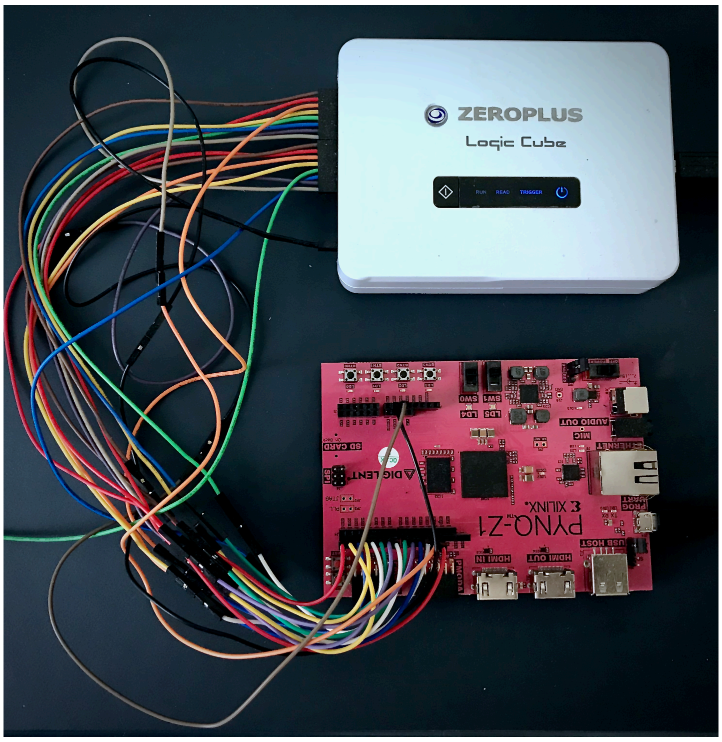


Figure 4.4: The measurement setup with logic analyzer. The wires connect the XADC parallel to the logic analyzer to get a high bandwidth with creating as minimal noise as possible

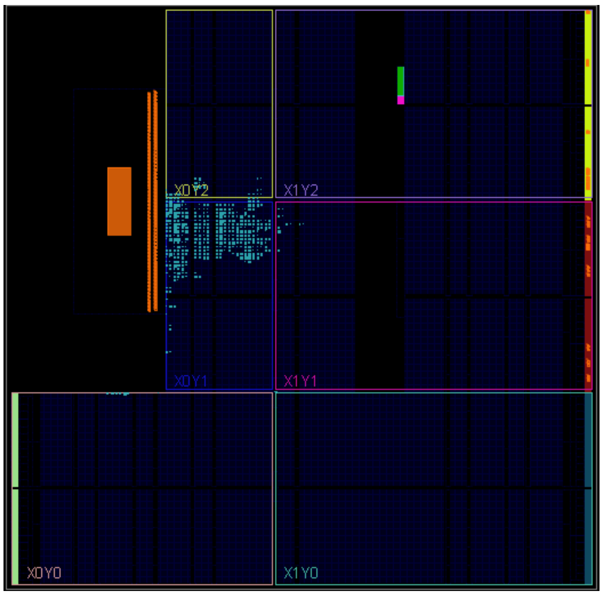


Figure 4.2: The area in green shows where the XADC is located on the FPGA. The block around the orange blocks is the locations of the ARM core. It also shows that most of the Programmable Logic (PL) is empty.

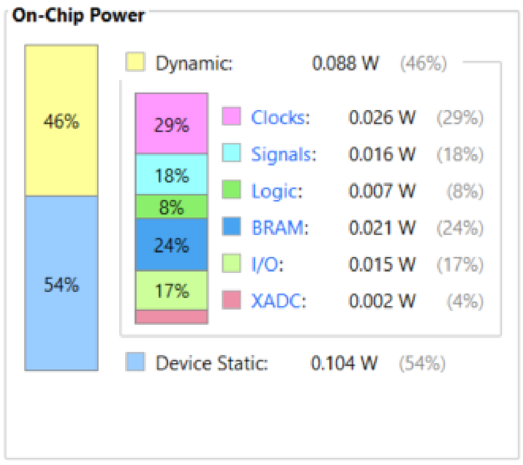
In order to retrieve the data from PYNQ-Z1 without creating to much noise, an external devices is used to gather the sensor data. This is done by a logic analyzer. This logic analyzer is able to gather 32000 traces on 16 channels before it has to transfer it's data to a computer. The logic analyzer used in this setup is a Zeroplus Logic Cube (LAP-C 16032). An overview of the total setup can be seen in Figure 4.5. A picture of the setup can be seen in Figure 4.4

The Logic Analyzer is controlled via the Sigrock CLI [73]. This Command Line Interface allows for automated measurements with the aid of a bat-script (on Windows). After the logic analyzer captured all of it's samples, it transfers the data to the connected computer in a txt-file.

Because the logic analyzer creates a lot of files, it takes a long time to process the actual information. Therefore it makes sense to pre-process the measurements. This is done with a Python script called convert.py. This script first reads the individual txt-files, removes any reading errors and compresses the data to one large binary file. This last step is the most important one and reduces the read-time to about one minute depending on the computer.

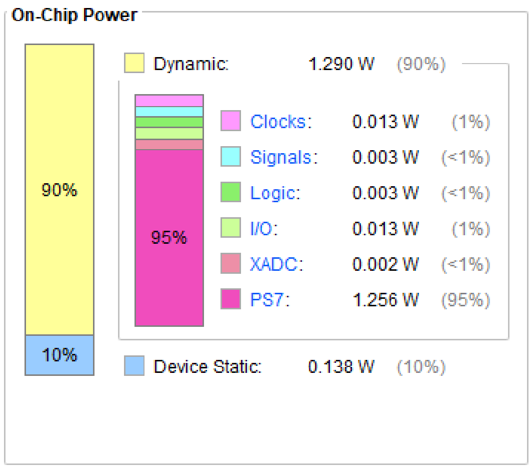
4.2. Simple Thermal Analysis

One problem that occurred during the measurements, was that the PYNQ-Z1 board started to heat up. This made it harder to differentiate between the different types of operations and lengthened the necessary cool down period. In order to minimize this, a fan was added to the setup. This reduced the overall temperature by almost 10 degrees Celsius and decreased the noise level. The effect of the fan can be seen in Figure 4.6.


On-Chip Power

🟨	Dynamic:	0.088 W	(46%)
🟪	Clocks:	0.026 W	(29%)
🟦	Signals:	0.016 W	(18%)
🟩	Logic:	0.007 W	(8%)
🟦	BRAM:	0.021 W	(24%)
🟩	I/O:	0.015 W	(17%)
🟥	XADC:	0.002 W	(4%)
🟦	Device Static:	0.104 W	(54%)

(a) Energy Consumption XADC compared to the rest of MicroBlaze system


On-Chip Power

🟨	Dynamic:	1.290 W	(90%)
🟪	Clocks:	0.013 W	(1%)
🟦	Signals:	0.003 W	(<1%)
🟩	Logic:	0.003 W	(<1%)
🟩	I/O:	0.013 W	(1%)
🟥	XADC:	0.002 W	(<1%)
🟪	PS7:	1.256 W	(95%)
🟦	Device Static:	0.138 W	(10%)

(b) Energy Consumption XADC compared to the rest of ARM-core

Figure 4.3: Comparison of energy consumption of Microblaze and ARM, it shows that in both cases the power consumption of the XADC is negligible compared to the rest of the system.

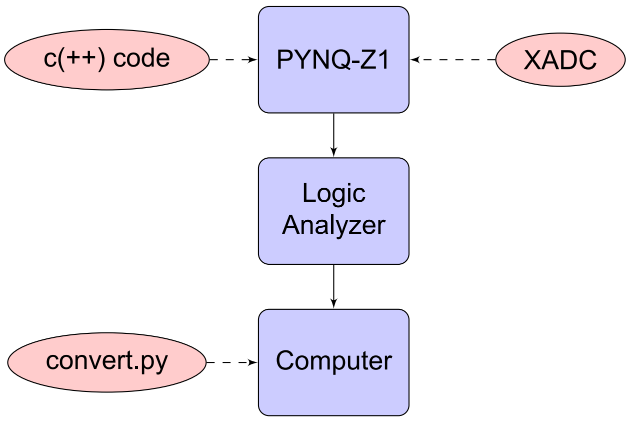
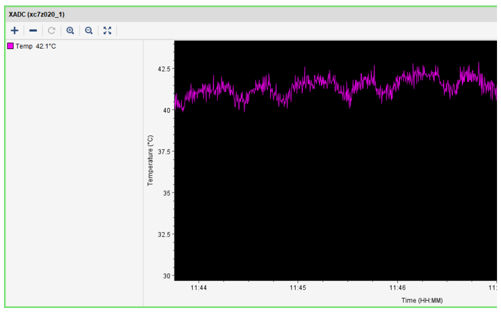
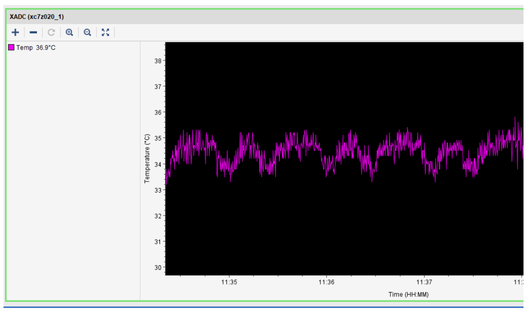


Figure 4.5: The PYNQ-Z1 runs the C-code while a logic analyser is taking the values from the XADC



(a) Temperature trace of un-cooled setup running the short loop, long loop c-code



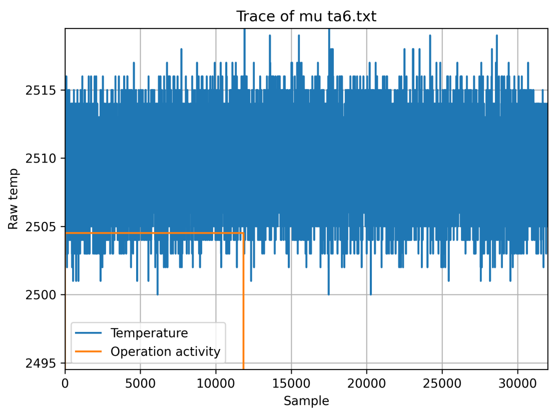
(b) Temperature trace of cooled setup running the short loop, long loop c-code

Figure 4.6: Comparison of cooled and un-cooled setup. The cooled setup not only runs at a lower temperature, but it also cools down faster which makes a side channel attack easier.

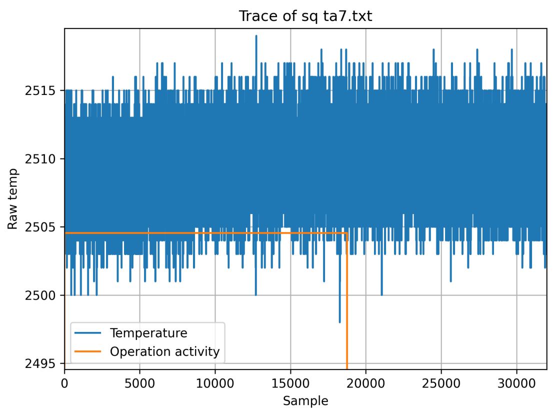
Since thermals change very slow compared to voltages, the sample frequency can be much lower. The optimal sampling frequency was established by two parameters. First the results were analyzed by hand to see if it was required to up or downgrade the frequency. Next the resulting temperature traces were inserted in the CNN from Section 4.6. First the measurements were done at the highest possible sample frequency of 1 MSPS. The sample frequency was lowered until it reached 10 kHz. The traces that received the highest accuracy were 50 kHz and 100 kHz. Because temperature takes some time to react, it takes a approximately 12000 samples at 50 kHz to respond. Therefore a longer measuring time (and with that a lower sampling frequency) was preferred to increase the accuracy even more. This resulted in a sampling frequency of 50 kHz.

The result of the traces with cooling and the correct sample frequency can be seen in Figure 4.7. Although there is a difference visible between the multiply and square operation, it is not very clear. Especially when compared to power, performing a manual analysis on this data can be tedious. Also, analyzing over a 1000 plots by hand is a rather tedious job. Therefore it is necessary to automate this and make it more accurate.

The first technique that was considered was to convert DPA to the thermal domain. However, during the research it became clear that temperature tends to have a lot of drift compared to voltage. This is a problem for DPA since it requires both groups of traces to have similar offsets. If this is not the case, the differential trace might suggest a correct guess, whilst the difference actually comes from different temperature drift. By removing offset this problem can be somewhat mitigated, but it makes it less suitable for thermal side channel analysis. For CPA and CNN this is less of a problem. The following sections explain more about these various methods that have been tested and verified.



(a) Temperature trace of a cooled setup performing a multiply operation

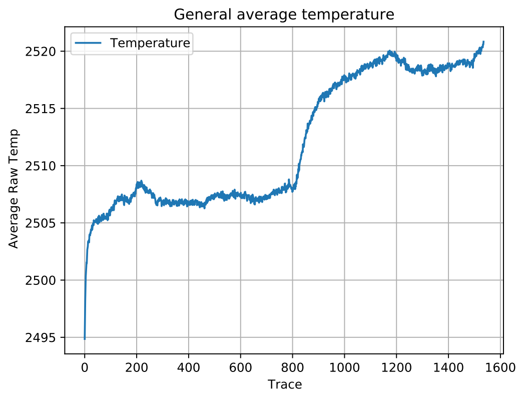


(b) Temperature trace of a cooled setup performing a multiply operation

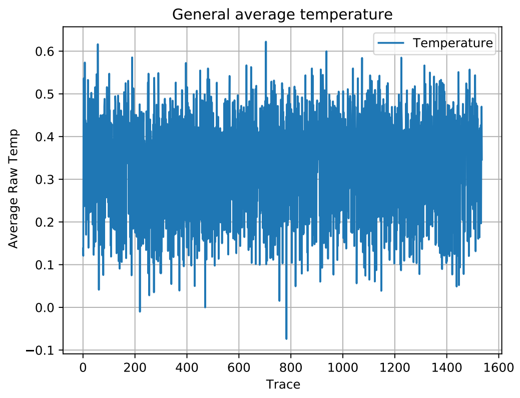
Figure 4.7: Comparison of square and multiply operation for Simple Thermal Analysis in a BINEX implementation of RSA. It shows that the square operation heats up more then the multiply operation.

4.3. Correlation Thermal Analysis

When performing CTA, the first thing that is noticeable is the drifting temperature over time. This is clearly visible when the averages of every trace are plotted. This can be seen in Figure 4.8a. This is a problem because the HW is not drifting. To solve this, either the HW has to be compensated or the averages of the traces.



(a) Temperature drift of different traces



(b) Removed offset due to drift from temperature traces

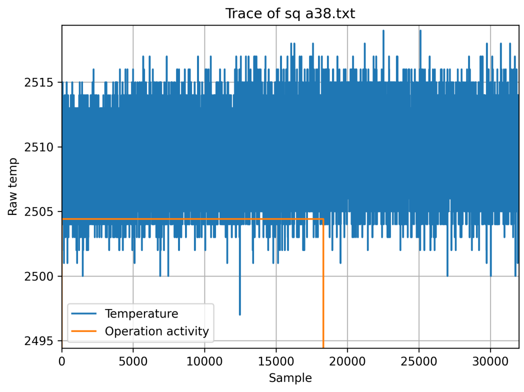
Figure 4.8: Comparison of two temperature traces with BINEX implementation with and without offset removal. It shows that the temperature drift over time can be compensated by removing the average of the first part of the trace.

One way of solving this, could be by subtracting the average of each trace of each trace. This is a method is used to remove noise [82]. However, this is not suitable in this case because the average of each trace is used to correlate against the HW. This problem becomes visible in Equation 4.3.

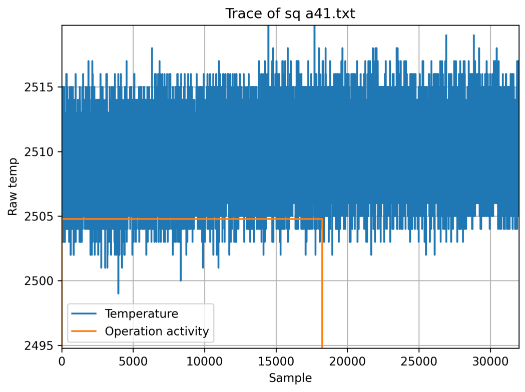
$$\begin{aligned}
T &= \begin{bmatrix} t_0 & t_1 & t_2 & .. & t_n \end{bmatrix} \\
\overline{T} &= \frac{1}{n}\sum_{i=0}^{n} t_i \\
T_{\text{reduced offset}} = T - \overline{T} &= \begin{bmatrix} t_0 - \overline{T} & t_1 - \overline{T} & t_2 - \overline{T} & .. & t_n - \overline{T} \end{bmatrix} \\
\overline{T_{\text{reduced offset}}} = \frac{1}{n}\sum_{i=0}^{n} t_i - \overline{T} = \frac{-n\overline{T}}{n} + \frac{1}{n}\sum_{i=0}^{n} t_i &= -\overline{T} + \overline{T} = 0
\end{aligned} \tag{4.3}$$

Another way of getting rid of the offset due to temperature drift, is by taking the first value of the trace and subtracting it. However, due to noise this can be tricky. To make this more robust, it is better to take the average of the first n-samples and subtract that value from the trace. This reduces the noise. This technique is based on the auto-zero amplifier [38]. Instead of charging a capacitor in the sampling phase, the first values are accumulated and normalized.

In Figure 4.9 it shows that this is possible. The orange line indicates the activity of the processor. In both Figures, the activity takes around 18000 samples. The temperature however, only starts to rise somewhere between 10000 and 15000 samples (due to the integrator effect of the thermal behaviour). This means that the first 10000 samples can be used to reduce the offset.



(a) Temperature trace (a) of square operation in BINEX



(b) Temperature trace (b) of square operation in BINEX

Figure 4.9: Two temperature traces that show the rising temperature during an operation.

In Figure 4.8b the drifting offset is removed by subtracting the average of the first 12000 samples. It also shows that the average of each trace is unequal to zero and therefore it is possible to use it for correlation.

Next, the H-matrix has to be created. Since there are only two possible operations, it is either square or multiply. Because the modulo operation is the most expensive operation, it makes sense to calculate the HW of the result of the square or multiply operation to use for the H-matrix. To make a proper estimation of these values, a simulator was created.

The first version of the simulator simply was the C-code that was used on the Pynq-z1 to run RSA calculations. With the aid of a few print statements and a python wrapper, the HW of the square and multiply operations were captured. The input for this C-code were some random key pairs of the same size that was used on the FPGA. Although this was a very accurate simulation, it was relative slow. Therefore, the c-code was converted to python to make the system integrate better and speed up the process.

After 10 different key pairs, the average HW for the result of the square operation was 922 and 461 for the multiply operation result. Although the simulator gave some insight in the different HW, it is not necessary in case the BINEX implementation in combination with CTA. The reason for this is the fact that the HW and the trace values are both normalized. As long as the ratio between square and multiply HW is around 2:1, the attack will work. It might sound that because of this, the simulator is obsolete. This is not the case. The simulator is very useful the get insight in the behaviour of an implementation but also it plays an essential part in PCTA and PCPA. However, for this implementation a Montgommery simulation was created.

Now that the H-matrix and the T-matrix are both generated, it is possible to calculate the correlation matrix r. To do this, Equation 2.21 is implemented. For each trace, there are two possible values for r calculated, r_{square} and r_{multiply}. If $r_{\text{square}} \geq r_{\text{multiply}}$ the key guess is 1. Else the key guess is 0. The pseudo code can be seen in Algorithm 11.

If the key_guess list is compared with the labels of the traces, the result are impressive. On a measurement with 9 traces, the correctness varies between 92 and 95 percent. Not only that, most errors also differ per trace. Meaning, that it is possible to reconstruct the complete key. The only place were all results were wrong were for the first two bits. This makes sense because input of the square and multiply is not limited by the modulo yet. Also, this is not a problem to reconstruct the key. Because the location of the faulty bits is known and it only encounters for the first few bits, the amount of brute forcing is very limited. With only 8 traces in a vertical attack and a correction for the first two bits, it was possible to 100 percent reconstruct the original key. The results can be seen in Table 4.1. The error histogram can be seen in Figure 4.10. Although the errors occurs everywhere uniform along the

Algorithm 11 Correlation Thermal Analysis for a BINEX implementation of RSA

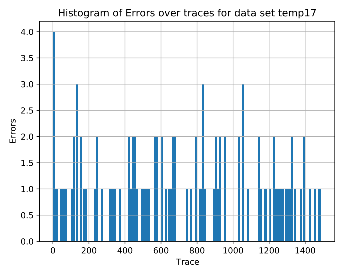
```
for i := 0 to length(trace) do
    $r_{\text{multiply}} \leftarrow$correlation($h_{\text{multiply}}$, trace[i])
    $r_{\text{square}} \leftarrow$correlation($h_{\text{square}}$, trace[i])
    if $r_{\text{square}} \geq r_{\text{multiply}}$ then
        key_guess[i] $\leftarrow$1
    else
        key_guess[i] $\leftarrow$0
    end if
end for
```

traces, it is clear that the first bits are always wrong. The combined error histogram can be seen in Figure 4.11. The errors seem to appear pretty uniform except for the first bits.

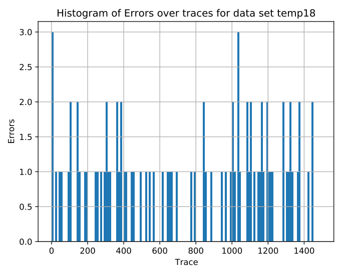
Although this result seems very impressive, it should be mentioned that this was tested with a perfect scenario. The traces are all perfectly sliced, the algorithm on the crypto-system is known and measurements were obtained from an internal sensor. Nevertheless, the goal of this research was to prove the possibility of a thermal side channel attack and with this example it does. When this method is compared with traditional CPA, it shows that the performance of CTA is not yet on the same level as CPA. When the same setup is used to measure power traces, CPA only requires one trace to fully retrieve the key (with compensation for the first bits).

	Correct
Key guess list of trace 14	93.1 %
Key guess list of trace 15	92.8 %
Key guess list of trace 16	92.7 %
Key guess list of trace 17	93.0 %
Key guess list of trace 18	94.4 %
Key guess list of trace 19	92.8 %
Key guess list of trace 20	94.3 %
Key guess list of trace 21	93.6 %
Key guess list of trace 22	92.9 %
Majority voting on trace 14:21	100 %

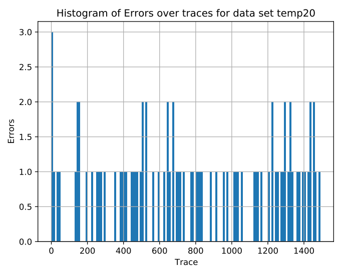
Table 4.1: Results of CTA on a BINEX implementation of RSA



(a) Error histogram of the guess list of temp17



(b) Error histogram of the guess list of temp18



(c) Error histogram of the guess list of temp20

Figure 4.10: The error histogram of the guess list of various traces with CTA

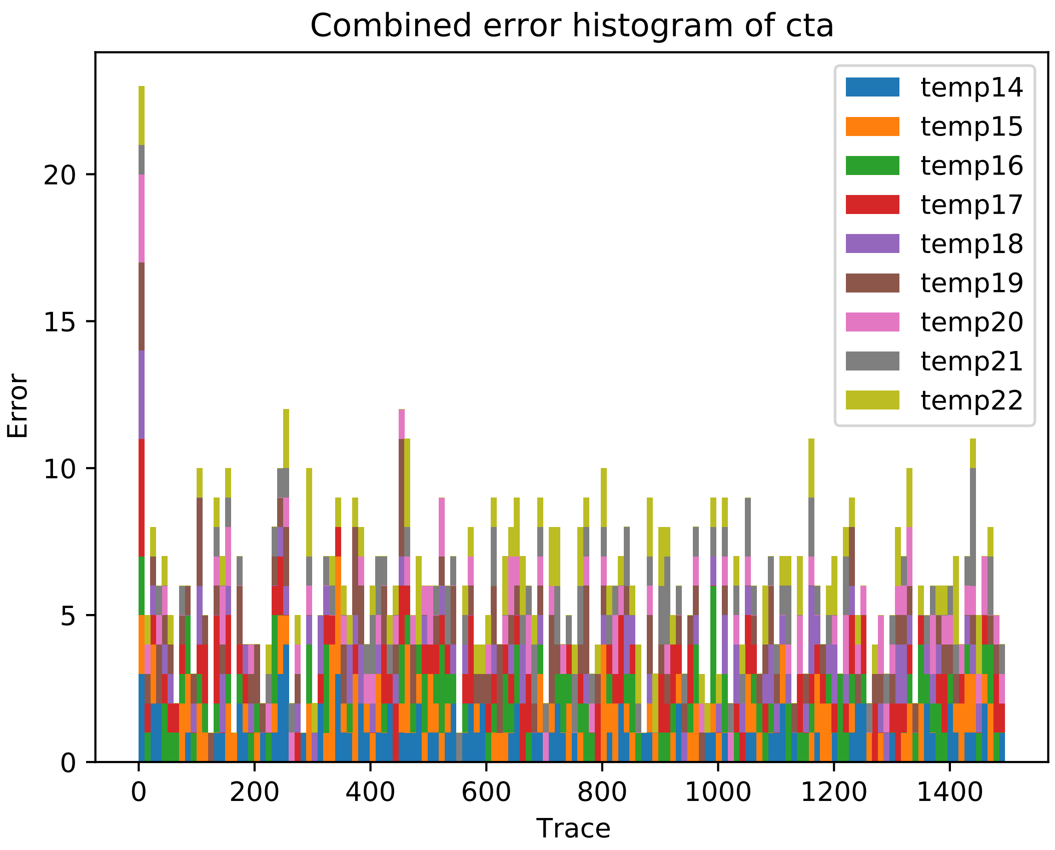
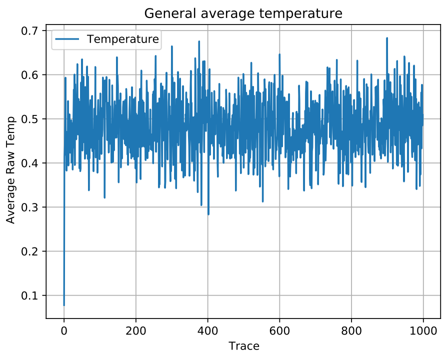


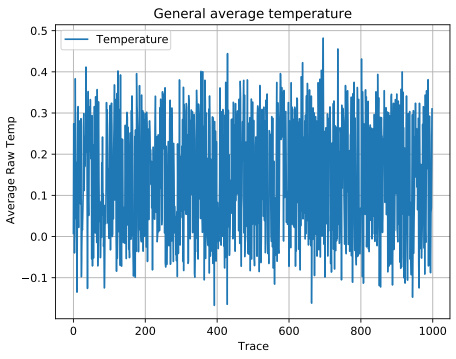
Figure 4.11: Error histogram of CTA of trace 14 until trace 22. It shows a pretty uniform error distribution except for the first bits. This is due to the fact that the first bits have a much smaller HW. Luckily the position of these first bits is known and can therefore be easily compensated

Because CTA turned out to be a success for BINEX it was also tested on a Montgommery implementation of RSA. After the traces were gathered, the first challenge occurred, how to construct the H-matrix. The problem here was that the zero and the one both give a very similar based on HW. This made it impossible to distinguish a zero from a one. This is visible in Figure 4.12a. Compared to the averages of the BINEX implementation in Figure 4.8b it clearly shows that almost all values are between 0.4 and 0.6 while the values of the BINEX implementation are between 0.1 and 0.6. This shows that variance is much smaller and therefore it is much harder to differentiate. The correctness of the key_guess list eventually resulted in 49.9 %. This might seem higher then expected. However, the key that was guessed by CTA only consisted of ones.

https://www.overleaf.com/project/5f18114deaa42a0001320d1a



(a) Temperature trace with removed offset of a Montgommery implementation of RSA with a random value for c



(b) Temperature trace with removed offset of a Montgommery implementation of RSA with $c = N - 1$

Figure 4.12: Comparison of two temperature traces with Montgommery implementation with different values for c. It shows that the average temperature per trace is much larger in the trace with $c = N - 1$. The average temperatures are roughly between -0.1 and 0.4 ($\Delta = 0.5$) instead of between 0.4 and 0.6 ($\Delta = 0.2$). This makes side channel attacks much easier in the case of $c = N - 1$.

In order to try to improve the results, the cipher $c = N - 1$ entered and tested. Although the H-matrix did end up with more distinguishable values, it was still not enough to reconstruct the original key. The main problem that occured here, was the fact that the HW of the every operation depends on the previous key bits. Because CTA only looks at one bit at the time, it was method was doomed to fail. In Figure 4.12b it does show that the averages have a bigger difference compared to the measurement without the specific message. The values are between -0.1 and 0.4 for $c = N - 1$ instead of between 0.4 and 0.6.

4.4. Progressive Correlation Thermal Analysis

To tackle the problem from CTA with a Montgommery Implementation of RSA, PCTA was developed. The first thing that was tested was an attack on a temperature trace without the use of a specific message. It turned out that there was too much noise to differentiate between the small differences in HW of the intermediate answers. An example of the H-matrix combined with the average value can be seen in Equation 4.4.

$$\overline{h} = 61.48$$

$$H = \begin{bmatrix} .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ 61 & 67 & 68 & 67 & 53 & 53 & 55 & 60 & 68 & 56 \\ 61 & 67 & 68 & 67 & 53 & 53 & 55 & 60 & 68 & 59 \\ 61 & 67 & 68 & 67 & 53 & 53 & 55 & 59 & 62 & 65 \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \end{bmatrix} \tag{4.4}$$

To reduce the noise from the traces, the specfic message attack is sueed. This increases the differences in HW and reduces the possible options. An example of the H-matrix and average h value can be seen in Equation 4.5.

$$\overline{h} = 38.0$$

$$H = \begin{bmatrix} .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ 49 & 27 & 27 & 27 & 49 & 49 & 49 & 49 & 27 & 27 \\ 49 & 27 & 27 & 27 & 49 & 49 & 27 & 49 & 27 & 27 \\ 49 & 27 & 27 & 27 & 49 & 49 & 27 & 49 & 27 & 27 \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \end{bmatrix} \tag{4.5}$$

The first tests with PCTA worked well for the first sub-keys. However, whenever the sub key contained a lot of zeros and ones, the algorithm was not able to make a correct guess. The solution for this was rather easy. If a sub key contains only zeros, $\bar{t}$ is not the average power. It is the average power of only zeros. Because the algorithm tries to find differences, it will try the find a difference between each bit (which is zero). To fix this problem, a larger window to calculate the average is required. One way to fix this, would be to look at more bits at the time. This is however very unpractical because a window of 50 bits would require a massive amount of computing power. Therefore the window that is used to calculate $\bar{t}$ is different from the amount of bits that is compared.

A larger window to obtain $\bar{t}$ has some disadvantages. First of all, the it is slower. But more important, because the temperature changes over time, it is harder to remove the offset correct. If the window is too small, it is not able to get a uniform average (because only ones or only zeros can mess up the average). On the temperature traces for this research a window between 30 and 40 to calculate $\bar{t}$ and a window of 10 bits to compare turned out to be optimal.

In contrast to CTA on a BINEX implementation, PCTA is not able to corrects itself. When a bit is guess incorrect, the next bits will also be incorrect. Therefore it makes sense to measure the performance based on the sequence of correct bits. The results of this method can be seen in Table 4.2. Because one error can break the streak, the variation between every trace is huge (as seen in Figure 4.13). An example of some temperature traces of a Montgommery implementation of RSA can be seen in Figure 4.14.

	Sequence of correct bits
Key guess list of trace 28	257
Key guess list of trace 29	97
Key guess list of trace 30	96
Key guess list of trace 31	47
Key guess list of trace 33	108
Majority voting on traces above	1024

Table 4.2: Results of PCTA on a Montgommery implementation of RSA with $c = N - 1$ on a 1024 bit key

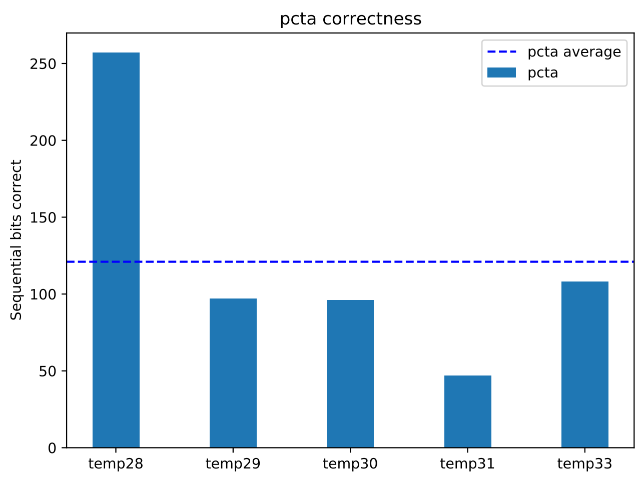
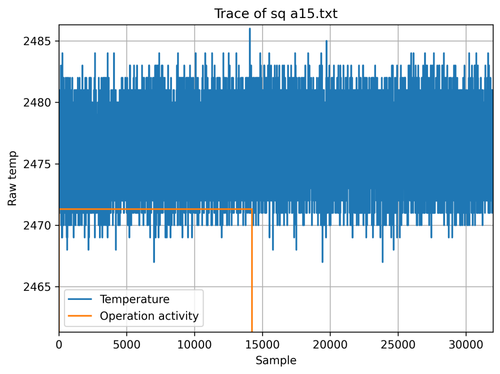
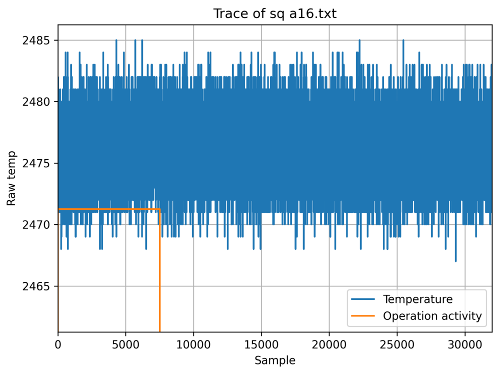


Figure 4.13: Sequence of correct bits using PCTA. With one trace it is not possible to fully retrieve a key. However, with five traces and majority voting it is possible

To reduce this problem, more traces are needed. There are various options to increase to accuracy. The first method would be simply adding traces together [33]. This increases the differences between low intensity and high intensity operations. However, for this to work properly, all traces should start at the same time, be perfectly aligned and most importantly have the same offset. The last property is a problem in the case of temperature because the offset is not completely removed and is not as



(a) Temperature trace of a Montgommery implementation of RSA with $c = N - 1$, long operation



(b) Temperature trace of a Montgommery implementation of RSA with $c = N - 1$, short operation

Figure 4.14: Comparison of two temperature traces with Montgommery implementation. It shows that the temperature traces contain a lot of noise and are hard to distinguish with the naked eye.

static as in the case of voltage. If the traces would simply be added together, the trace with the highest temperature would have the most influence even if it's wrong. Therefore a different approach was used.

Instead of adding all the traces together, the correlation result of every trace was used. This meant that every trace would have an equal vote during the majority voting step. Because the previous bits are so important for this attack, the majority voting is done during every step.

There are two ways majority voting is possible. The first method is to compare all n-bit answers at once. The sub key that is the most common, is probably the correct one. The other method that can be used it to apply majority voting on bit level of every sub key. This method is less preferable since it may result in a sub-key that is not chosen by any of the traces. For this research it was chosen to first apply majority voting on the sub key. If all the sub key would be unique, the majority voting would be applied on bit level. With this system in hand, it was possible to retrieve the full 1024 bit key with 5 traces. To obtain this result a window to calculate $\bar{t}$ of 20 was used and 10 bits at the time. The frequency of the correct intermediate answers can be seen in Figure 4.15. 5 means that PCTA has 5 equal intermediate results, 2 means that there are only two intermediate results that are equal. The other 3 intermediate results are different. With 5 out of 5, PCTA is very certain of it's case. With only 2, the algorithm is not very sure.

4.5. Progressive Correlation Power Analysis

This method was also tested for power resulting progressive power correlation analysis. Although the method works similar, there is one major difference. That is the generation of the H-matrix. Because temperature rises with activity and voltage drops with activity, the relationship is inverted. Also the voltage drop is smaller then the temperature rise. To correct this, the values from the H-matrix are subtracted from $1.5 \cdot hw_{\text{max}}$.

In order to improve result, also here a specific message was used. An example of the H-matrix that was used can be seen in Equation 4.6. Because the power traces are much clearer, it is easier get to a correct answer (as seen in Figure 4.16. In this case it is possible to retrieve the key with only one trace. The result can be seen in Table 4.3. As a result, it shows that the technique developed for thermals (PCTA) also works for power. In this case, it even works better because the data contains less noise.

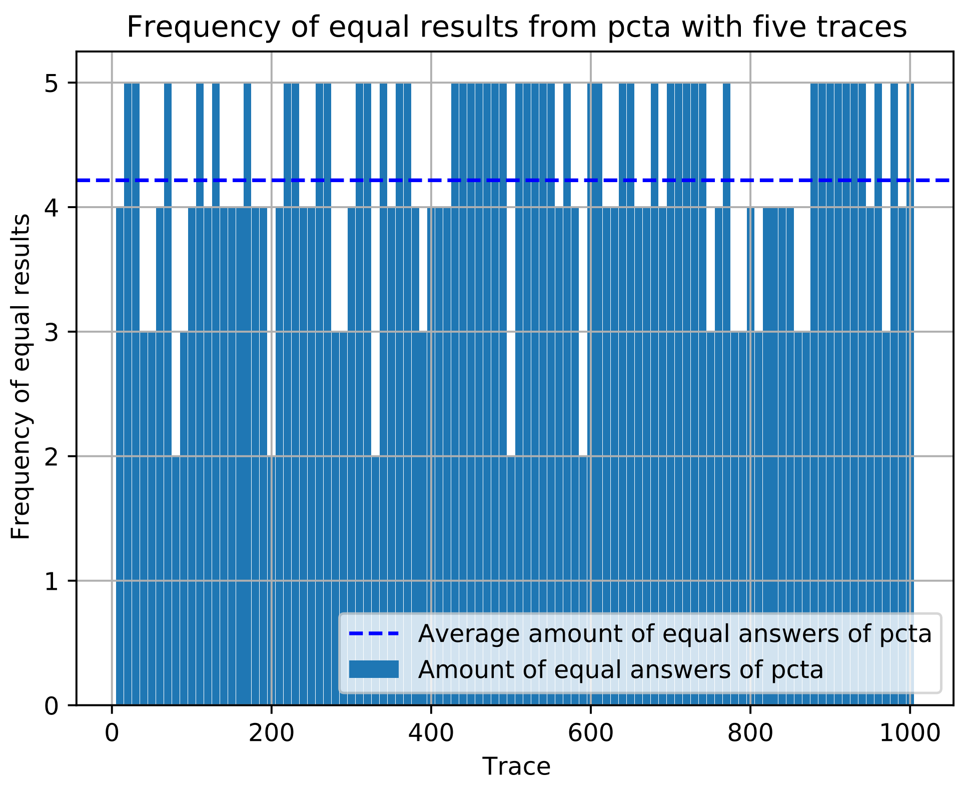


Figure 4.15: Frequency of intermediate results with five traces. Each trace produces an intermediate answer. The most frequent intermediate answer is used for the calculation of the next bits. Higher frequency means that PCTA is more certain of it's intermediate answer.

$$\bar{h} = 35.0$$

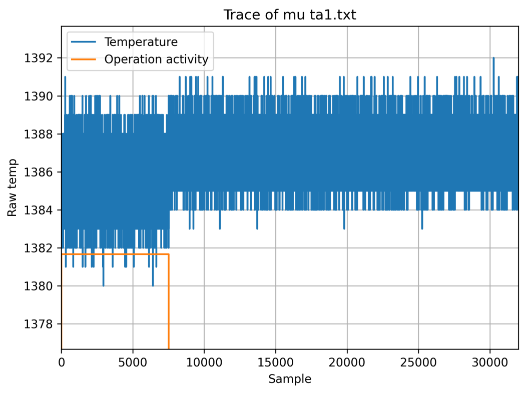
$$H = \begin{bmatrix} .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \\ 24 & 46 & 46 & 46 & 46 & 46 & 46 & 24 & 24 & 46 \\ 24 & 46 & 46 & 46 & 46 & 46 & 46 & 46 & 24 & 46 \\ 24 & 46 & 46 & 46 & 46 & 46 & 46 & 46 & 24 & 24 \\ .. & .. & .. & .. & .. & .. & .. & .. & .. & .. \end{bmatrix} \tag{4.6}$$

	Sequence of correct bits
Key guess list of voltage trace 5	1024
Key guess list of voltage trace 6	1024

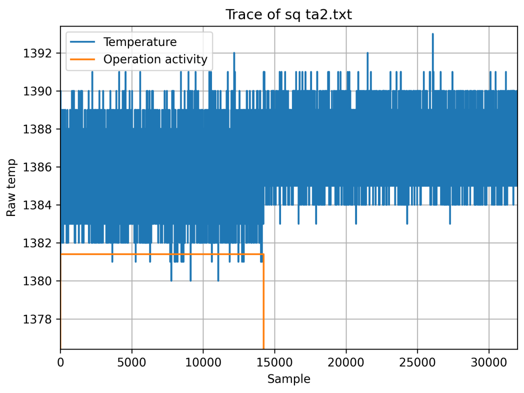
Table 4.3: Results of PCPA on a Montgommery implementation of RSA with $c = N - 1$ on a 1024 bit key

4.6. Convolutional Neural Network Thermal Analysis

Originally the CNN that was used, was designed for power side channel attack. Although concept of classifying a trace remains the same, some adaptation for thermal traces had to be made. The first step in working with a CNN is pre-processing the data. The data that was used in this network was a 1 dimensional temperature trace. Every trace was accompanied with a matching label. In the case of a BINEX implementation, there were two labels. One for the square operation and a zero for the multiply operation.



(a) Voltage trace of one operation in Montgommery implementation of RSA



(b) Voltage trace of one operation in Montgommery implementation of RSA

Figure 4.16: Voltage traces of operation in Montgommery implementation. It shows that the differences between the two operations are very different. As a result of this, PCPA has no trouble at all retrieving the private key

In order to train the network, these traces had to be separated in two groups. The first group consisting of 90 % of the total traces for training and 10 % for validation. To make sure that the validation group was uniform, all the traces were first shuffled and afterwards separated. This made sure that the CNN was able to handle all kinds of offset.

After the shuffling a percentage of ones and zeros was calculated for each data set. In the case of a BINEX implementation this number was usually around 66 %. This score was used to check the performance of the network. For example, if the CNN decided that all the traces resulted in a one, the validation score would still be around 66 % while the CNN was not able to learn anything. In order to make an attack feasible, the accuracy had to be at least 80%. However, with 80 % it would still require a lot of traces to reconstruct the original key. Therefore the training of the network was considered successful with an accuracy of at least 90%.

The next step in the pre-processing was the application of a Fourier transform. By switching from the time domain to the frequency domain, it is much faster to apply filters. If a filter is applied in the time domain, a convolution is required. This is a very intensive operation. However, in the frequency domain, only a point wise multiplication is required [68]. And since the Fast Fourier Transform is highly optimized, it is much faster to do the filtering in the frequency domain. The difference between filtering in the time domain (by convolution) and in the frequency domain (by point wise multiplication) can be seen in Example 4.6.1

Example 4.6.1: Calculating a high pass filter in two ways

Input:

$$\text{trace} = f[0...31999]$$
$$\text{brick wall filter} = g[-15999...16000]$$

For every one of the 32000 elements, the following calculation has to be performed:

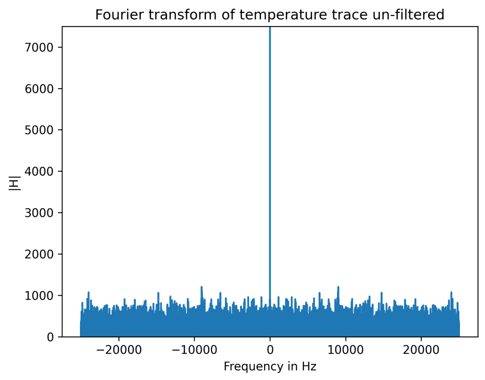
$$input_{cnn}[n] = (f * g)[n] = \sum_{m=-15999}^{16000} f[n-m]g[m]$$

With a Fourier transform, this just requires 32000 multiplications

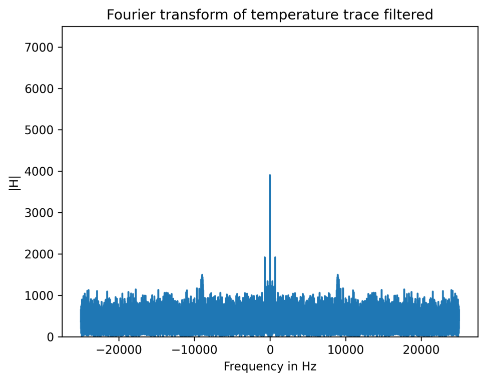
$$F = fft(f)$$
$$G = fft(g)$$
$$input_{cnn}[n] = F[n] \cdot G[n]$$

To get the same result, a ifft should be performed. Although this is just as fast as an fft, it could lead to some rounding errors. And since the CNN is able to work with the fourier transformed trace as well, this step is not performed

The filter that is used is a brick wall high pass filter. This is done to remove the drifting temperature over time. Because the temperature drifts very slow, it is only necessary to remove the very low frequencies. The high frequencies on the other hand are still necessary to analyze the data. Therefore a brick wall high pass filter is applied. The effect of the filter can be seen in Figure 4.17. The setup can be seen in Figure 4.18. The CNN is able to work with data from a Fourier Transform if the values are real. Therefore, the absolute value is taken before it enters the CNN. All the filtered traces can be seen in Appendix B in Figure B.1.



(a) Fourier transform of temperature trace of square operation in RSA



(b) Fourier transform with a high pass filter of temperature trace of square operation in RSA

Figure 4.17: The effect of a high pass filter on temperature traces of a RSA calculation. It shows that the lower frequencies are removed.

Although a CNN can be very powerful in recognising pictures, the training of the network requires and immense amount of computing power. Luckily the training of these CNN can be highly paralleled.

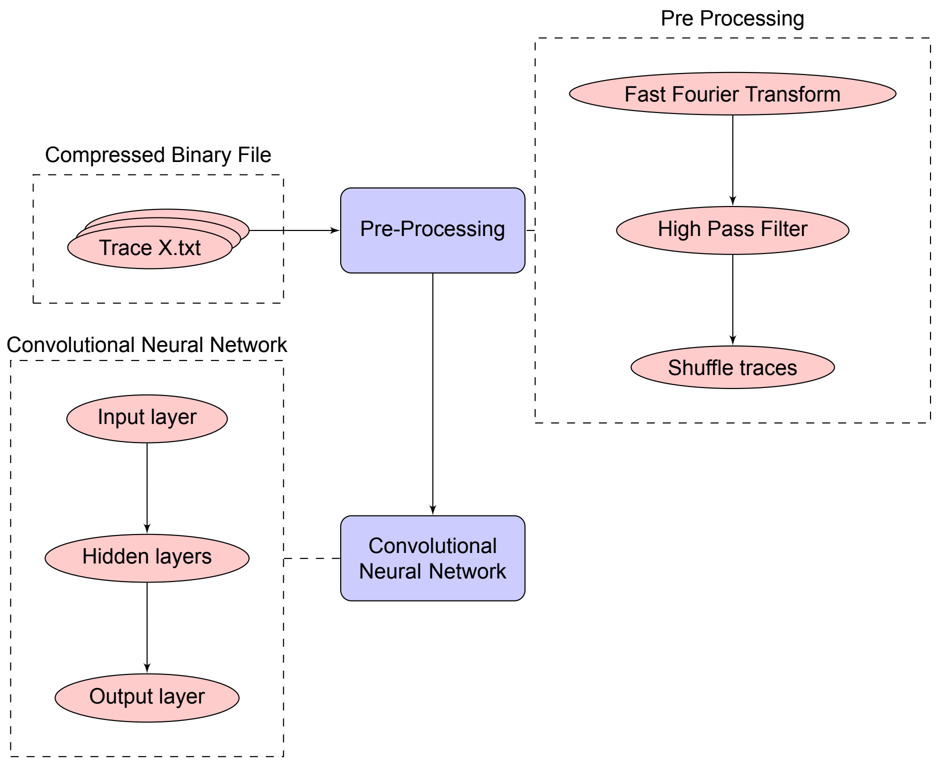


Figure 4.18: Convolutional Neural Network training setup. The traces first get convert to a large binary file. Next they are filtered and shuffled before they're used for training.

The creators of Tensorflow[17] are aware of this and therefore created a version of Tensorflow that is able to run on a GPU. However, this is not an easy task to get going. The version of Tensorflow, CUDA, Nvidia drivers and python have to all match. If this is not the case, it will simply not run on a GPU. To solve this, a docker container was created. With the aid of Nvidia Docker[12][5] it was possible to use GPU acceleration in a virtualized environment. The advantage of this was that it was not necessary to keep the same versions of all the required software exactly the same and that is was more or less portable between systems with a modern Nvidia GPU. With this setup, the training time per epoch was reduced from 85 seconds to 5 seconds on a Nvidia GTX1060. This can be seen in Figure 4.19

The neural network that was used consisted of three convolution layers. After each convolution layer, MaxPooling was applied to down sample the data. This made processing faster. After the MaxPooling, the activation layer (with a ReLu function), the batch normalization and the Gaussian Noise layer were applied. The CNN ends with a flatten layer and a fully contact layer before it enters the SoftMax function. In this last step, the values from the neurons get normalized to a value between 0 and 1. An overview of these layers can be seen in Figure 4.20.

On the first attempts, the CNN has great trouble learning the traces. It starts to over fit very fast. To reduce this problem, some of the solutions from 3.6 were tried. The first solution that was tested was Batch Normalization [58]. Like the description in Section 3.6 it did provide better generalization. However, it was not enough to get the accuracy on the unknown traces up to desired level of 90 %.

To improve this, noise was added at every convolution layer. The idea of this, would be that the noise would create a more variable data input and rely less on a specific neuron. This improved the result up to 90%. To further improve results, the L1 value was altered from 0 to 0.075. This turned out to be the best solution. The accuracy on the validation set peaked at 96.02 %.

After optimizing the hyper-parameters, the training curve looks like Figure 4.21. It shows that only after a few epochs it is able to properly recognise the square and multiply operations. The data set

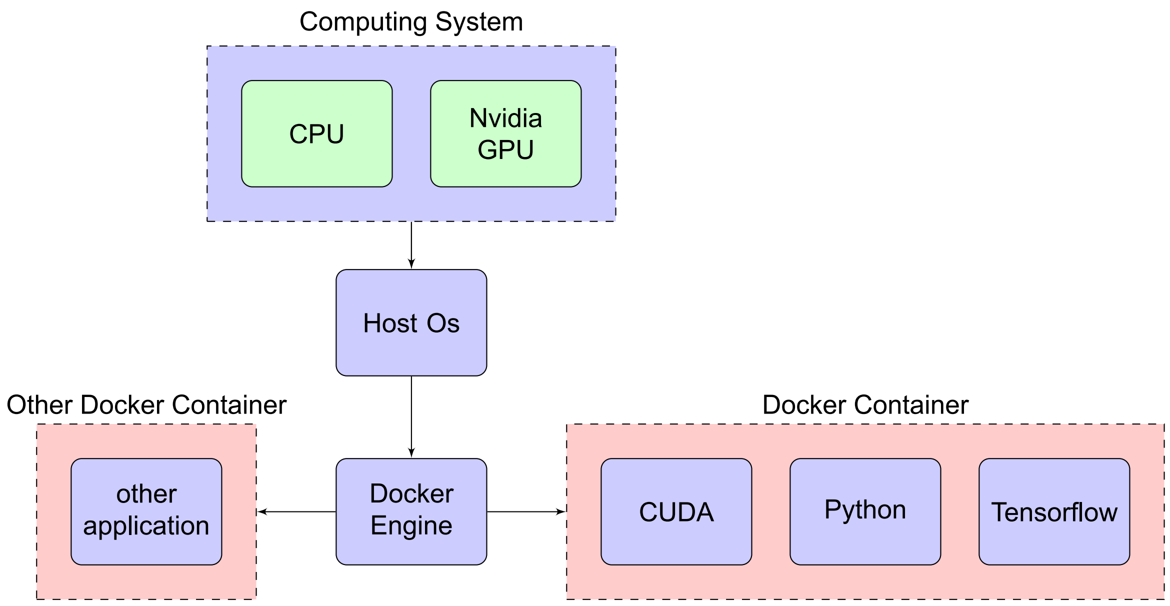


Figure 4.19: Setup with Nvidia-Docker and the CNN. Because of the containerization, it is possible to migrate easily between platforms

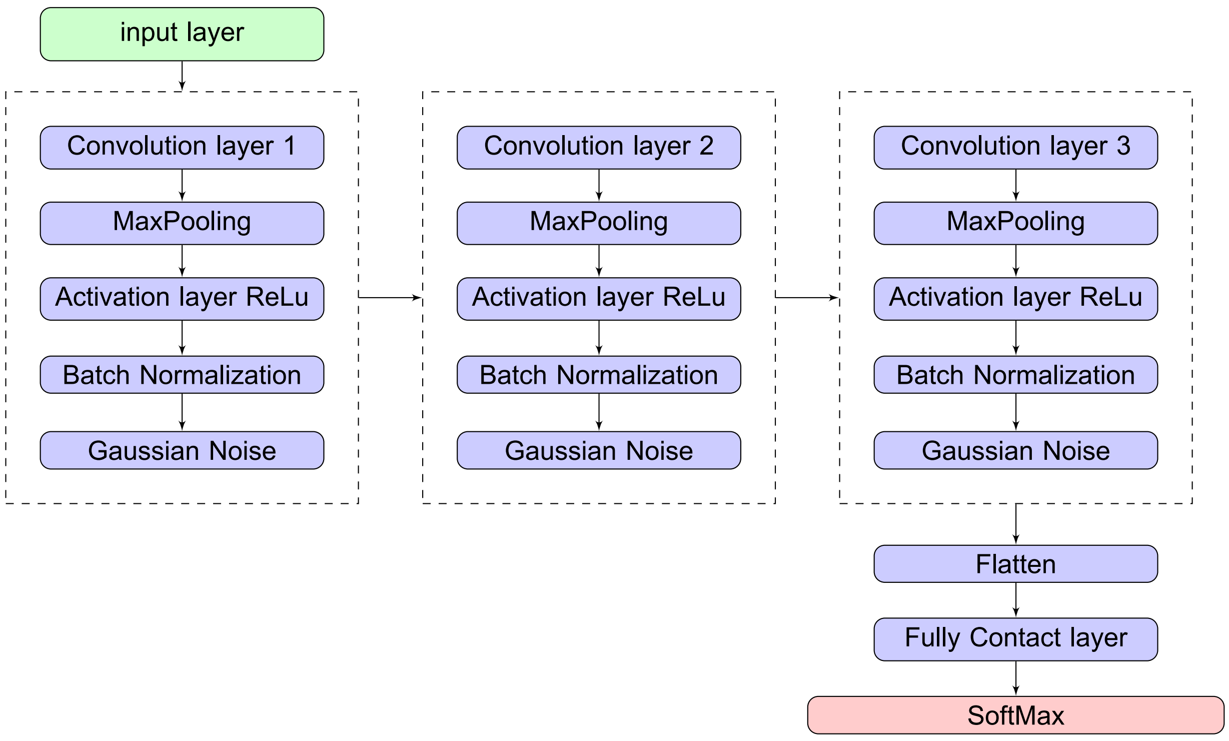


Figure 4.20: This layer description shows every layer in the CNN. After the input layer, there are three main convolution block that finally lead to the output layer.

that is used to train this network consisted of trace temp6, temp7 and temp9. These three sets should create a uniform learning environment.

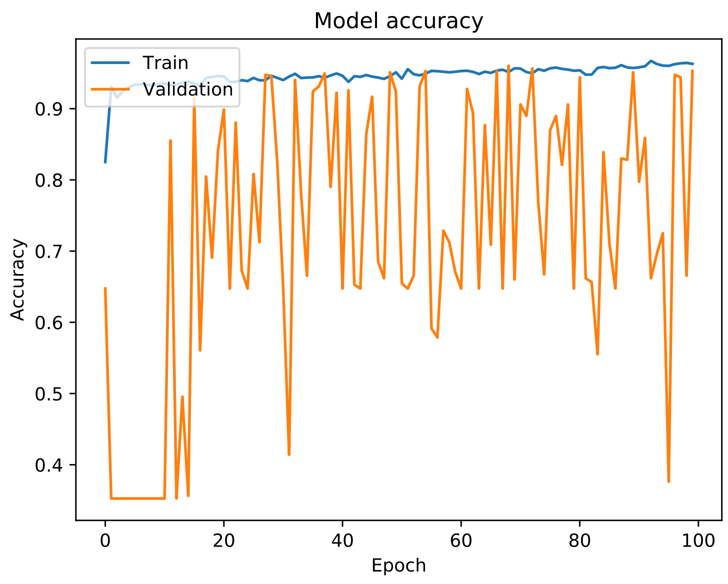


Figure 4.21: Training and validation curve of the CNN. It shows that network is able to train and validate to an accuracy of around 90% with 20 traces.

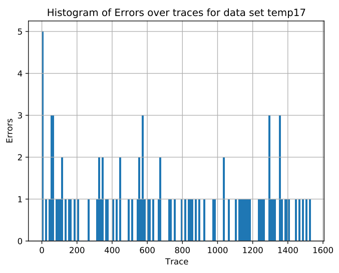
Temp6 is a data set where the key only consists of ones. This results in an even amount of squares and multiplies. Temp9 is a regular RSA key with 1024 bits. Temp12 on the other hand has 2096 bits. This is done because the temperature drift of a longer operation is higher.

With this data set, the pre-processing and the model for the neural network, the CNN is able to get an accuracy of 96.02 %. Although this looks for nice, the network hasn't been tested on real unknown data yet. To test this, the network is tested on a unknown set of traces with an unknown key. On these traces, the CNN is able to get between 91 % and 95 % correct. The results can be seen in Table 4.4.

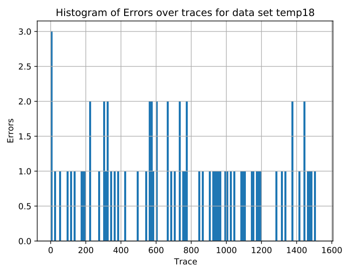
Just like in the case of the CTA, most errors happen at the beginning of the measurement. This can be seen at the error histograms in Figure 4.22. This is due to the same reason as in CTA. In the first few rounds the numbers are much smaller compared to the rest of the calculation. The combined error histogram of all the traces can be seen in Figure 4.23. It shows a uniform distribution of the errors except for the first bits. With majority voting on 9 traces, the network is able to completely retrieve the key. In other words, the CNN is successfully able to perform a thermal side channel attack. If this setup was used for measuring power traces, the results would even be better since power traces are much clearer. It is expected to have results close to 100 % correct with only one trace.

	Correct
Training	96.0 %
Key guess list of trace 14	93.7 %
Key guess list of trace 15	93.9 %
Key guess list of trace 16	93.0 %
Key guess list of trace 17	93.6 %
Key guess list of trace 18	95.1 %
Key guess list of trace 19	91.0 %
Key guess list of trace 20	94.1 %
Key guess list of trace 21	94.2 %
Key guess list of trace 22	92.7 %
Majority voting on trace 14:21	100 %

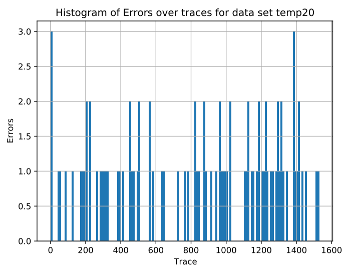
Table 4.4: Results of CNN on a BINEX implementation of RSA, on average they score around 93.48 % correct. With 9 traces it is possible to fully retrieve the original key.



(a) Error histogram of the guess list of temp17



(b) Error histogram of the guess list of temp18



(c) Error histogram of the guess list of temp20

Figure 4.22: The error histogram of the guess list of various traces with CNN

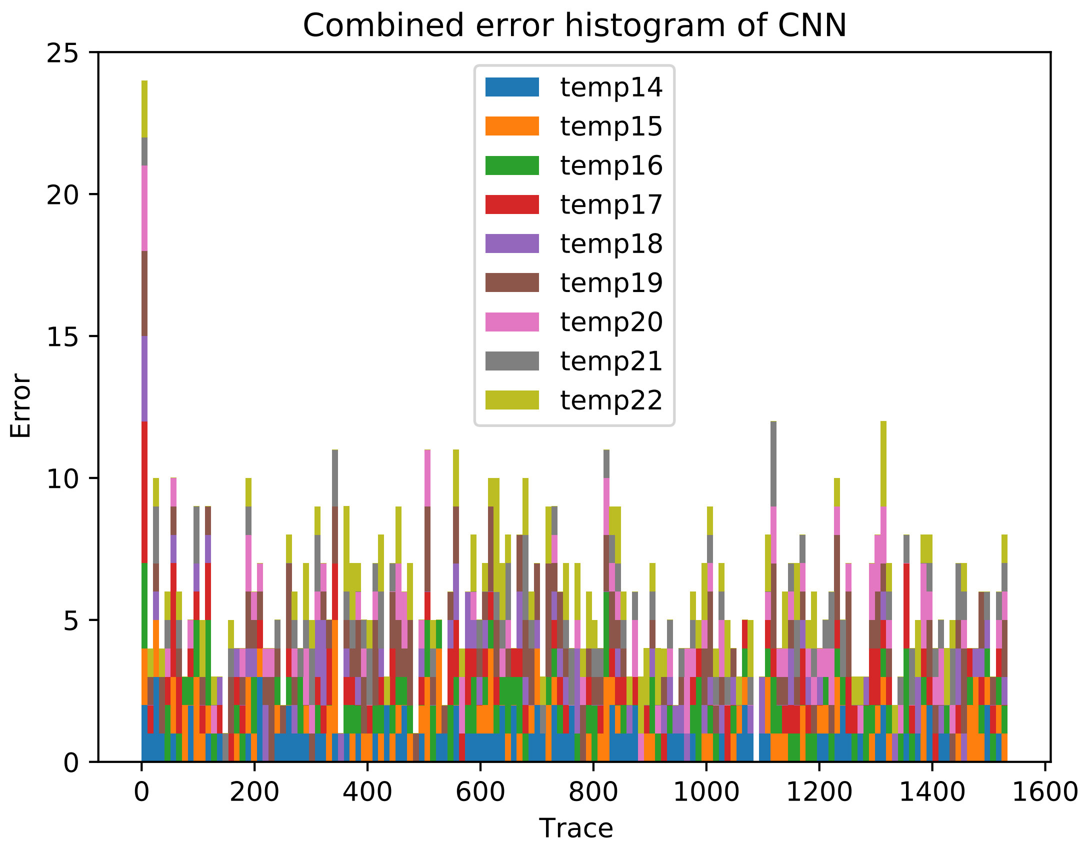


Figure 4.23: Error histogram of CNN of trace 14 until trace 22. Just like in the case of CTA, it shows a pretty uniform error distribution except for the first bits

4.7. Discussion

This research clearly shows that a thermal side channel analysis is possible. It does however require that the operations are relative slow and that there is a fast and accurate temperature sensor. However, the techniques that are described here can also be applied in different domains. Especially the power domain is very accessible with only some minor changes.

In Table 4.5 CTA is compared with the accuracy of the CNN. It turns out that for these temperature traces, both methods are roughly equal accurate (only 0.19 % difference). This can be seen in Figure 4.24. Also the location of the errors seems to appear in similar fashion. Both CTA and CNN have a uniform appearance of errors. However, it must be mentioned that the CNN is a profiled attack and that CTA does not require profiling. Also, CTA is requires far less computing power. This could make CTA more favorable then the CNN. However, CTA does require a bit of tuning depending on the data. The removal of the offset and the size that is being used are very important parameters that have to be tuned for every data set. With profiling this tweaking is much easier, but it still requires some manual labour, whereas the CNN does most of this by itself. The price for this automation on the other hand is a lot of data and a lot of computing power. All the histograms showing the error distribution from Table 4.5 can be seen in the Appendix in Figure B.3 and B.2.

Table 4.5: Comparison of the accuracy of CTA versus CNN

	CTA	**CNN**
Key guess list of trace 14	93.1 %	93.7 %
Key guess list of trace 15	92.8 %	93.9 %
Key guess list of trace 16	92.7 %	93.0 %
Key guess list of trace 17	93.0 %	93.6 %
Key guess list of trace 18	94.4 %	95.1 %
Key guess list of trace 19	92.8 %	91.0 %
Key guess list of trace 20	94.3 %	94.1 %
Key guess list of trace 21	93.6 %	94.2 %
Key guess list of trace 22	92.9 %	92.7 %
Average accuracy	93.29 %	93.48 %

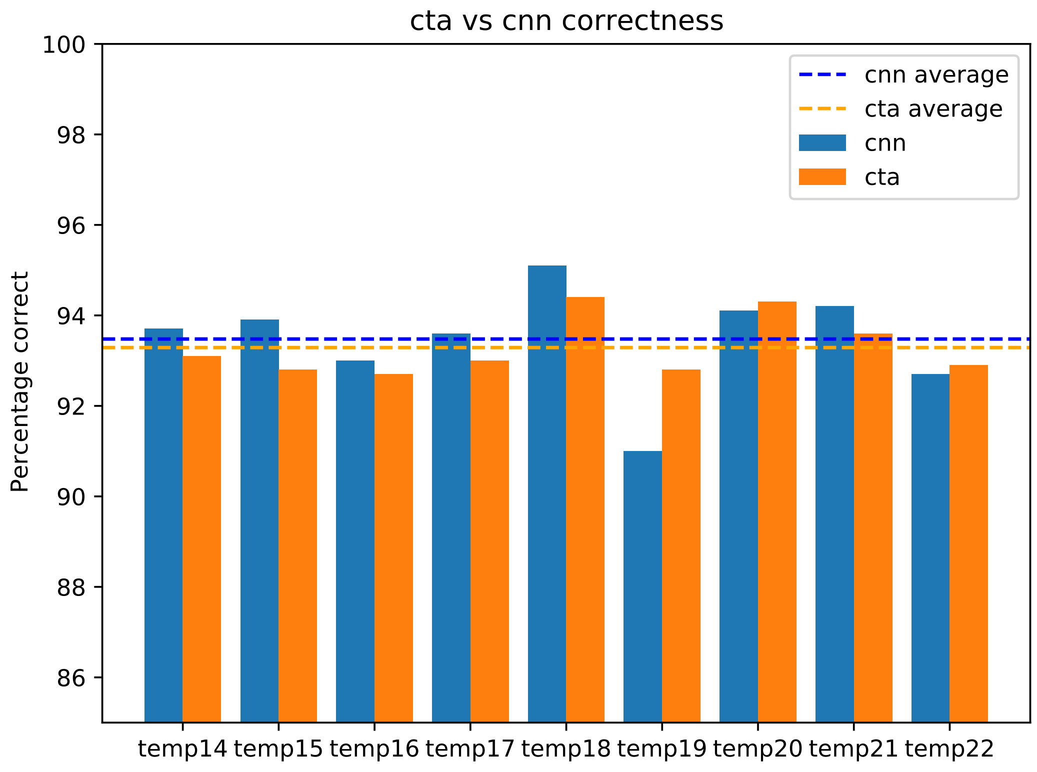


Figure 4.24: Correctness of CTA and CNN compared. They both score around 93% correct and the correctness is very similiar.

PCTA is the only attack method that is suitable for a Montgommery implementation. If there is no specific message used, the attack requires a lot of traces and might not even be possible in the thermal domain. However, with the aid of a specific message, $c = N - 1$ or $m = N - 1$, it is possible to reconstruct the complete key with 5 traces. Although this attack is un-profiled, it is more powerful when the parameter such as the average window, the size of the average to remove the offset and the size of the trace that is being used are optimized.

The big advantage of PCTA versus CTA is that it is able to use information about previous bits. This is required for a Montgommery implementation but not for a BINEX implementation. Because of the regularity of the BINEX implementation, PCTA is able to handle incorrect bits. This makes PCTA just as suitable as CTA for attacking a BINEX implementation. However, because it does not add any value and does require a bit more computing power, CTA is preferred in this case. This also applies to the power domain. If the implementation is BINEX, CPA is better simply because it's a simpler attack. In the case of the a Montgommery Implementation, PCPA is the only way.

With this results it is clear that a thermal side channel attack is possible. It is however much more complex then a power side channel attack. There are two main reasons for this. First, temperature changes much slower and second, temperature is unregulated. This results in a drifting offset. The combat this first problem, the FPGA was paused on a regular interval to measure the temperature and to cool down for the next operation. Although this makes the attack possible, it is an intrusive operation for an attacker. Pausing the clock without breaking a system can be a big challenge.

The next problem with temperature is that most system do not have a dedicated fast temperature sensor. Although the PYNQ-Z1 was equipped with one, in most devices this is not the case. A work around for this problem, would be adding a temperature sensor to a chip by the attacker it self. The disadvantage of this, is that the temperature sensor must be able to react quickly enough and therefore has to be very small.

Nevertheless, there is information leakage in the thermal behaviour of a chip. This is something to keep in mind when designing a secure system. Although the countermeasures can be relative easy (for example making operation very fast or very energy efficient), it is something that is that should be taken into account.

5 Conclusion

This chapter is the last chapter of this thesis. During these final words on this research the summary and the future work are given.

5.1. Summary

The goal of this research was to confirm the existence of a thermal side channel attack. It has been mentioned in the past, but it has never been tested in an elaborate way. For this research four different methods were developed. Three of them (STA, CTA and CNN were strongly based on power side channel analysis but had to be converted to the thermal domain. One method however, was specifically engineered for the thermal domain (PCTA). This method turned out be so effective, it was also converted to the power domain (PCPA).

In **Chapter 1** the introduction for this research is described. This chapter also gives the reader an overview of the state of art, the contribution of this research and states the thesis organization.

During **Chapter 2** the necessary background information is given. This background is divided in three parts. The need for secure systems, cryptography and side channel attacks. In cryptography to following algorithms are explained: SHA1, AES, ChaCha and RSA. The part on side channal attacks discussed the following methods: SPA, DPA, CPA, CNN, TiA, fault injection and countermeasures.

Chapter 3 describes the methodology of this research. It first tells the reader about the thread models and then continues to the different leakage models. After this the four thermal side channel attack methods are introduced to the reader. It also gives a deeper explanation on the various side channel attacks.

Chapter 4 is the most important part of this thesis. It tells the reader about the validation and the results of this research. All four methods, STA,CTA,PCTA and CNN have been successfully tested and verified in their own scenario. This chapter tests the theory from Chapter 3 and described the challenges along the way. During this chapter, the existence of a thermal side channel attack is proven with four different methods. It describes that it is possible to retrieve the private key of a RSA implementation (unproteced and protected) with the aid of thermal traces. Although power side channel analysis still perform better, it is a feasible alternative in case it is very to measure power. For example when a complex power regulation system is used to feed the microprocessor.

The last chapter, **Chapter 5** describes the Summary and the Future Work.

5.2. Future work

This section describes the topics that can be further improved for research. With the research of this thesis, some interesting thoughts came along. With these concepts came the following potential fields of research:

- **Attack methods for thermal side channel attacks on other encryption algorithms:** this research has focused on attack RSA. However, there are many more encryption algorithm. AES, ECC are both very popular, and might also be vulnerable to thermal side channel attacks.
- **Attack methods for other protected implementations of RSA:** in this thesis, only the Montgommery Ladder implementation of RSA is considered. However, there are other protected implementations for example with key and message bliding [59]. On these algorithm it is also possible to further investigate the possibilities of thermal side channel attacks.
- **Explore different pre-processing methods:** this research validates two pre-processing methods for thermal side channel attacks, auto-zero and using a high pass filter. There are however more ways to reduce the amount of noise and remove the drift.
- **Countermeasures for thermal side channel attack:** just like in the case for power side channel analysis, thermal side channel analysis also needs countermeasures. Because thermals behave very different from power, it also can be an interesting research topic.
- **Differential Thermal Analysis:** although this method is more complex on thermals due to the drifting behaviour, it could be possible. With the aid of differential thermal analysis or higher order differential thermal analysis, the accuracy could further be increased. It does however require proper pre-processing.
- **Hybrid attack with Progressive Correlation Analysis and Neural Networks:** PCPA and PCTA both have proven to be very useful in attacking a Montgommery Ladder. However, in some cases a CNN can be very accurate in classifying traces. If a CNN would be used to estimate a value for a trace, PCPA could be used to link this value to a key bit. For example, the CNN would estimate the power consumption on a trace to a 7. With the estimated HW and information of the previous bits, PCPA (but also PCTA) is able to predict the key bit.

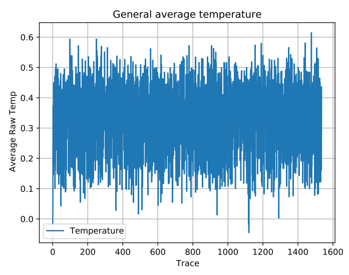
A
Trace List

Name	Domain	Algorithm	Key size	Key information	Cipher inforomation
temp6	Temperature	RSA BINEX	1024	all ones	random
temp7	Temperature	RSA BINEX	1024	generated OpenSSL	random
temp9	Temperature	RSA BINEX	2048	generated OpenSSL	random
temp12	Temperature	RSA BINEX	1024	generated OpenSSL	random
temp13	Temperature	RSA BINEX	1024	generated OpenSSL	random
temp14	Temperature	RSA BINEX	1024	unknown test key	random
temp15	Temperature	RSA BINEX	1024	unknown test key	random
temp16	Temperature	RSA BINEX	1024	unknown test key	random
temp17	Temperature	RSA BINEX	1024	unknown test key	random
temp18	Temperature	RSA BINEX	1024	unknown test key	random
temp19	Temperature	RSA BINEX	1024	unknown test key	random
temp20	Temperature	RSA BINEX	1024	unknown test key	random
temp21	Temperature	RSA BINEX	1024	unknown test key	random
temp22	Temperature	RSA BINEX	1024	unknown test key	random
temp23	Temperature	RSA Montgommery	1024	generated OpenSSL	random
temp24	Temperature	RSA Montgommery	1024	generated OpenSSL	random
temp25	Temperature	RSA Montgommery	1024	ones and zeros	random
temp26	Temperature	RSA Montgommery	1024	generated OpenSSL	random
temp27	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
temp28	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
temp29	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
temp30	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
temp31	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
temp32	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
temp33	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
temp34	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
temp35	Temperature	RSA Montgommery	1024	unknown test key	$c = N - 1$
vcc1	Voltage	RSA BINEX	1024	generated OpenSSL	random
vcc2	Voltage	RSA Montgommery	1024	generated OpenSSL	random
vcc3	Voltage	RSA Montgommery	1024	generated OpenSSL	random
vcc4	Voltage	RSA Montgommery	1024	generated OpenSSL	random
vcc5	Voltage	RSA Montgommery	1024	unknown test key	$c = N - 1$
vcc6	Voltage	RSA Montgommery	1024	unknown test key	$c = N - 1$
vcc7	Voltage	RSA Montgommery	1024	unknown test key	$c = N - 1$

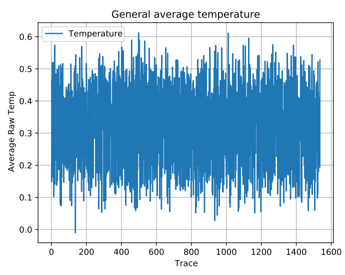
Table A.1: Information about all the traces for this research. Unknown key means that the key is the same for those traces that have this mentioned. These traces are used for majority voting. Generated keys are random keys and are different for each trace.

B

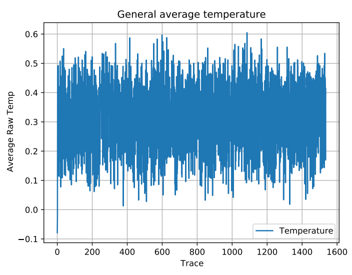
Figures



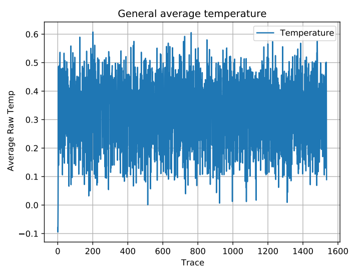
(a) Temperature trace set 14 with offset removed



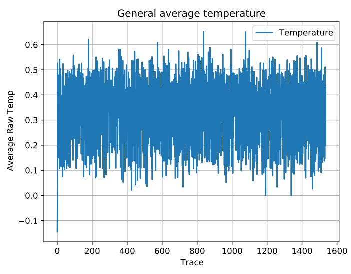
(b) Temperature trace set 15 with offset removed



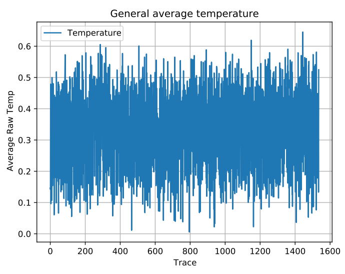
(c) Temperature trace set 16 with offset removed



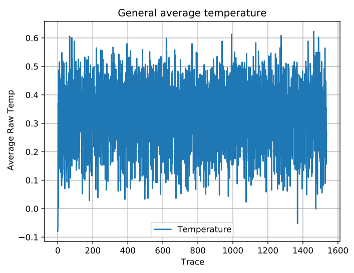
(d) Temperature trace set 17 with offset removed



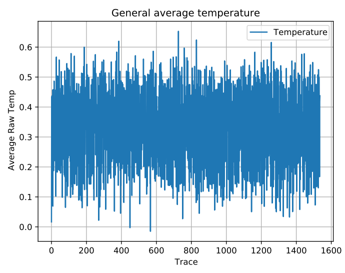
(e) Temperature trace set 18 with offset removed



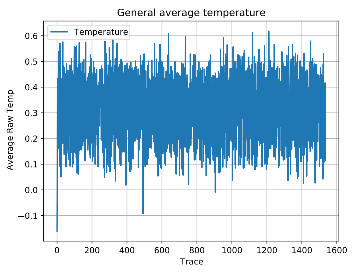
(f) Temperature trace set 19 with offset removed



(g) Temperature trace set 20 with offset removed

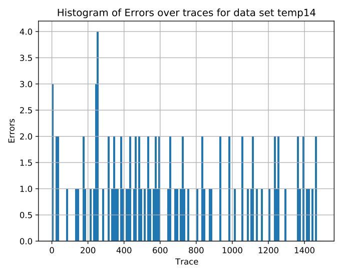


(h) Temperature trace set 21 with offset removed

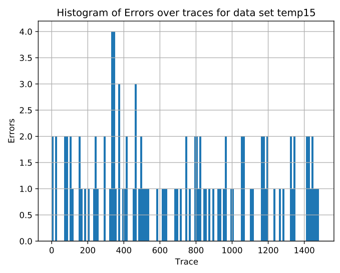


(i) Temperature trace set 22 with offset removed

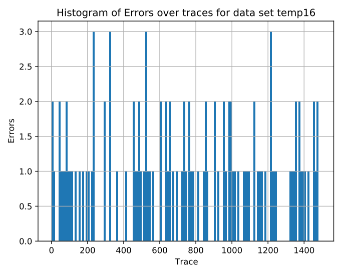
Figure B.1: The offset is removed from these traces by subtracting the local average from the total. This removes the temperature drift.



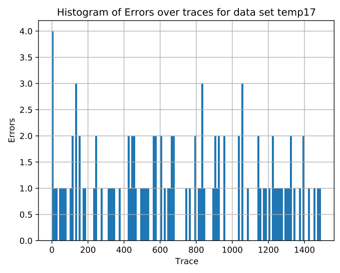
(a) Error histogram CPA on trace set 14



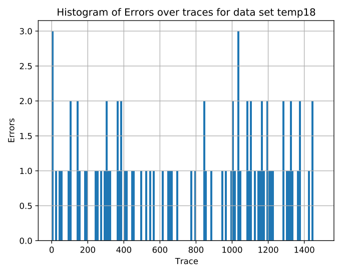
(b) Error histogram CPA on trace set 15



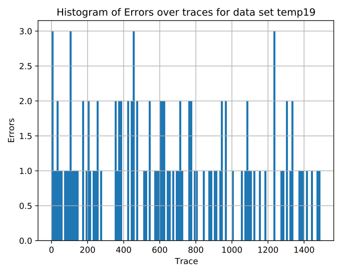
(c) Error histogram CPA on trace set 16



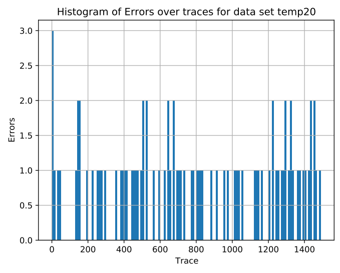
(d) Error histogram CPA on trace set 17



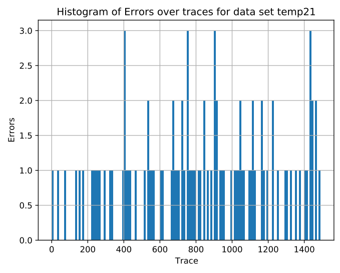
(e) Error histogram CPA on trace set 18



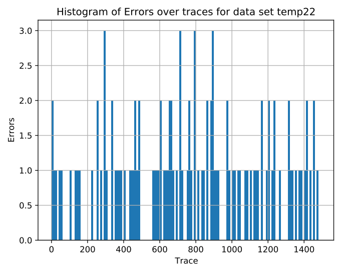
(f) Error histogram CPA on trace set 19



(g) Error histogram CPA on trace set 20

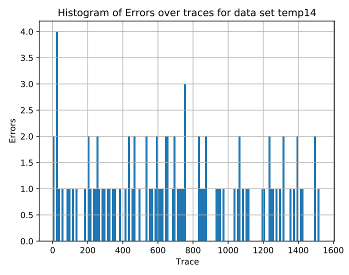


(h) Error histogram CPA on trace set 21

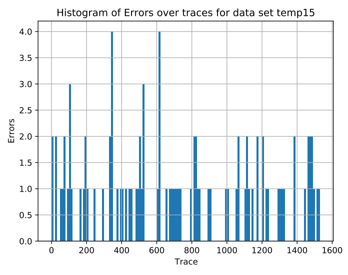


(i) Error histogram CPA on trace set 22

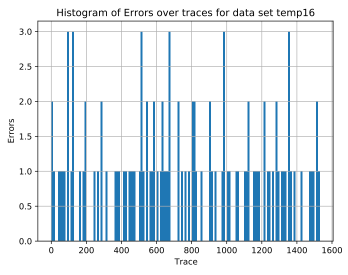
Figure B.2: All the error histograms of CPA. It shows that the beginning usually contains an error but the rest of the errors is pretty uniform.



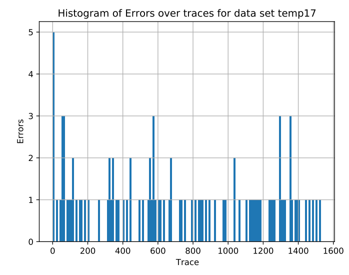
(a) Error histogram CNN on trace set 14



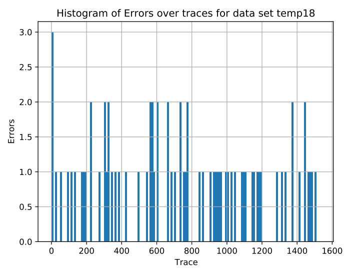
(b) Error histogram CNN on trace set 15



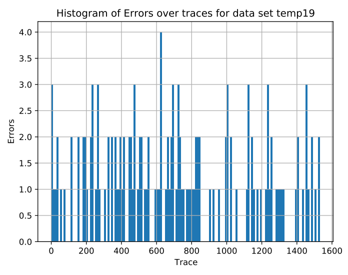
(c) Error histogram CNN on trace set 16



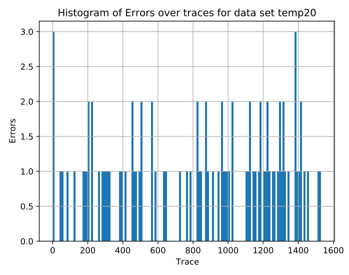
(d) Error histogram CNN on trace set 17



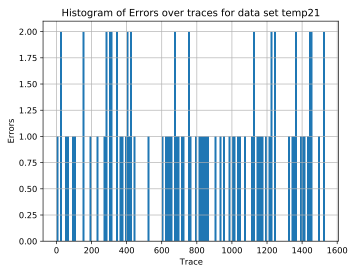
(e) Error histogram CNN on trace set 18



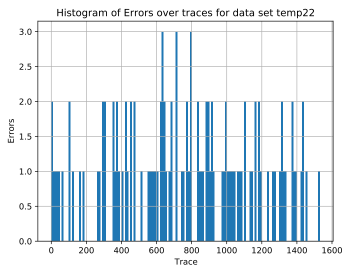
(f) Error histogram CNN on trace set 19



(g) Error histogram CNN on trace set 20



(h) Error histogram CNN on trace set 21



(i) Error histogram CNN on trace set 22

Figure B.3: All the error histograms of CNN. It shows that the beginning usually contains an error but the rest of the errors is pretty uniform.

Bibliography

[1] Aes galois field. URL `https://www.samiam.org/galois.html`.

[2] Ascii. URL `https://en.wikipedia.org/wiki/ASCII`.

[3] Ceasar cipher image. URL `https://tex.stackexchange.com/questions/371906/how-to-draw-a-caesar-cipher-diagram-with-tikz`.

[4] Cia triangle. URL `https://www.techniumnetworking.com/platform/`.

[5] Docker. URL `https://docker.com`.

[6] *Flatten*. URL `https://keras.io/api/layers/reshaping_layers/flatten/`.

[7] Hamming distance. URL `https://en.wikipedia.org/wiki/Hamming_distance`.

[8] Https. URL `https://en.wikipedia.org/wiki/HTTPS`.

[9] Hamming weight. URL `https://en.wikipedia.org/wiki/Hamming_weight`.

[10] *Using the MicroBlaze Processor to Accelerate Cost-Sensitive Embedded System Development*.

[11] *Noise layer*. URL `https://keras.io/api/layers/regularization_layers/gaussian_noise/`.

[12] Nvidia-docker. URL `https://github.com/NVIDIA/nvidia-docker/`.

[13] Openssl. URL `https://www.openssl.org/`.

[14] Digital signature diagram, . URL `https://upload.wikimedia.org/wikipedia/commons/2/2b/Digital_Signature_diagram.svg`.

[15] Rsa crt example, . URL `https://www.di-mgt.com.au/crt_rsa.html`.

[16] Secure shell. URL `https://www.ssh.com/about/`.

[17] Tensorflow. URL `https://www.tensorflow.org/`.

[18] Vivado design suite. URL `https://www.xilinx.com/products/design-tools/vivado.html`.

[19] *7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter*.

[20] How much computing resource is required to brute-force rsa?, Jun 2012. URL `https://crypto.stackexchange.com/questions/3043/how-much-computing-resource-is-required-to-brute-force-rsa`.

[21] S. J. Aboud, M. A. AL-Fayoumi, M. Al-Fayoumi, and H. S. Jabbar. An efficient rsa public key encryption scheme. In *Fifth International Conference on Information Technology: New Generations (itng 2008)*, pages 127–130, 2008.

[22] Dakshi Agrawal, Bruce Archambeault, Josyula R. Rao, and Pankaj Rohatgi. The em side—channel(s). In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36400-9.

[23] Abdullah Aljuffri. Exploring deep learning for hardware attacks. Master's thesis, Delft University of Technology, Mekelweg 4, 2628CD Delft, 4 2018.

[24] Gaylord O Asoronye, Goodluck I Emereonye, Ibiam A Agha, et al. An efficient implementation for the cryptanalysis of caesar's cipher. *The Melting Pot*, 5(2), 2019.

[25] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The sorcerer's apprentice guide to fault attacks. *Proceedings of the IEEE*, 94:370–382, 2004.

[26] Aurélie Bauer, Eliane Jaulmes, Emmanuel Prouff, and Justine Wild. Horizontal and vertical side-channel attacks against secure rsa implementations. In *Cryptographers' Track at the RSA Conference*, pages 1–17. Springer, 2013.

[27] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ascad database. *ANSSI, France & CEA, LETI, MINATEC Campus, France. Online verfügbar unter https://eprint. iacr. org/2018/053. pdf, zuletzt geprüft am*, 22:2018, 2018.

[28] Daniel J Bernstein. Cache-timing attacks on aes. 2005.

[29] Daniel J Bernstein. Chacha, a variant of salsa20. In *Workshop Record of SASC*, volume 8, pages 3–5, 2008.

[30] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[31] J. Brouchier, T. Kean, C. Marsh, and D. Naccache. Temperature attacks. *IEEE Security Privacy*, 7(2):79–82, 2009.

[32] J. Brouchier, T. Kean, C. Marsh, and D. Naccache. Temperature attacks. *IEEE Security Privacy*, 7(2):79–82, March 2009. ISSN 1540-7993. doi: 10.1109/MSP.2009.54.

[33] Christophe Clavier, Benoit Feix, Georges Gagnerot, Mylène Roussellet, and Vincent Verneuil. Improved collision-correlation power analysis on first order protected aes. In Bart Preneel and Tsuyoshi Takagi, editors, *Cryptographic Hardware and Embedded Systems – CHES 2011*, pages 49–62, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg. ISBN 978-3-642-23951-9.

[34] Joan Daemen and Vincent Rijmen. The block cipher rijndael. volume 1820, pages 277–284, 01 1998. doi: 10.1007/10721064_26.

[35] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

[36] Zhaojing Ding, Wei Guo, Liangjian Su, Jizeng Wei, and Haihua Gu. Further research on n-1 attack against exponentiation algorithms. In Willy Susilo and Yi Mu, editors, *Information Security and Privacy*, pages 162–175, Cham, 2014. Springer International Publishing. ISBN 978-3-319-08344-5.

[37] Donald Eastlake and Paul Jones. Us secure hash algorithm 1 (sha1), 2001.

[38] C. C. Enz and G. C. Temes. Circuit techniques for reducing the effects of op-amp imperfections: autozeroing, correlated double sampling, and chopper stabilization. *Proceedings of the IEEE*, 84 (11):1584–1614, 1996.

[39] Pierre-Alain Fouque, Nicolas Guillermin, Delphine Leresteux, Mehdi Tibouchi, and Jean-Christophe Zapalowicz. Attacking rsa–crt signatures with faults on montgomery multiplication. *Journal of Cryptographic Engineering*, 3(1):59–72, 2013.

[40] Daniel Genkin, Adi Shamir, and Eran Tromer. Rsa key extraction via low-bandwidth acoustic cryptanalysis. In Juan A. Garay and Rosario Gennaro, editors, *Advances in Cryptology – CRYPTO 2014*, pages 444–461, Berlin, Heidelberg, 2014. Springer Berlin Heidelberg. ISBN 978-3-662-44371-2.

[41] Henri Gilbert and Helena Handschuh. Security analysis of sha-256 and sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, pages 175–193, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-24654-1.

[42] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[43] Andy Greenberg. Hackers remotely kill a jeep on the highway-with me in it, Jul 2015. URL `https://www.wired.com/2015/07/hackers-remotely-kill-jeep-highway/`.

[44] M. Happe, A. Agne, and C. Plessl. Measuring and predicting temperature distributions on fpgas at run-time. In *2011 International Conference on Reconfigurable Computing and FPGAs*, pages 55–60, 2011.

[45] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1 (4):293, 2011.

[46] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1 (4):293, 2011.

[47] Michael Hutter and Jörn-Marc Schmidt. The temperature side-channel and heating fault attacks. volume 8419, 11 2013. doi: 10.1007/978-3-319-08302-5_15.

[48] Vincent Immler, Johannes Obermaier, Kuan Kuan Ng, Fei Xiang Ke, JinYu Lee, Yak Peng Lim, Wei Koon Oh, Keng Hoong Wee, and Georg Sigl. Secure physical enclosures from covers with tamper-resistance. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 51–96, 2019.

[49] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015. URL `http://arxiv.org/abs/1502.03167`.

[50] C. H. Kim and J. Quisquater. Faults, injection methods, and fault attacks. *IEEE Design Test of Computers*, 24(6):544–545, 2007.

[51] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan. Leakage current: Moore's law meets static power. *Computer*, 36(12):68–75, 2003.

[52] Ágnes Kiss, Juliane Krämer, Pablo Rauzy, and Jean-Pierre Seifert. Algorithmic countermeasures against fault attacks and power analysis for rsa-crt. In François-Xavier Standaert and Elisabeth Oswald, editors, *Constructive Side-Channel Analysis and Secure Design*, pages 111–129, Cham, 2016. Springer International Publishing. ISBN 978-3-319-43283-0.

[53] Neal Koblitz, Alfred Menezes, and Scott Vanstone. The state of elliptic curve cryptography. *Designs, codes and cryptography*, 19(2-3):173–193, 2000.

[54] Paul Kocher, Joshua Jaffe, Benjamin Jun, et al. Introduction to differential power analysis and related attacks, 1998.

[55] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9.

[56] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-68697-2.

[57] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 104–113, Berlin, Heidelberg, 1996. Springer Berlin Heidelberg. ISBN 978-3-540-68697-2.

[58] S. Liu and W. Deng. Very deep convolutional neural network based image classification using small training sample size. In *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, pages 730–734, 2015.

[59] Zhe Liu, Johann Großschädl, and Ilya Kizhvatov. Efficient and side-channel resistant rsa implementation for 8-bit avr microcontrollers. 01 2010.

[60] Owen Lo, William J Buchanan, and Douglas Carson. Power analysis attacks on the aes-128 s-box using differential power analysis (dpa) and correlation power analysis (cpa). *Journal of Cyber Security Technology*, 1(2):88–107, 2017.

[61] Rita Mayer-Sommer. Smartly analyzing the simplicity and the power of simple power analysis on smartcards. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 78–92, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-44499-2.

[62] J. Nagi, F. Ducatelle, G. A. Di Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 342–347, 2011.

[63] Nan Li. Research on diffie-hellman key exchange protocol. In *2010 2nd International Conference on Computer Engineering and Technology*, volume 4, pages V4–634–V4–637, 2010.

[64] Rachel Nuwer. Computer scientists hack michigan traffic lights to show glaring security flaws, Aug 2014. URL `https://www.smithsonianmag.com/smart-news/computer-scientists-hack-michigan-traffic-lights-show-glaring-security-flaws-180`

[65] Dwiti Pandya, Khushboo Ram Narayan, Sneha Thakkar, Tanvi Madhekar, and BS Thakare. Brief history of encryption. *International Journal of Computer Applications*, 975:8887, 2015.

[66] B. Parhami. *Algorithms and Design Methods for Digital Computer Arithmetic*. Oxford University Press, 2012. ISBN 9780199766932. URL `https://books.google.nl/books?id=kEiIvgAACAAJ`.

[67] Behrooz Parhami. Oxford University Press, 2010. ISBN 978-0-19-532848-6. URL `https://app.knovel.com/hotlink/toc/id:kpCAAHDE0D/computer-arithmetic-algorithms/computer-arithmetic-algorithms`.

[68] John G. Proakis and Dimitris K Manolakis. *Digital Signal Processing (4th Edition)*. Prentice Hall, 4 edition, 2006. ISBN 0131873741.

[69] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978. ISSN 0001-0782. doi: 10.1145/359340.359342. URL `https://doi.org/10.1145/359340.359342`.

[70] Ronald Rivest and S Dusse. The md5 message-digest algorithm, 1992.

[71] Eloi Sanfelix, Cristofaro Mune, and Job de Haas. Unboxing the white-box. In *Black Hat EU 2015*. 2015.

[72] J. Schmidt and C. Herbst. A practical fault attack on square and multiply. In *2008 5th Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 53–58, 2008.

[73] Sigrok. Sigrok command line interface, 2019. URL `https://sigrok.org/wiki/Sigrok-cli`.

[74] Gurpreet Singh. A study of encryption algorithms (rsa, des, 3des and aes) for information security. *International Journal of Computer Applications*, 67(19), 2013.

[75] Rashmi Sharan Sinha, Yiqiao Wei, and Seung-Hoon Hwang. A survey on lpwa technology: Lora and nb-iot. *Ict Express*, 3(1):14–21, 2017.

[76] M. E. Smid and D. K. Branstad. Data encryption standard: past and future. *Proceedings of the IEEE*, 76(5):550–559, 1988.

[77] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1): 1929–1958, January 2014. ISSN 1532-4435.

[78] Marc Stevens. Cryptanalysis of md5 & sha-1. *Proceedings of the Special-Purpose Hardware for Attacking Cryptographic Systems (SHARCS'12)*, pages 17–18, 2012.

[79] Mottaqiallah Taouil. Cryptography. University Lecture, 2020.

[80] Mottaqiallah Taouil. Introduction to cybersecurity. University Lecture, 2020.

[81] M Tehranipoor. Physical attacks and tamper resistance. *ECE6095: Hardware Security & Trust University of Connecticut ECE Department*.

[82] Charles R Trimble. What is signal averaging. *Hewlett-Packard Journal*, 19(8):2–7, 1968.

[83] Marc Witteman. A dpa attack on rsa in crt mode, 2009.

[84] Jun Wu, Yong-Bin Kim, and Minsu Choi. Low-power side-channel attack-resistant asynchronous s-box design for aes cryptosystems. In *Proceedings of the 20th symposium on Great lakes symposium on VLSI*, pages 459–464, 2010.

[85] Xilinx. Pynq-z1: Python productivity for zynq-7000 arm/fpga soc. URL `https://www.xilinx.com/products/boards-and-kits/1-hydd4z.html`.