UNIVERSIDADE D
COIMBRA

Eduardo Filipe Ferreira da Cruz

SECURE FIRMWARE ENCRYPTION AGAINST POWER ANALYSIS ATTACKS

Dissertation in the context of the Master's in Cyber Security, advised by Professor Dr. António Jorge da Costa Granjal and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023

1 2 9 0

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Eduardo Filipe Ferreira da Cruz

Encriptação de Firmware segura contra ataques baseados em Análise de Consumo Elétrico

Dissertação no âmbito do Mestrado em Segurança Informática, orientada pelo Professor Doutor António Jorge da Costa Granjal e apresentada ao Departamento de Engenharia Informática da Faculdade de Ciências e Tecnologia da Universidade de Coimbra.

Julho de 2023

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE D
COIMBRA

DEPARTAMENTO DE ENGENHARIA INFORMÁTICA

Eduardo Filipe Ferreira da Cruz

Secure Firmware Encryption against Power Analysis attacks

Dissertation in the context of the Master in Cyber Security, advised by Professor Dr. António Jorge da Costa Granjal and presented to the Department of Informatics Engineering of the Faculty of Sciences and Technology of the University of Coimbra.

July 2023

Acknowledgment

I would like to express my sincere gratitude and appreciation to my dissertation supervisor at *University of Coimbra*, Prof. Dr. António Jorge Granjal. I am very fortunate to have benefitted from your exceptional support, guidance, and expertise throughout this process.

I would like to extend my heartfelt appreciation to Nabil Hamzi, my internship supervisor at *Logitech*, for entrusting me with the opportunity of working on such a meaningful and interesting project. Your guidance, valuable advice, and willingness to share your vast knowledge with me were essential in achieving this outcome and also in propelling my own development.

I wish to thank everyone at *Logitech* who played a part in assisting me throughout my internship – you were truly amazing.

To the members of the jury, Prof. Dr. Ricardo Chaves and Prof. Dr. Fernando Boavida, I want to convey my gratitude for your valuable involvement.

To my beloved parents, whose support and endless encouragement have always impacted my journey. Thank you for embracing my aspirations and granting me the privilege to pursue my studies.

To my dearest brother, I hope you're proud of this one. Thank you for shaping me and supporting me in every chapter of life.

To my cheerful girlfriend, Micaela, your immense love, care, and support have brought joy and strength through the tough times. I'm truly grateful to have had you by my side.

Thank you.

Resumo

Firmware consiste no software que fornece aos dispositivos eletrónicos o controlo sobre o seu próprio hardware. Dada esta característica, é inevitável refletir sobre as ameaças e os riscos de segurança envolvidos, tanto para os usuários como para as empresas que produzem os dispositivos. É, portanto, relevante propor soluções concretas que permitam minimizar as consequências e a probabilidade de ocorrência de falhas que possam comprometer segurança relacionada com o *firmware*. Um dos vários possíveis mecanismos aplicados pelas empresas que produzem e vendem produtos baseados em circuitos integrados consiste na encriptação do *firmware* armazenado no dispositivo. O objetivo é evitar que outras contrapartes consigam examinar o conteúdo do *firmware* e eventualmente tirarem partido dele. Para poder ser executado pela unidade de processamento, o *firmware* é desencriptado durante o processo de inicialização, pelo *bootloader*, utilizando a chave secreta armazenada no dispositivo.

No entanto, é sabido que os dispositivos eletrónicos, inadvertidamente, expõem informações indiretas sobre o processamento interno do dispositivo, da quais é possível tirar partido para adquirir conhecimento sobre o conteúdo que está a ser processado. A análise do consumo elétrico é um tipo de ataque de canal auxiliar bastante eficaz que tira partido da dependência existente entre o consumo instantâneo de um dispositivo eletrónico e os dados por ele processados. Um adversário malicioso, com equipamento relativamente acessível, pode explorar esta dependência para extrair a chave secreta que é utilizada durante a desencriptação do *firmware* pelo dispositivo, para comprometer a eficácia deste mecanismo de segurança em vigor.

Neste trabalho, o meu objetivo é avaliar a segurança de um SoC (*System-on-Chip*) que pode executar tanto implementações baseadas em software como em hardware do algoritmo criptográfico mais amplamente usado em encriptação de *firmware*, o AES (*Advanced Encryption Standard*). O principal foco é demonstrar que as chaves criptográficas processadas pelo SoC podem ser deduzidas através de técnicas de análise do consumo elétrico, a partir de medições do consumo elétrico ou do campo eletromagnético associado. Tal demonstração evidencia a viabilidade de se comprometer o mecanismo de encriptação de *firmware* no dispositivo, e revela, num sentido mais geral, que o dispositivo não deve ser utilizado para fins criptográficos sem a proteção adequada. Por fim, possíveis medidas de proteção, com especial ênfase para a implementação de encriptação segura de *firmware*, são discutidas.

Para alcançar isto na prática, a minha abordagem envolve estabelecer o contexto necessário, investigar o estado da arte neste tópico e subsequentemente, na prática, demonstrar os ataques contra um SoC moderno. Esta componente prática permitirá perceber a que nível ficam expostas a este tipo de ataques as implementações de AES quando executadas no CPU (*Central Processing Unit*) ou no acelerador criptográfico do dispositivo. O conhecimento adquirido e as conclusões obtidas na componente prática permitem uma discussão informada sobre a possível implementação segura, do mecanismo de encriptação de *firmware*, em dispositivos inadvertidamente inseguros.

Palavras-Chave

Ataques de canal auxiliar, Análise do consumo elétrico, Emissão do campo eletromagnético, *Advanced Encryption Standard*, Extração de chave, Encriptação de firmware, *System-on-Chip*, Segurança de hardware, Criptografia.

Abstract

Firmware is a piece of software that provides the low-level control of a device's specific hardware. Due to the nature of firmware, it is inevitable to reason about the security threats and risks involved, for both users and manufacturing companies, and propose concrete solutions to minimize the likelihood of breaches and mitigate their potential consequences. One of many security procedures enforced by companies that manufacture and market IC-based (Integrated Circuit) products is the encryption of the firmware stored inside the device, with the aim of preventing other parties from examining it and potentially abusing it. To be executed by the device's processing unit, the stored firmware is decrypted during the booting process, by the bootloader, using its embedded secret key.

However, devices leak side-channel information about their inner-processing, that can be leveraged and exploited to retrieve secrets. A very powerful type of side-channel attacks is called Power Analysis and it consists of exploiting the fact that the instantaneous power consumed by a device depends on the data that it is processing. A motivated adversary, with relatively low-cost equipment, can take advantage of this dependency to extract the secret key that is used during the decryption of the firmware, and consequently break firmware encryption.

In this work, my objective is to evaluate the security of a SoC (System-on-Chip) that can employ both software- and hardware-based implementations of the widely used AES (Advanced Encryption Standard) algorithm for firmware encryption. The main focus is to demonstrate whether the SoC's hardware inadvertently exposes cryptographic keys to attack, making them susceptible to extraction through PA (Power Analysis) techniques. This indicates the possibility of undermining mechanisms enforcing firmware encryption on the device. Ultimately, protective countermeasures for achieving secure firmware encryption are discussed.

In order to accomplish this, my approach involves establishing the necessary context, researching the state of the art work on this subject matter, and subsequently conducting experiments on an advanced SoC. These experiments aim to assess and document the extent to which the hardware of the SoC is vulnerable to PA attacks. Attacks on software AES-128 leverage power measurements from the device, while attacks against the device's dedicated ASIC (Application-Specific Integrated Circuit) implementation rely on EM (Electromagnetic) emanation measurements. With the insights gained from the literature and conducted experiments, an informed discussion can be pursued regarding the implementation of secure firmware encryption on inherently unprotected devices.

Keywords

Side-Channel attack, Power analysis, Electromagnetic emission, Advanced Encryption Standard, Key extraction, Firmware encryption, System-on-Chip, Hardware security, Cryptography.

Table of Contents

Chapter 1 Introduction 1

1.1 Planning 2
1.1.1 First semester 2
1.1.2 Second semester 3
1.1.3 Risk management 4

1.2 Structure 6

Chapter 2 Background 8

2.1 Firmware Encryption 8

2.2 Side-channel attacks 10

2.3 Leakage of cryptographic devices 12
2.3.1 Power consumption 12
2.3.2 Electromagnetic emanation 13
2.3.3 Leakage models 14
2.3.3.1 Hamming-Distance model 15
2.3.3.2 Hamming-Weight model 16
2.3.3.3 Least Significant Bit model 18
2.3.3.4 Other leakage models 18
2.3.4 Measuring leakage sources 19

2.4 A Brief Introduction to Advanced Encryption Standard 22
2.4.1 High-level description of the algorithm 23
2.4.1.1 Key Expansion layer 24
2.4.1.2 Key Addition layer 25
2.4.1.3 Byte Substitution layer (S-box) 25
2.4.1.4 ShiftRows layer 27
2.4.1.5 MixColumns layer 27
2.4.2 Software implementations 27
2.4.3 Hardware implementations 28
2.4.4 Block cipher mode of operation 31
2.4.4.1 ECB 31
2.4.4.2 CTR 32
2.4.4.3 CBC 33
2.4.4.4 CFB 34
2.4.4.5 OFB 35

2.5 Conclusions 36

Chapter 3 State of the Art 37

3.1 Power Analysis techniques 37
3.1.1 Simple Power Analysis 38
3.1.2 Model-based 40
3.1.2.1 Differential Power Analysis 43
3.1.2.2 Correlation Power Analysis 46
3.1.2.3 Mutual Information Analysis 51
3.1.2.4 Linear Regression Analysis 52
3.1.2.5 Scatter 55
3.1.3 Profile-based 57
3.1.3.1 Template attack 58
3.1.3.2 Stochastic attack 60
3.1.3.3 Machine Learning 61

3.2 Leakage assessment 62
3.2.1 Signal-to-Noise Ratio 62
3.2.2 Welch's *t*-test 64

3.3 Pre-processing 65
3.3.1 Filtering 65
3.3.2 Synchronization 66
3.3.2.1 Sum-of-Differences 66
3.3.2.2 Pearson's correlation coefficient 68
3.3.2.3 Cross-correlation 68
3.3.2.4 Dynamic Time Warping 68
3.4 Countermeasures 69
3.4.1 Physical level 69
3.4.2 Technological level 69
3.4.3 Algorithmic level 70
3.4.3.1 Hiding 71
3.4.3.2 Masking 71
3.4.4 Protocol level 72
3.5 Tools 73
3.6 Conclusions 75
Chapter 4 Experimental Setup and Evaluation 77
4.1 Target device 77
4.2 Preparation 78
4.3 Attacking software-based AES-128 79
4.3.1 Power measurement setup 81
4.3.2 Leakage assessment 84
4.3.3 Non-profiled attacks 86
4.3.3.1 Correlation Power Analysis 87
4.3.3.2 Linear Regression Analysis 89
4.3.3.3 Mutual Information Analysis 91
4.3.4 Profiled attacks 93
4.3.4.1 Template attack 93
4.3.4.2 Machine Learning classifier 94
4.3.5 Conclusion 95
4.4 Attacking ASIC-based AES-128 96
4.4.1 EM measurement setup 97
4.4.2 Leakage assessment 100
4.4.3 Non-profiled attacks 102
4.4.3.1 Correlation Power Analysis 102
4.4.3.2 Linear Regression Analysis 104
4.4.3.3 Mutual Information Analysis 106
4.4.4 Profiled attacks 107
4.4.4.1 Template attack 107
4.4.5 Conclusion 108
4.5 Discussion on protective countermeasures 109
Chapter 5 Conclusion and Future Work 112
5.1 Future work 112
References 114

Acronyms

AC – Alternate Current

AES – Advanced Encryption Standard

ALU – Arithmetic Logic Unit

ASIC – Application-Specific Integrated Circuit

CBC – Cipher Block Chaining

CFB – Cipher FeedBack

CMOS – Complementary Metal-Oxide Semiconductor

CNN – Convolutional Neural Network

CPA – Correlation Power Analysis

CPU – Central Processing Unit

CTR – Counter

CW – ChipWhisperer

DC – Direct Current

DES – Data Encryption Standard

DoM – Difference of Means

DPA – Differential Power Analysis

ECB – Electronic Code Book

EEPROM – Electronically-Erasable Programmable Read-Only Memory

EM – Electromagnetic

FPGA – Field Programmable Gate Array

HD – Hamming Distance

HW – Hamming Weight

IC – Integrated Circuit

IV – Initialization Vector

LRA – Linear Regression Analysis

MIA – Mutual Information Analysis

ML – Machine Learning

MLP – Multi-Layer Perceptron

NIST – National Institute of Standards and Technology

OFB – Output FeedBack

PA – Power Analysis

PC – Personal Computer

PDF – Probability Distribution Function

POI – Point of Interest

ROM – Read-Only Memory

RSA – Rivest-Shamir-Adleman

SA – Stochastic Attack

SCA – Side-Channel Attack

SNR – Signal-to-Noise Ratio

SoC – System-on-Chip

SPA – Simple Power Analysis

TA – Template Attack

UART – Universal Asynchronous Receiver-Transmitter

USB – Universal Serial Bus

XOR – Exclusive OR

Figures List

Figure 1 - First semester work plan 3
Figure 2 - Second semester work plan 4
Figure 3 - Risk exposure matrix 6
Figure 4 - Side-channel monitoring [3]. 11
Figure 5 – Distribution of power consumption when the microcontroller transfers different data from the internal memory to a register [6]. 17
Figure 6 – MOV instruction with different Hamming Weights [6]. 17
Figure 7 – Advanced Encryption Standard input/output parameters [1]. 22
Figure 8 – Advanced Encryption Standard encryption block diagram [1] 24
Figure 9 – In AddRoundKey, the round key is added to the state with an exclusive-or operation [6]. 25
Figure 10 – AES S-box: Substitution values in hexadecimal notation for input byte (xy) [1]. 26
Figure 11 – SubBytes works on the individual bytes of the state [6]. 26
Figure 12 - The two operations that constitute the S-box 26
Figure 13 – Block diagram of the data path on ASIC-based AES implementation, from [6]. 29
Figure 14 – Block diagram of an AES S-box, based on composite field arithmetic, according to [6]. 30
Figure 15 - ECB mode encryption [26]. 31
Figure 16 - ECB mode decryption [26]. 32
Figure 17 - ECB mode weakness [26]. 32
Figure 18 - CTR mode encryption [26] 33
Figure 19 - CTR mode decryption [26]. 33
Figure 20 - CBC mode encryption [26]. 34
Figure 21 - CBC mode decryption [26]. 34
Figure 22 - CFB mode encryption [26]. 34
Figure 23 - CFB mode decryption [26]. 35
Figure 24 - OFB mode encryption [26] 35
Figure 25 - OFB mode decryption [26] 36
Figure 26 – Instantaneous power consumption of an AES-128 encryption run [7] 38
Figure 27 – Closer look at AES-128's first round and second round power consumption [8]. 39

Figure 28 – Comparison of data-dependent power samples, when key-byte is decimal 0 and 47 [7]. 39

Figure 29 – Power trace of a portion of an RSA exponentiation operation [10]. 40

Figure 30 – AES AddRoundKey and SubBytes operations on a byte of data [7]. 41

Figure 31 – First AddRoundKey output for plaintext-byte valued decimal 167 over all 256 possible key-byte values. 42

Figure 32 – First SubBytes output for plaintext-byte valued decimal 167 over all 256 possible key-byte values. 42

Figure 33 – DoM over samples for a key-byte guess ($K' = 10_{dec}$), based on LSB model [6]. .. 44

Figure 34 – High-level description of DPA and CPA attacks [6]. 50

Figure 35 – DPA attack on key-byte [3]. 51

Figure 36 – CPA attack on key-byte [3]. 51

Figure 37 – Transformation of traces portions from time domain to corresponding distributions [17] 56

Figure 38 - Mean traces for the 9 distinct Hamming weights and corresponding Signal-to-Noise ratio [6] 63

Figure 39 – Desynchronization between two power traces 66

Figure 40 – Sliding window synchronization approach. 67

Figure 41 – TinyCrypt's AES-128 encryption function [33]. 80

Figure 42 - Flowchart of general trace acquisition procedure 81

Figure 43 – Power consumption during 10 rounds of TinyCrypt's AES-128 82

Figure 44 - TinyCrypt's AES-128 first round view on the Tektronix oscilloscope 83

Figure 45 - Diagram of power trace acquisition setup 83

Figure 46 – SNR per sample, for each plaintext-byte: software-based AES. 85

Figure 47 – SNR per sample, for each SubBytes output byte position: software-based AES. 86

Figure 48 - SNR per sample, for each SubBytes output byte's HW: software-based AES. ... 86

Figure 49 – Correlation value per sample: CPA attack on 3rd key-byte (software-based AES) 88

Figure 50 – Maximum correlation value per key guess: CPA attack on 3rd key-byte (software-based AES) 88

Figure 51 – Evolution of key-byte guess correlation, as the amount of analyzed traces increases: CPA attack on 3rd key-byte (software-based AES) 88

Figure 52 – R^2 value per sample: LRA attack on 3rd key-byte (software-based AES). 90

Figure 53 – Maximum R^2 value per key guess: LRA attack on 3rd key-byte (software-based AES). 90

Figure 54 – Evolution of key-byte guess R^2 value, as the amount of analyzed traces increases: LRA attack on 3rd key-byte (software-based AES) 90

Figure 55 – Mutual Information value per sample: MIA attack on 3rd key-byte (software-based AES). 92

Figure 56 – Maximum Mutual Information value per key guess: MIA attack on 3rd key-byte (software-based AES). 92

Figure 57 – Evolution of key guess mutual information value, as the amount of analyzed traces increases: MIA attack on 3rd key-byte (software-based AES). 92

Figure 58 – Maximum confidence per key-byte guess: Template attack on third key byte (software-based AES). 94

Figure 59 - Vertical alignment problem between profiling and target datasets: software-based AES. 95

Figure 60 - EM signal on oscilloscope: 10 rounds of TinyCrypt's software-based AES-128 . 98

Figure 61 – Actual EM acquisition setup: probe placement. 99

Figure 62 – EM traces from hardware-based AES-128 encryption. 99

Figure 63 - Plaintext bytes' leakage: ASIC-based AES. 101

Figure 64 - Ciphertext bytes' leakage: ASIC-based AES 101

Figure 65 - HD leakage between bytes of plaintext and of first SubBytes output: ASIC-based AES. 101

Figure 66 - HD leakage between bytes of first AddRoundKey and SubBytes outputs: ASIC-based AES. 101

Figure 67 - Correlation value per key-byte guess in sample 150: CPA attack on 3rd key byte (ASIC-based AES). 103

Figure 68 - Evolution of key-byte guess correlation, as the amount of analyzed traces increases: CPA attack on 3rd key-byte (ASIC-based AES). 103

Figure 69 – R^2 value per key-byte guess in sample 150: LRA attack on 3rd key byte (ASIC-based AES). 104

Figure 70 - Evolution of key-byte guess R2 value, as the amount of analyzed traces increases: LRA attack on 3rd key-byte (ASIC-based AES). 105

Figure 71 - Mutual Information value per key-byte guess in sample 150: MIA attack on 3rd key byte (ASIC-based AES). 106

Figure 72- Evolution of key-byte guess Mutual Information value, as the amount of analyzed traces increases: MIA attack on 3rd key-byte (ASIC-based AES). 106

Tables List

Table 1 - Risk 1 – Full-time internship while doing courses 4
Table 2 – Risk 2 - Working in electronic/embedded systems related field 4
Table 3 - Risk 3 - Unable to acquire exploitable power measurements 5
Table 4 - Risk 4 - Unable to exploit power measurements of device running hardware-based AES 5
Table 5 - Hamming-Distance model: hypothetical power consumptions 15
Table 6 - Switching-Distance model: hypothetical power consumptions 18
Table 7 - Key lengths and number of rounds for AES 22
Table 8 – CPA attack against 3rd key-byte, on first 250 samples: software-based AES. 87
Table 9 – LRA attack against 3rd key-byte, on first 250 samples: software-based AES. 89
Table 10 – MIA attack against 3rd key-byte, on POIs: software-based AES. 91
Table 11 – Template attack against 3rd key-byte, on POIs: software-based AES. 94
Table 12 - Summary of minimum required traces for each attack (scale of x1000): software-based AES. 96
Table 13 - Summary of confidence levels from each attack: software-based AES. 96
Table 14 - CPA attack against 3rd key-byte, on 150th sample: ASIC-based AES-128. 102
Table 15 - LRA attack against 3rd key-byte, on sample 150: ASIC-based AES. 104
Table 16 - LRA coefficients for first six key-byte positions: ASIC-based AES. 105
Table 17 - Summary of minimum required traces for each attack (scale of x1000): ASIC-based AES. 109
Table 18 - Summary of confidence levels from each attack: ASIC-based AES. 109

Chapter 1
Introduction

With the increased integration of embedded systems in our lives, such as smart cards, mobile phones, IoT devices, and many others, concerns have arisen regarding the security guarantees of these systems. Embedded systems strongly rely on the robustness of the underlying implementation of cryptographic algorithms to secure themselves from potential adversaries, thus the fact that these devices are often physically accessible, makes them more prone to be exploited for vulnerabilities. Different cryptographic algorithms, such as AES, Twofish, RSA (Rivest-Shamir-Adleman), or others, can be implemented depending on the performance, computational cost, power, and security requirements of a specific resource-limited system. As embedded systems are usually very limited in terms of available computational resources, symmetric-key algorithms are predominantly used in these systems for data encryption and preferred over asymmetric-key algorithms whose operations are more computationally intensive and slow. Mathematical assumptions and the cryptographic design behind these algorithms determine how these are computationally infeasible to break through classical cryptanalysis and brute-forcing. However, in contrast to attacks that exploit flaws in the design of the algorithms, side-channel attacks take advantage of information that is inadvertently leaked by the device executing the algorithm. A Side-Channel Attack (SCA) involves monitoring and analyzing extra information that can be gathered from the hardware running the algorithm, such as heat, sound, time, electromagnetic radiations, or power consumption, during the execution of an operation of interest, as a means to infer some secret information that the target device may be leaking.

From computer peripherals to sewing machines, embedded devices need to be equipped with firmware that provides the low-level control of the device's specific hardware. To prevent adversarial parties from dumping the often-proprietary firmware and abusing it for illegal purposes (e.g., counterfeiting exact replicas of the device, reverse-engineering it to gain competitive advantage, or compare firmware versions to find vulnerabilities that have been patched in recent versions but still impact devices running older firmware versions), firmware is kept encrypted in the device's flash memory and is decrypted with the encryption key in-memory during the secure boot procedure. Justified by its high efficiency, on both hardware and software implementations, AES is a symmetric-key algorithm, widely used not only for Firmware Encryption but also for other cryptographic applications, resisting differential and linear cryptanalysis due to the usage of rounds and nonlinear transformations, but usually susceptible to key extraction through SCAs when implemented on unprotected devices.

A very powerful form of SCAs is entitled PA in which the varying power consumption of the microcontroller, during a cryptographic operation, is measured and analyzed in order to deduce the secret key. Depending on the data or code instruction being processed in the device, its power consumption varies, leaking the internal characteristics of the processing. As such, when a device is processing cryptographic secrets, its data-dependent power usage can expose those secrets to attack. Making it

possible to break AES as well as other widely deployed cryptographic algorithms, PA attacks have become a serious security issue for devices relying on cryptography.

All things considered, there seems to be a great motivation in gaining a deeper understanding of the state-of-art techniques for PA targeting devices running AES, and researching possible countermeasures against these attacks, to be integrated in the design and development of more secure systems, with particular interest in securing firmware encryption.

Even though this work is mainly focused on breaking AES-128 through the exploitation of side-channel information from power consumption and EM emanation, most principles apply to other cryptographic algorithms or applications running on embedded devices and may be adapted as such.

This project was proposed by Nabil Hamzi, Chief Product Security Architect at *Logitech*, and was conducted at *Logitech Europe S.A.* corporate offices in Lausanne, Switzerland. Logitech is a Swiss-American multinational manufacturer of computer peripherals and software, and is considered one of the world's leading manufacturers of input and interface devices for PCs (Personal Computers) and many other digital products. These include products like keyboards, mice, tablet accessories, headphones and headsets, webcams, Bluetooth speakers, universal remotes and many more. It is worth emphasizing that the device experimented with in this document is not manufactured by Logitech.

1.1 Planning

This project was divided into two semesters, the first and the second semesters.

1.1.1 First semester

Task 1 – Research and study the state-of-art side-channel attacks with special focus on Power Analysis techniques. Explore leakage assessment, pre-processing mechanisms and power trace acquisition setups and procedures. Research commonly used firmware encryption techniques. Understand how Power Analysis attacks can be conducted in a real-world setup. Gather the relevant details and information concerning the studied content, from various sources, in a document.

Task 2 – Explore the open-source tools for Power Analysis available online, get comfortable with the preferred tools and successfully conduct attacks on a publicly available dataset of simulated power traces, from CHES 2016 Capture-the-Flag challenges, with different characteristics (e.g., unsynchronized traces).

Task 3 – Develop own tool to conduct the state-of-art attacks, leakage assessment, and pre-processing mechanisms studied during Task 1, and test its efficacy on the simulated dataset.

Task 4 – Develop firmware for the targeted microcontroller, to run both software-based and hardware-based implementations of the Advanced Encryption Standard encryption algorithm. Allow the communication between the computer and the target microcontroller, via UART serial communication, to enable setting the encryption parameters, triggering the execution of the encryption algorithm, and retrieving the resultant ciphertext.

Task 5 – Instrument the oscilloscope for power trace acquisition.

Task 6 – Find the proper measurement setup (where in the circuit, and how to perform the power measurement), that allows to measure data-dependent instantaneous power consumption that can be exploited with power analysis.

Task 7 – Acquire a large number of power traces from the target while it is encrypting random known data using the software-based AES implementation.

Task 8 – Perform leakage assessment on the acquired dataset from Task 7 and depending on the characteristics of the data perform the most suitable attacks. Interpret results and prepare a presentation for the company.

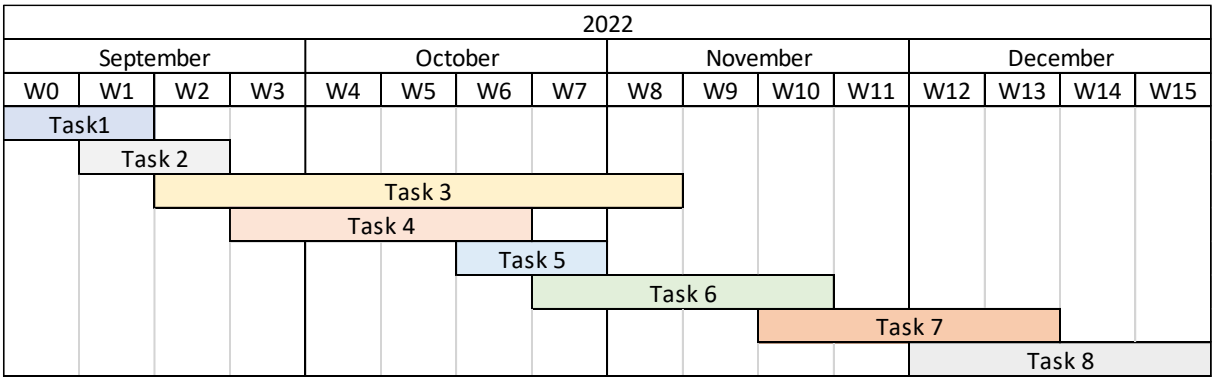

Figure 1 - First semester work plan

1.1.2 Second semester

Task 9 – Find the power measurement setup that allows to acquire power traces associated with the execution of the ASIC-based AES computation. Acquire a large number of power traces from the target while it is encrypting random known data using the ASIC-based AES implementation.

Task 10 – Discover the power model that describes the handling of data inside the cryptoaccelerator component of the SoC by assuming possible hardware-based AES architectures and corroborate it by performing leakage assessment on the acquired dataset from Task 9. Depending on the characteristics of the data, conduct the most suitable attacks.

Task 11 – Research firmware encryption approaches and the state-of-art countermeasures against the attacks conducted before. Document the learning outcome.

Task 12 – Scrutinize the state-of-art countermeasures and discuss their potential implementations on the previously targeted device, for the purpose of secure firmware encryption. Do it for both CPU based implementation (software AES), and dedicated ASIC implementation (hardware AES running on crypto block).

Task 13 – Assess the effectiveness of the previously implemented countermeasures against state-of-art Power Analysis attacks, by evaluating how they reduce the attack surface while considering the trade-off between performance, security guarantees, feasibility, and the required effort to break them.

Task 14 – Prepare and present a final presentation regarding the work conducted throughout the internship and the achieved outcomes.

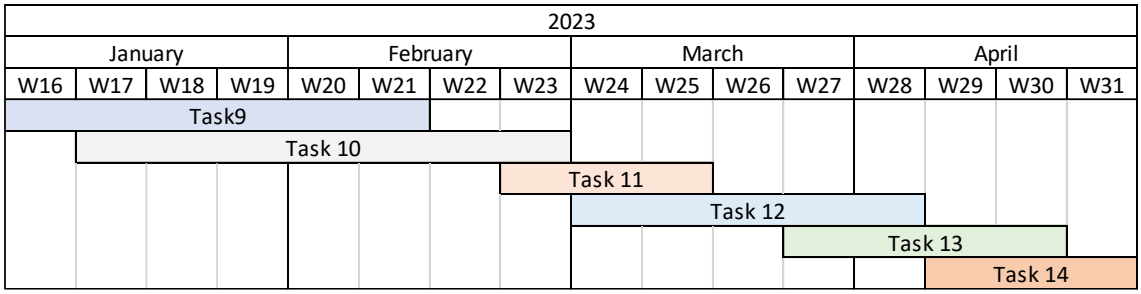

Figure 2 - Second semester work plan

After this period, the present dissertation document was improved and finalized for review and final delivery in June/July 2023.

1.1.3 Risk management

During the development of a project, unforeseen and unfavorable events may arise. These may jeopardize progress and force the modification of existing plans. Therefore, it is essential to contemplate and evaluate potential risks that can impact the success of the project.

Accordingly, the aim was at identifying these risks, calculating their probability of occurrence and associated level of impact on the overall project, while proposing a mitigation strategy to either prevent potential risks or to adjust the methods for accomplishing specific tasks in a controlled manner. Thereafter, the risks are classified according to their probability of occurrence, on a scale of "0 – Very Low Probability/Rare" to "5 – Very High Probability/Very Likely", and consequent impact, on a scale of "0 – No Impact/Insignificant" to "5 – Very High Impact/Severe". A status on whether the risk has been observed, meaning it occurred, is provided.

Table 1 - Risk 1 – Full-time internship while doing courses

Risk	Full-time internship while doing courses.
Description	Working full-time during the internship while doing other courses from the master's degree might affect the quality of the deliverables for both the internship and the courses.
Probability	4
Impact	4
Mitigation	Plan and organize the time well, inside and outside the internship to accomplish both.
Status	Observed.

Table 2 – Risk 2 - Working in electronic/embedded systems related field

Risk	Working in electronic/embedded systems related field.
Description	Since I don't have previous experience with hardware topics, much less with hardware security, it will be challenging to acquire the required understanding in this field.

Probability	5
Impact	3
Mitigation	Place an extra effort in understanding the basics and don't be afraid to ask questions, to obtain a good understanding over everything that I'm working on.
Status	Observed.

Table 3 - Risk 3 - Unable to acquire exploitable power measurements

Risk	Difficulties measuring power associated with AES computation.
Description	Since experiments are being conducted on a SoC for which there is no public information on power management or past PA attacks against the target, the task of finding a proper measurement setup for attacking the device will be more challenging. Also, it's unknown whether the SoC implements protective countermeasures against SCAs. These can have a substantial impact over the planning of the project.
Probability	4
Impact	5
Mitigation	The success of the project depends on the success of the measurements. Accordingly, tasks may be adapted to fulfil time constraints, such as not conducting the analysis on the ASIC-based implementation of AES or dedicating less time to assessing potential protective countermeasures. If the microcontroller employs countermeasures, the tasks related with the power analysis techniques, pre-processing, etc, may take considerably more time to achieve the desired outcome.
Status	Observed (difficulty of finding proper measurement setup).

Table 4 - Risk 4 - Unable to exploit power measurements of device running hardware-based AES

Risk	Unable to exploit power measurements from device running ASIC-based AES.
Description	The ASCI-based implementation of AES is executed on a dedicate cryptoprocessor that is independent from the general purpose CPU of the SoC. As such, the power consumed by the AES computation on the cryptoprocessor may be difficult to measure from the circuit, either because its power consumption is negligible or because SCA countermeasures are employed. Also, the implementation details about this implementation on hardware are not publicly available which adds a layer of difficulty to the attack.
Probability	4
Impact	5
Mitigation	Exploring alternative leakage sources such as the EM emanation of the cryptoprocessor holds potential. The measurement of the EM field presents additional challenges such as noise interference or guaranteeing probe stability. If countermeasures are in-place, power analysis related tasks will take considerably more time, and as for Risk 3, some workload may have to be reduced.
Status	Observed (unable to measure power associated with cryptoprocessor computation).

In Figure 3, the identified risks are placed in a Risk Exposure matrix in function of their probability of occurrence and impact level in the absence of prior consideration, according to the defined scale.

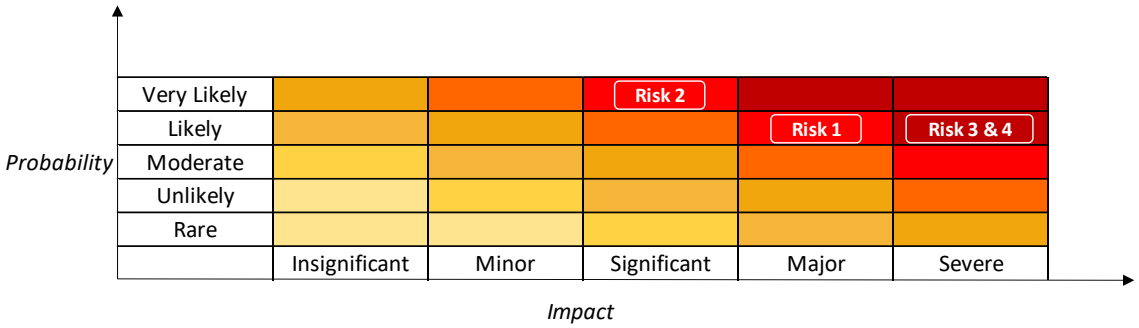

Figure 3 - Risk exposure matrix

Since these risks and corresponding mitigations were considered in advance, their impact was highly reduced. For instance, the inability of finding data-dependent leakage on the power measurements associated with the activity of the dedicated cryptoprocessor, could render the attacks against this implementation infeasible and result in inefficient use of time, if the EM side-channel was left unexplored.

Despite the observation of Risk 1, its impact on the overall outcome of both the internship and the courses was minimal, at the expense of additional time and effort.

The occurrence of Risk 2 had the most significant impact at the beginning of the internship when I started delving into the concepts and practical work. This was expected as I had no prior hands-on experience with hardware, let alone hardware security. However, I believe it provided me with extra motivation to learn new things and prevented me from rushing past background knowledge. I was encouraged to ask questions that helped me build the foundation and propelled the overall enthusiasm I felt throughout the internship.

Regarding Risk 3, finding the proper measurement setup was challenging for attacking both software and hardware-based implementations. Consequently, this stage required a significant amount of time and was deemed the most crucial part to the success of the experimental work. Still, the time was managed and well allocated according to the established deadlines.

The process of recognizing Risk 4 involved spending time acquiring power measurements that ended up not being exploitable for the purpose of key extraction. Furthermore, there was an additional time overhead associated with the challenge of determining the location and positioning of the probe to acquire EM measurements that captured leakage from the cryptoprocessor's activity.

1.2 Structure

The present document is organized in five main chapters. The **Introduction** (Chapter 1), the **Background** chapter (Chapter 2), the **State of the Art** chapter (Chapter 3), the **Experimental setup and evaluation** chapter (Chapter 4), and the **Conclusion and Future Work** (Chapter 5).

Chapter 1 – In the **Introduction** chapter the motivation, goals, and desirable outcome of this work are defined. The planning of the project and the way the work was organized throughout the internship is displayed with the tasks and timelines for each semester being described. The structure of this document is presented.

Chapter 2 – The **Background** chapter includes an overview on firmware encryption practices and side-channel attacks. Context on the leakage from the power consumption and EM-field of embedded devices, as well as a survey on common approaches for measuring these leakage sources is provided. Leakage model assumptions are introduced. Finally, the Advanced Encryption Standard is presented.

Chapter 3 – In the **State of the Art** chapter, power analysis techniques are introduced and detailed, to provide the reader with the ability to critically reason about the advantages and drawbacks of some methods over the others and make informed decisions on which to choose, given the nature of the measurements under analysis. Leakage assessment mechanisms to detect and quantify the data-dependency in the instantaneous power consumption of the device, as well as pre-processing mechanisms are presented. The state-of-the-art countermeasures against the detailed attacks are researched, and at last, in this chapter, a set of available open-source tools are mentioned.

Chapter 4 – In the **Experimental setup and evaluation** chapter, a set of the state-of-art Power Analysis attacks along with leakage assessment techniques are translated into practice. The goal is to evaluate whether the device targeted in this document is vulnerable to key extraction, on both software and hardware-based implementations of AES, through PA techniques. The accuracy, efficiency, and limitations of the techniques used are examined from the obtained results. The use of countermeasures for the purpose of securing the process of firmware encryption in the device is discussed. These can be extended for other devices or applications that align with the described context. Relevant considerations from the experimental process are drawn.

Chapter 5 – In this chapter, conclusions concerning the results and the outcome of the internship are drawn, alongside a reflection on potential future work.

Chapter 2
Background

Modern systems leverage cryptographic algorithms to provide confidentiality, integrity, and authenticity of data [6]. Cryptographic algorithms are mathematical functions that typically take two input parameters: a message (referred to as the plaintext) and a cryptographic key. These parameters are mapped to an output, referred to as the ciphertext, in the so-called encryption process. The fact that electronic devices are used to perform cryptographic algorithms leads to a new issue for the practical security of the algorithms. In practice, not only the security of the cryptographic algorithm is of interest. The security of the whole system (i.e., cryptographic algorithm, cryptographic key, and device running cryptographic implementation) needs to be considered.

2.1 Firmware Encryption

Firmware is the piece of software that provides a device with the control over his specific hardware [34]. Take the example of an IoT device such as a vacuum cleaning robot that is able to clean the house, automatically and in a seemingly intelligent way. This device contains hardware peripherals such as wheels connected to an electric motor, distance sensors, and most likely a Bluetooth interface to allow the robot to exchange information with other nearby devices. All these hardware parts of the device, that work interconnectedly with each other to achieve the main goal of cleaning the house (e.g., the sensors allow the robot to map the house and avoid hitting the walls, the wheels allow the robot to move around, etc.), are controlled by the firmware that is executed inside the product's microcontroller, thus the device depends on the firmware to achieve its intended functionality.

Given its role, it becomes clear how important it is to secure the firmware. In the previous example, if an attacker was able to reprogram or modify the firmware running inside the robot, he could leverage the device's peripherals, for instance the distance sensor, to collect sensitive information regarding the mapping of the house, unnoticedly. Also, take the example use-case of smart-home door lockers that are used to allow the locking and unlocking of doors inside a home, using for instance a mobile application. If an adversary is capable of dumping and analyzing the firmware of this product, he might find bugs that can be exploited to surpass an authentication mechanism in-place to control the locking/unlocking of doors, and ultimately abuse this vulnerability to rob houses without having to force the entry.

Compromising firmware can have a tremendous harmful impact on the device manufacturer and end-users involved. These hypothetical examples expose the importance of guaranteeing security requisites such as the confidentiality, integrity and authenticity of the firmware [35], which can be achieved by the employment of cryptographic mechanisms such as the verification of digital signatures based on public key encryption, to verify that the firmware hasn't been tampered with and that it comes from a trusted source (i.e., integrity and authenticity), and the encryption of the firmware image that's stored inside the device's flash ROM (Read-only memory), as a means to safeguard its confidentiality, to ultimately prevent adversarial parties from dumping and analyzing the firmware with the intent of abusing it for

illegal purposes [10]. These requirements are typically enforced during the booting and the firmware update procedure of the device [25], as follows:

1. During manufacturing, the secret key required to decrypt the firmware is embedded into the device's storage, usually the SoC's internal flash memory, or preferably a secure area of the device's memory that is protected against unauthorized access or modification. A company's public key is also stored, to later be used by the device to verify the firmware's digital signature [9].
2. At the moment of manufacturing or in the event of a firmware update, the encrypted firmware image is written to the flash memory or EEPROM (electronically-erasable programmable ROM) of the device. Somewhere in this process, before writing it to memory, the device should verify whether the software hasn't been tampered with and comes from a trusted source [35]. Also, at boot time, before executing the decrypted firmware, the same should be validated. There are many distinct ways of achieving this goal in practice, over certain assumptions, as illustrated in the following examples:

   Considering,

   - F : firmware image;
   - *Enc()* : symmetric encryption function, hence *Enc(F)* describes encrypted firmware image;
   - *h()* : hash function;
   - *Sig()* : signature function;
   - + : concatenation.

   a) *Enc(F + Sig(h(F)))*

   The hash of the firmware is digitally signed, and this signature is concatenated to the firmware image, which are both consequently encrypted using a symmetric cryptographic algorithm of choice. This design option implies that the encrypted firmware must be first decrypted to obtain F, and only afterwards the signature verification can take place.

   b) *Enc(F) + Sig(h(Enc(F)))*

   The signature of the hash of the encrypted firmware is concatenated to the encrypted firmware image. As such, the microcontroller can verify the authenticity and integrity of the encrypted firmware, before proceeding with its decryption.

   c) *Enc(F + Sig(h(F)) + Sig(h(Enc(F + Sig(h(F)))) = a + Sig(h(a))*

   Two signatures allow the device to verify both the encrypted and the decrypted firmware. It's interesting to understand the advantages of this solution over the others, from an integrity and authentication point-of-view.

   Take the following hypothetical example of a device using a flash memory that is external to its microcontroller. When the device is started, the bootloader would verify the signature of the encrypted firmware *Sig(h(Enc(F + Sig(h(F)))))*, and given a valid signature it would decrypt and place the firmware in the external flash. Without the signature *Sig(h(F))* of the firmware image, the chip would load the decrypted firmware from the external flash and execute it in the CPU under the assumption that it had already been verified by the bootloader, and indeed the

signature of the encrypted firmware was validated, but that's not enough. A malicious attacker could interrupt this procedure by putting the device at rest (i.e., interrupting the power to the chip), tamper the decrypted firmware inside the external flash, and then let it continue so that the microcontroller would load and execute a modified version of the firmware instead, thus the importance of having the microcontroller verifying the signature of the decrypted firmware before executing it. This shouldn't be a problem in the case where the flash memory is internal to the microcontroller, because to write to it, one must have access to the debug interface/pins that should only be available during development and closed in manufacturing. The provided example serves the purpose of demonstrating how designing security for electronic devices is not a straightforward task.

3. The safe and trusted firmware is kept encrypted inside the device's flash memory and is decrypted by the bootloader, using the embedded secret key, to be executed by the processing unit.

For the interest of this thesis, particular attention is devoted to the encryption and decryption of firmware, which shall not be confused with digital signing. Readers who are interested in more information regarding firmware security, are referred to [9], [34], [35] and [36]. In [25], the specification and implementation details of a typical bootloader with AES encryption are described.

Firmware encryption is a common industry practice. Despite undisclosed software casting suspicion around poor programming and backdoors, this decision might be beneficial for both product users and company [36]. Without this layer of secrecy, an attacker could, for example, study the patches applied between firmware updates to find and identify the flaws that were fixed and consequently target users of unpatched devices. Moreover, firmware in the clear, can be leveraged by illegitimate parties for product counterfeiting or even to gain unfair competitive advantage over the proprietary company.

In this work, the AES-128 decryption process of the firmware, that is carried out during the booting of the device, is targeted with the intent of extracting the secret key. Nonetheless, the intuition of the attacks and analysis covered in this document can be used and is of interest to target any cryptographic mechanism running on a microcontroller. The Advanced Encryption Standard is one of the most widely used cryptographic algorithm for the encryption and decryption of firmware, since it can be efficiently implemented, in both software and hardware, which is particularly relevant for resource constrained devices such as microcontrollers, while providing good security guarantees against various cryptographic attacks.

2.2 Side-channel attacks

The security of a cryptosystem (cryptographic algorithm, cryptographic key, and device running cryptographic implementation) doesn't depend only on its theoretical quality (e.g., use of robust algorithms and parameters, certified protocols, and long enough cryptographic keys), but also on its robustness against physical attacks.

A SCA is a physical attack that aims to exploit the dependency between the secret information being processed and the resulting physical leakage of the device, to break the system [3]. It is carried out by monitoring and analyzing the physical outputs of a system (e.g., power consumption, electromagnetic emission, timing information, emission of heat, sound),

as depicted in Figure 4, while it's conducting cryptographic operations, in order to deduce secret information from the processing, like cryptographic keys.

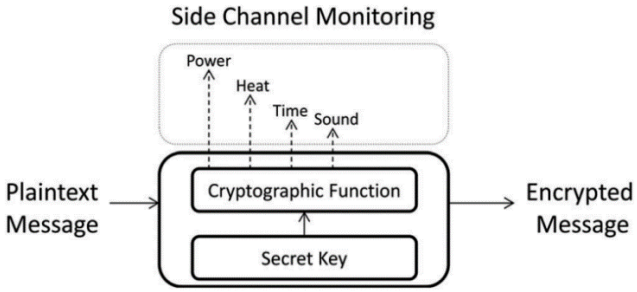

Figure 4 - Side-channel monitoring [3].

Unsecure PIN verification mechanisms are easily broken with SCAs. In the following naive implementation of a PIN verification algorithm, by going through a trial-and-error process of guessing each PIN digit, incrementally, a change in the power trace or time consumption from its execution, unequivocally leaks the correct digit. As this function checks digit by digit and immediately returns when a mismatch between the inserted and the valid digit is detected, a malicious user who can run the program, can measure the amount of time taken or the power consumed by the function for a given trial PIN, and use this information to learn the valid PIN, digit by digit. As simple as this example (1) might look, it provides an interesting ground for further reasoning about side-channel leakages.

```
for i in range(4):
      if input_pin[i] != valid_pin[i]:
            print("Wrong PIN")
            exit()
print("Valid PIN")
exit()
```
(1)

For the scope of this thesis, the focus was placed on two very powerful types of SCA, denominated Power Analysis and EM Analysis, which are based on the analysis and exploitation of data dependency in power consumption or electromagnetic emission as a means to learn information about the data being processed by the microcontroller [14], and ultimately deduce the secret cryptographic key that breaks the firmware encryption. The fact that the electromagnetic emanation of a microcontroller is directly related to its power consumption, makes it an important information source whose data dependency can be further exploited, under similar assumptions as of PA [39]. In the available literature, these have been demonstrated to be extremely powerful and effective in practice [6], mountable with relatively simple and affordable equipment, hence posing a serious threat to the security of cryptographic systems running on microcontrollers. Today, most embedded devices found in consumer products or even critical infrastructure are exposed and vulnerable to this type of attacks [38] which can result in unprecedented damage and losses for both customers and companies, therefore, the prospect of conducting research and adding value to this particular area of study is highly motivating. Manufacturing companies should assess the potential vulnerability of their products to such attacks as well as the risks involved, and consequently invest the necessary efforts into the research and development of protective countermeasures.

2.3 Leakage of cryptographic devices

As mentioned in the previous section, there are many potential leakages in cryptographic devices that can be exploited in a side-channel attack for the purpose of key extraction. However, the scope of this thesis is confined to the analysis of power consumption or electromagnetic emanation. As such, this section provides relevant context on the source of these two types of leakages, how they relate to each other, how they can be modeled for the purpose of PA attacks, and finally some considerations on how they are typically measured.

2.3.1 Power consumption

As further explained in Section 2.4.2 and 2.4.3, there are essentially three ways of implementing a cryptographic system on a chip: on a general-purpose CPU, a FPGA (Field Programmable Gate Array), or an ASIC. In any of these cases, the dedicated hardware is composed of a digital circuit able to perform computations according to sets of instructions or a logic design.

The smallest building blocks of all digital circuits are known as logic cells. They take one or more logic values as input and output a logic value according to their logic function. There are two fundamental types of logic cells. Combinational cells implement basic Boolean logic functions like NAND, XOR, or others where their output values are a logical combination of their input values. On the other hand, sequential cells implement more complex functions like latches, flip-flops, and registers, in which their output values not only depend on their current input values, but also on preceding input values or on their initial state, meaning sequential cells memorize previous input values [6].

These logic cells are implemented using transistors. Transistors are essentially electronic switches that consist of a special arrangement of so-called p-type and n-type semiconductor structures [44]. Although there are various types of transistors, the majority of digital logic circuits developed in the 21st century is based on CMOS (Complementary Metal-Oxide Semiconductor) transistor cells due to several advantages, one of them being the lower power consumption during idle periods comparatively to other types of transistors [37][44]. There are two main components that constitute the power used by a CMOS integrated circuit: static power and dynamic power. Static power refers to the power consumed by the transistor when it is in an idle state and not actively switching, while dynamic power is consumed when switching occurs in transistors, that is, when a CMOS cell changes from 0_{bit} to 1_{bit} or from 1_{bit} to 0_{bit}, and therefore it is dependent on the operation being executed and the data being processed. Since power analysis leverages such dependency, dynamic power is the relevant one for the present work. Also, since the static power is mostly constant, any variation in the total power can be attributed to the dynamic power component [23]. As such, the total power consumption can be directly considered when carrying out an attack. In [48], by measuring the power consumption of a simple CMOS gate during bit transitions, *Peeters et al*. observed that charging the CMOS gate consumed more than discharging it, implying that transitioning from a bit state 0 to 1 consumes more power than transitioning from bit 1 to 0. Note that charging a CMOS gate means applying a voltage level to its input allowing current to flow between the input and output terminals, resulting in the gate's output being set to logic high (i.e., bit 1 state). In contrast, discharging means applying a voltage level that turns off the transistors in the gate, interrupting current flow between the gate's input and output terminals, resulting in the output being set to logic low (i.e., bit 0 state).

Cryptographic systems running on computing platforms rely on electronics to manipulate bits 1's and 0's. Internally, the devices have integrated circuits that consist of logic gates which

are composed of specific arrangements of transistors combined with other electrical components like resistors and diodes. As noted earlier, to change a bit 1 to a 0 or vice versa, a current I is applied or removed from each transistor, thus power is consumed. At these moments, peaks are visible in the instantaneous current drawn by the circuit, meaning the power consumption reflects the aggregate activity of its individual elements, as well as the capacitance and other electrical properties of the system. In microcontrollers, the data bus which is often long and connected to many components has a quite big capacitance (i.e., ability of a component or circuit to store electric charge) [6]. Writing to the data bus during a memory read or write causes a significant power consumption, therefore depending on the data or the code instruction being processed in the device, its power consumption varies, leaking the internal characteristics of the processing. When a device is processing cryptographic secrets, its data-dependent power usage can expose those secrets to attack. This is particularly useful to disclose cryptographic keys by analyzing the intermediate data processed during an algorithm. Using statistical tools, the key can be fully retrieved by pieces using a divide-and-conquer approach [36].

By disregarding the static power component according to the reasoning presented earlier, the instantaneous power consumption of an electronic device can be described as follows:

$$P_{instant} = P_{operation} + P_{data} + P_{algo\_noise} + P_{elec\_noise} \quad (2)$$

- $P_{operation}$ is the current drawn by logic cells to perform an arithmetic operation.
- P_{data} is the current caused by the internal manipulation of the data (e.g., instruction execution, memory read/write, or transmission on buses).
- P_{algo_noise} refers to the algorithmic noise caused by a parasitic activity (e.g., irrelevant data processed in parallel, hardware peripherals, or multi-core CPUs).
- P_{elec_noise} refers to the electronic noise that is a combination of the physical noise (e.g., power supply, clock, radiations) and the measurement noise (e.g., quantization error).

Both $P_{operation}$ and P_{data} are potential sources of exploitable leakages.

2.3.2 Electromagnetic emanation

Just as the power consumption of CMOS devices, its electromagnetic radiation can be shown to be data-dependent [45]. From a theoretical point of view, electromagnetic leakages are usually explained from the Biot-Savart law. The law, in (3), expresses the contribution to the magnetic field, denoted *dB*, of some point p in the conductor, where μ is the magnetic permeability of the medium, I is the current carried on the conductor of infinitesimal length *dl*, and scalar r and normalized vector $\hat{r}$ describe the distance and orientation, respectively, from p to a field point in space, which means that the magnetic field generated by a steady current I, flowing through a conductor, at a point in space, is proportional to the current, the length of the conductor, and the sine of the angle between the conductor and the vector connecting the point to the conductor.

$$dB = \frac{\mu * I * dl * \hat{r}}{4 * \pi * r^2} \quad (3)$$

Even though the emanation from an integrated circuit is complex and cannot be fully described by a simple equation, Biot-Savart's law emphasizes the important fact that the EM field is data-dependent since it directly depends on the current intensity [45], and therefore it is an exploitable source of leakage whose variation can be analyzed using the same techniques of Power Analysis. It's also worth noting that the field's orientation depends on the current

direction because it is characterized by the cross product between the current direction and the position vector [37].

Electromagnetic emissions are generated when current flows within the control I/O, data processing, or other parts of an electronic device. These emissions are specific to each component and are dependent on their unique physical and electrical properties. Any movement of electric charges is accompanied by an electromagnetic field [47]. Attackers are particularly interested in the emanations resulting from data processing operations. In CMOS devices, ideally, current only flows when there is a change in the logic state of the circuit, i.e., when the input values change, the transistors in the logic cells switch on and off, allowing current to flow through the circuit. Such emanations contain valuable information about the current flow and the events occurring during each clock cycle. Most modern devices pack a large number of circuits and components into a very small area which results in the modulation of various fields emitted by distinct elements, meaning that a field generated in the circuit is coupled with other electrical and electromagnetic fields originated by surrounding components depending on their proximity [37][39]. Coupling refers to the association of the fields, while modulation denotes the variation that occurs as a result of this association, meaning the resulting signal consists of the original signal affected by changes in amplitude, frequency, or phase of the EM wave due to the influence of the other signals. Therefore, it might be possible to observe leakage in otherwise uninteresting signals, such as the clock signal, due to coupling with data-dependent signals of the circuit [37].

Not only the EM field can encode the data-dependent leakage associated with the power consumed by the device, it can also leak information regarding the direction of the current flow inside the measured circuit that can be leveraged, for example, to reverse engineer the chip and locate functional blocks, or even focus on the activity of particular components of the microcontroller by concentrating the EM field capture on them [45]. Accordingly, electromagnetic emanations may offer a more comprehensive source of leakage information than power signals. Nonetheless, the quality and strength of the EM signal can make it considerably more difficult to exploit this particular side-channel [37].

2.3.3 Leakage models

To conduct Power Analysis attacks, we often need to assume a leakage model, so that data values, processed by the target device, can be mapped to hypothetical power consumption values. The leakage model, also referred to as the power model, describes the dependency between the data and the power that is consumed by the device to process that data. Most commonly, attackers have very limited knowledge regarding the internal workings (e.g., power management) of the target device, thus the assumed leakage models are often simplistic and abstract from the inner complexities of the system. Nevertheless, the closer the leakage model aligns with reality, the more effective and successful the attack will be. Good leakage models have a strong impact on the efficiency of a side-channel attack [45].

Recall that, in the context of PA, the absolute values of the power consumption are not relevant. Instead, only relative differences between power consumption values are important. Therefore, throughout this thesis, the term ‘power consumption’ is used interchangeably to refer to any direct or indirect measurement of power consumption (i.e., voltage drop across resistor, EM emanation, or other) that can be exploited using power analysis techniques. As mentioned in Section 2.3.1, the power consumption of CMOS circuits depends on the transitions of the logic gates, and therefore on the operations and data manipulates by the circuit. With respect to data-dependency, whenever data is loaded into a cryptographic device’s bus, the transitions that are triggered in the CMOS cells, are related to the change in bits of the value being loaded and

of the value that was previously on the bus [37]. From these physical properties, leakage models have been derived.

In this section, the Hamming-Distance and the Hamming-Weight leakage models, which are commonly used for PA attacks, are introduced. Then, the Least Significant Bit model is presented. Furthermore, other variants and alternatives of these models are briefly discussed. All these models allow describing the data-dependent power consumption of a microcontroller as well as its electromagnetic behavior [48].

2.3.3.1 Hamming-Distance model

When targeting an embedded device, high chances are that it is structured similarly to publicly known devices, with registers, a data bus, memory, an ALU (Arithmetic Logic Unit), a communication interface, etc. These components have well known properties, such as the data bus being generally quite long and connected to multiple components, resulting in a significant capacitive load that substantially contributes to the overall power consumption of the microcontroller. Additionally, it can be reasonably assumed that the capacitive loads of all the individual wires within the bus are approximately equal [6]. Reflecting on these observations, the suitability of the Hamming-Distance model for describing the power consumption of data buses becomes evident. Other types of buses, such as address buses, exhibit similar characteristics. Besides the power consumption of buses, also the power consumption of registers, in hardware implementations of cryptographic algorithms, can be described very well by the HD model. Registers are triggered by a clock signal, and so their value is only changed once per clock cycle [6].

The Hamming-Distance model assumes that the power consumption that is caused by changing the data bus from a value x_0 to a value x_1 is proportional to $HD(x_0, x_1)$. Hence, when a value x_0 contained in a CMOS device is changed into a value x_1, the side-channel leakage of such transition is correlated with the Hamming distance between these two values [45]. The assumption is that, for a bit to transition from 0→1 or from 1→0 a charge must be applied to a bus line, thus power is consumed. Accordingly, if a data bus has four lines, meaning it can transfer four bits of data simultaneously, to go from x_0 to x_1, where x_0= 1010 and x_1= 0011, in binary representation, only two bits change in the bus and therefore the Hamming distance is equal to 2, which represents the power contribution to process those four bits from one state (x_0), to the other (x_1). Note how it is assumed that 0→1 and 1→0 bit transitions contribute equally to the power consumption with +1, while no power is considered to be consumed during bit transitions from 0→0 and 1→1. These contributions are summarized in Table 5.

Table 5 - Hamming-Distance model: hypothetical power consumptions

Transition	Power
$0 \rightarrow 1$	1
$1 \rightarrow 0$	1
$0 \rightarrow 0$	0
$1 \rightarrow 1$	0

Due to the physical aspects of CMOS circuits, where the significant power consumption occurs during transitions [23], these restrictive assumptions are quite realistic and have been widely referenced and exploited throughout the available literature [5][6][14][39]. They lead to a convenient expression for the leakage model, describing the number of flipping bits to go from x_0 to x_1, which is given by the Hamming distance between x_0 and x_1, that is

mathematically equivalent to computing the Hamming weight of the XOR between the two values, as in (4).

$$HD(x_0, x_1) = HW(x_0 \oplus x_1) \tag{4}$$

The Hamming weight corresponds to the number of bits that are set to one, and hence, $HW(x_0 \oplus x_1)$ corresponds to the number of bits that differ in x_0 and x_1.

$$HW(z) = \sum_{i=0}^{7} b_i \quad \text{with } z = (b_0 b_1 \dots b_7)_{\text{bin}} \tag{5}$$

Recall, registers are usually implemented using flip-flops or other types of memory cells, based on transistors, that store data as binary values (0's and 1's). To change a bit stored in a register, a specific voltage level is applied to the input of the corresponding flip-flop or memory cell, which involves the movement of electric charge, and therefore the consumption of power. Now, consider, for instance, an 8-bit register holding the decimal value 4 = 00000100_{bin}. Changing the value of this register from decimal 4 to decimal 125 = 01111101_{bin}, requires five bits to be flipped. Under the HD model, the power consumed to change the register's value is proportional to the number of bits that must be flipped, which corresponds to the Hamming distance between the previous and current value: $HD(4_{\text{dec}}, 125_{\text{dec}}) = HD(0\underline{0000}10\underline{0}_{\text{bin}}, 0\underline{1111}10\underline{1}_{\text{bin}}) = HW(0\underline{0000}10\underline{0}_{\text{bin}} \oplus 0\underline{1111}10\underline{1}_{\text{bin}}) = HW(0\underline{1111}00\underline{1}_{\text{bin}}) = 5$.

Note that the HD model doesn't represent the entire consumption of a chip but only the data dependent part. Considering a chip as a large set of elementary electrical components, where indeed the bus lines are considered the most consuming elements within a microcontroller, the basic linear model in (6) can be derived to describe the instantaneous power consumption $P_{instant}$ of the device in function of $HD(x_0, x_1)$, where b wraps offsets, time dependent components and noise (i.e., $P_{operation}$, P_{algo_noise}, and P_{elec_noise}, in the context of (2)), while a is a scalar gain between the Hamming distance and $P_{instant}$ [14].

$$P'_{instant} = P'_{data} + b = a * HD(x_0, x_1) + b \tag{6}$$

In essence, the HD model is mainly used for registers and buses. A constraint, to the application of the HD model in the context of a PA attack, is that it relies on the attacker's knowledge of the preceding or the succeeding value of the bus or register to be able to calculate the hypothetical power consumption associated with the transition.

2.3.3.2 Hamming-Weight model

In some microcontrollers, the data bus lines are reset to bit 0 or 1 before writing a value to it [23], in what's referred to as precharging. In the HD model, when the bus lines are precharged with bit 0, power consuming transitions occur for all bits of the new handled value that are set to bit 1. Contrarily, if the bus lines are precharged to bit 1, the meaningful transitions occur for all the bits of the new value that are not bit 1_t. For both cases, the power consumed to handle the new value x_1, will be related to the Hamming weight (i.e., number of bits set to 1) of the processed value. Based on this realization, the Hamming-Weight model can be deduced.

The HW model, described in (7), is a particular case of the HD model, based on the assumption that the data bus lines are precharged with 0 prior to processing x_1.

$$HD(x_0 = 0,\ x_1) = HW(0 \oplus x_1) = HW(x_1) \tag{7}$$

Accordingly, it is reasonable to say that, for the HW, the attacker assumes that the data-dependent component of power consumption is proportional to the number of bits that are set to 1 in the processed data value. In [6], the power consumption of a microcontroller was recorded while it transferred all 256 possible byte values from its internal memory to a register. The distributions exhibited in Figure 5, obtained from this experiment, showed that bytes with the same Hamming weight followed the same power consumption distribution, implying that, for that specific microcontroller, the power associated with storing a byte from the internal memory to the 8-bit register could indeed be modeled by the HW of the processed value, which might suggest that the register is precharged to 0 before having data written to it.

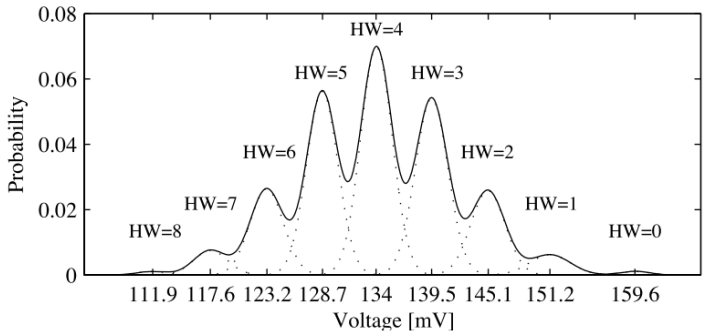

Figure 5 – Distribution of power consumption when the microcontroller transfers different data from the internal memory to a register [6].

Figure 6 depicts the power consumption fluctuation associated with the execution of a MOV instruction for bytes with different Hamming weights.

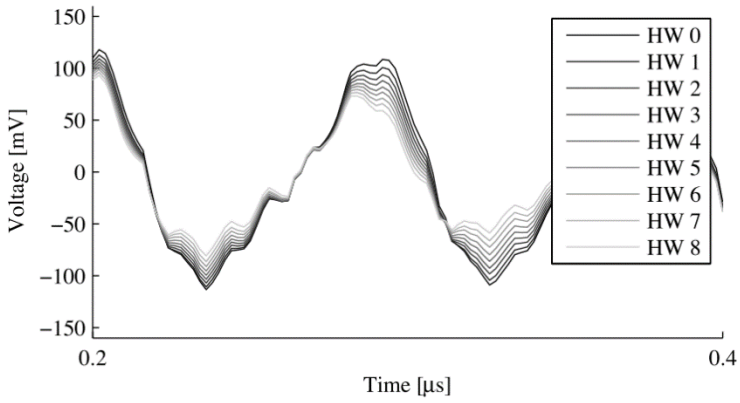

Figure 6 – MOV instruction with different Hamming Weights [6].

In fact, even when the previous value x_0 is unknown or not constant, the Hamming weight of the handled data value x_1, is usually not completely unrelated to the power consumption caused by the processing of this value [6], meaning there's always some information that can be obtained from the HW model and exploited with appropriate statistical analysis [37].

Whenever consecutive data values are unknown (i.e., attacker is only aware of one data value that is transferred over a bus) or information regarding the underlying design of the target is scarce, the HW model is typically attempted, for its simplicity and widely reported usage in the available literature covering this domain, i.e., [3][6][7][8][23].

2.3.3.3 Least Significant Bit model

The LSB leakage model establishes a relation between the least significant bit of a value x_1 and the power consumption associated with its processing. The LSB model takes the bit with the lowest corresponding integer value (i.e., in big-endian ordering, the LSB is the last bit) of the processed value x_1 [13]. For example, considering big-endian ordering, $LSB(x_1=1001011_{bin}) = 1_{bit}$. This model can produce only one of two values, either bit 0 or 1, hence it is referred to as a binary model.

In [11], the distributions of power consumption measurements according to the LSB of the handled value were calculated. It was determined that the two distributions were indeed significantly different, demonstrating that the power consumption was statistically correlated to the LSB of the processed values. In this important article by *Kocher et al.*, the LSB leakage was reportedly associated to "the placement of inverters by logic synthesis tools and other design details".

When compared to the models mentioned earlier, the LSB leakage model falls short in the sense that it relies only on one bit of the processed value, in contrast to the HD and HW that encompasses all bits. Consequently, in the context of a power-based side-channel attack, the LSB model requires a larger set of measurements to succeed. Notably, *Kocher et al.* in [11] states that if any bit other than the least significant bit of the processed value was considered instead, the resulting distributions would still be clearly distinguishable, meaning that the instantaneous power consumption is dependent on all bits and not exclusively on the least significant bit or other individual bit.

2.3.3.4 Other leakage models

Various ways exist to model the data-dependent power leakage of a target device. These are strongly determined by the circuit design and how data is processed within the circuit (i.e., cryptographic implementation details). The purpose of this subsection is to elucidate the reader for alternative leakage models, and to provide the necessary intuition for reasoning about new models for power- or EM-based side-channel attacks.

Section 2.3.1 suggests that, due to the physical characteristics of CMOS gates, switching from bit 0 to 1 consumes more than transitioning from bit 1 to 0, in CMOS devices. As such, it should be possible to model the power associated with these transitions more accurately than the HD model, in which 0→1 and 1→0 bit transitions are assumed to consume the same. Table 6 defines the Switching-Distance model, according to [48], with $\delta = P_{0\to1} - \frac{P_{1\to0}}{P_{0\to1}}$, where $P_{0\to1}$ is the probability of a 0→1 transition. An experiment carried out in [48], on an 8-bit PIC microprocessor, demonstrated that this improved model allowed for more accurate predictions than the HD model.

Table 6 - Switching-Distance model: hypothetical power consumptions

Transition	Power
$0 \to 1$	1
$1 \to 0$	$1 - \delta$
$0 \to 0$	0
$1 \to 1$	0

As mentioned in Section 2.3.2, EM measurements may contain information regarding the current flow within a localized area of the circuit allowing to differentiate a charge from a discharge of the bus depending on the sign of the measured EM emanation. Based on this observation, *Peeters et al.* in [48], proposes a particular case of the Switching-Distance model with $\delta = 2$, to specifically model EM emanations, assuming a leakage of +1 and of -1 for 0→1 and 1→0 bit transitions, respectively. For power leakages, $\delta = 0.17$ was assumed.

Note that the hypothetical power consumption or EM emission values modeled by the Switching-Distance model are spread over a much larger set of discrete values, than the ones produced from the HD and HW models where, for example, the processing of 8-bit data values yields one of nine possible results (i.e., range of Hamming weight values). In the context of a PA attack and from an information theoretic point of view, having a larger set of discrete values should ease the discrimination of the processed values associated with the observed power consumption through statistical analysis, resulting in enhanced efficacy. However, the authors acknowledge that the statistical tools used in DPA (Differential Power Analysis) or CPA (

Correlation Power Analysis) do not allow to take advantage of this additional leakage, in the sense that the number of measurements required to successfully recover the key using the improved model is not reduced. Nonetheless, other statistical tools might make it possible to achieve better results.

Leakage models can be defined to model data-dependent leakage associated with specific implementation details of an algorithm executed by target device. One such example is the so-called Zero-Value model defined in [6] to target an implementation of the AES's S-boxes based on composite field arithmetic. Such implementations have the property that if the S-box input is zero, they tend to consume significantly less power than that of any other input values. This is due to the fact that in case of zero, all multiplication in the S-box are multiplications by zero which usually require significantly less power than others. This observation results in the binary power model described in (8), that can be further leveraged in a DPA attack, on the inputs of the S-box, to recover a full AES's key [6].

$$ZV(x_1) = \begin{cases} 0, & \text{for } x_1 = 0 \\ 1, & \text{for } x_1 \neq 0 \end{cases} \tag{8}$$

2.3.4 Measuring leakage sources

It is the variation of the power consumed by the target device, during the processing of sensitive cryptographic data, that is exploited in a PA attack. The power consumption of the device can be measured directly by inserting a power measurement circuit between the power supply and the device, or indirectly by an EM probe.

Measurement setups for PA usually consist of several components that interact with each other [6]: device under attack, clock generator, power supply, power measurement circuit or EM probe, oscilloscope, and a PC. The device under attack typically provides an interface to communicate with the PC (e.g., UART, USB) that can be used to trigger the execution on the cryptographic algorithm. Some devices are provided with an internal clock signal generated, for example, by an internal quartz crystal oscillator, while others, such as smart cards, don't and therefore need to be supplied with an external clock signal from an external clock generator. In cases where the internal clock signal is poor (i.e., jittering), resulting in lower quality measurements, it should be replaced with a more stable external clock source. The stability of the power supply is also important for the quality of the measurements, since fluctuations in the power provided to the device are perceived as noise, in the measurements, which can

compromise the success of the attacks. Therefore, a DC (Direct Current) supply is preferred over AC (Alternate Current) given the fact that AC voltage alternates in direction and magnitude, which can lead to slight fluctuations in voltage and current, while DC voltage is generally more stable since it flows in one direction and does not alternate, thus DC power supplies are typically used to power electronic devices as many of their electronic components (e.g., transistors) require stable, constant voltage and current, for proper functioning [45].

The interaction between the mentioned components can be described as follows [6]: to make it operational, the device is supplied with the necessary power and clock signal. Then, the oscilloscope is configured and armed by the PC under control. The PC can then send commands to the target device to trigger the execution of the cryptographic algorithm. During the execution of the algorithm, the oscilloscope records the power consumption, either through a power measurement circuit or an EM probe. Finally, the captured trace is transferred to the PC. This process is repeated as often as required to obtain the necessary number of traces. The number of traces can range from a dozen to millions of traces, and it highly depends on the quality of measurement [36].

Typically, to measure the current draw on a circuit, a small resistor is inserted in series with the power or the ground input [23]. An oscilloscope is used to sample the voltage drop across the terminals of the resistor and so collect the power trace for later analysis and attack. The voltage drop is an indirect measure of the actual power consumption. In fact, in side-channel attacks, the term "power consumption" usually refers to an indirect measure of that consumption, such as the voltage drop across a shunt resistor. The actual value of power consumed is often irrelevant since the relative variations of measured power encode the information for the attack [36]. Current is described by Ohm's law in (9), where I is current, V is voltage, and R is resistance.

$$I = \frac{V}{R} \tag{9}$$

According to Ohm's law, the current I flowing through a conductor between two points is directly proportional to the voltage V, such that a higher resistance value results in a larger voltage difference. Thus, in order to provide sufficient voltage to power the chip, a shunt resistor must be employed to ensure an appropriate voltage drop across the resistor. To reduce relative noise caused by electronic components that consume power unrelated to processing, the resistor should be inserted as close as possible to the chip. When measuring power, one should make sure that the device under attack and the measurement device (i.e., oscilloscope), don't share common ground, to avoid ground loops. Capacitors in the circuit might interfere with the observable power, as they stabilize the current being fed to the microcontroller [23]. They work by charging up, holding the charge, and releasing it again when the voltage drops. To acquire a trace that describes the leaking power consumption of the chip, some capacitors might need to be removed from the circuit, to reduce the filtering effect from them. Nevertheless, if the capacitance of the circuit is drastically reduced, the microcontroller might cease to boot or maintain a stable execution, due to the lack of power stability. A trace refers to the measurement of the instantaneous power consumption taken over a period of interest (e.g., a full cryptographic operation). Frequently, when collecting the power trace of an operation, multiple times, the resulting traces may deviate from each other because of normally distributed noise induced by the measuring device itself. Fortunately, given the nature of a normal distribution, the effect of this noise can be slightly reduced by averaging out the gathered traces [6]. Yet, the process of averaging may lead to the partial loss of pertinent, exploitable information, which may or may not influence the outcome of the attack.

Note, however, that in practice, depending on the target's circuit design, the power measurement setup may not be as straightforward as the one mentioned. For cryptographic processors on embedded chips, installing a power tap may be difficult for various reasons [47]:

a) The power line may run though one of the inner layers of the printed circuit board;

b) The power pin that feeds the components of interest, related to the execution of the cryptographic implementation (i.e., cryptographic processor, data buses, or others), may be difficult to identify or select out of the many power pins;

c) As already mentioned, the insertion of a measurement resistor or current probe may affect the proper functioning of the microcontroller or even reduce measurement quality.

Moreover, the EM side-channel is considered the most viable avenue for attacking cryptographic devices where the power side-channel is unavailable [45]. The approximation of Bio-Savart's law, in (3), is valid when the field is measured close to the source, which called near-field [37]. On the other hand, Maxwell's equations state that the varying current generates EM radiation that propagates to the surrounding space, which is referred to as the far-field. The far-field is beyond the focus of this thesis, yet attacks in practice have been reported in the literature against microcontroller-based devices using such measurement technique, that involves the use of large size antenna with considerable gain.

The near-field EM side-channel attack has the advantage of not requiring the insertion of a measurement circuit into the power path. Instead, an EM measurement probe is placed on top of the area of interest of the circuit, while the device is operating, without the need for tampering with it. Depending on the size and positioning of the probe, the EM field is proportional to the power consumption of a part or the entire device under attack [6]. The near-field EM measurement setup can be configured to focus on the area of the chip that radiates the most leakage. Instead of capturing the total power consumption, focusing the EM field capture on certain areas avoids the influence of power consumed in parts of the device that do not leak useful information and are "noise" for our intended attack [37]. This is extremely relevant, for example, in the context of attacking complex SoCs, as it increases the chances of obtaining valuable information in such a compact layout, by capturing localized activity. One of the main challenges recognized by the authors of [36], when targeting the hardware cryptoprocessor embedded into a mobile phone SoC was the fact that the power domain of this low-power peripheral was shared by other components which made it impossible to detect the power leakage related to its execution. In consequence, the near-field EM side-channel was explored, by performing the measurement directly over the cryptoprocessor, greatly simplifying the signal acquisition process and leading to a successful attack. The quality and strength of the EM signals observed is commonly weak, therefore the usage of preamplifiers that are designed for the amplification of measuring signals is advised.

In essence, it's not practical to generalize and define a standard best way of measuring the variation of power, whether it be by inserting a power tap into the circuit or by using an EM probe. Such a decision is subject to a variety of factors and considerations, which were briefly covered in this section with the intent of providing the reader with sufficient background to critically reason about this matter.

2.4 A Brief Introduction to Advanced Encryption Standard

The Advanced Encryption Standard (AES) is the most widely used symmetric-key algorithm today [1][24]. The AES block cipher is mandatory in several industry standards, it is used in many commercial protocols such as TLS and SSH, and it can be highly optimized for embedded systems [1]. It is based on a substitution-permutation network, which is a direct implementation of the confusion-diffusion paradigm, being very efficient in both software and hardware implementations. AES divides data into blocks of 16 bytes and iteratively applies a series of operations that are linked to each other, to eventually obtain the 16-byte output block, contrary to predecessor block cipher designs based on Feistel networks, such as DES, where the data block is divided into two equal-sized pieces and for each round a round-function is run on half of the data to be encrypted, and its output XORed with the other half of the data, which is a completely different paradigm from the one of substitution-permutation networks.

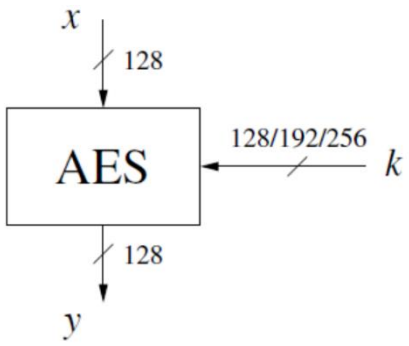

Figure 7 – Advanced Encryption Standard input/output parameters [1].

The AES cryptographic algorithm operates on fixed sized blocks of 128 bits and supports different key sizes of 128, 192, or 256 bits, as broadly illustrated in Figure 7. AES implements a key schedule algorithm to compute multiple 128-bit round keys, also known as subkeys (K_0, K_1, …, K_{nr}), from the master key (K). Table 7 describes how the number of rounds (n_r) is dependent on the length of the master key. The 128-bit data block and each round key is represented as a 2-D array of bytes consisting of four rows and four columns. Initially, in the encryption algorithm, the data array, also referred to as the state, describes the plaintext, and is manipulated by the processing steps of each round, to ultimately derive the ciphertext block. The process of decrypting is similar to that of encrypting, in the sense that, to transform the ciphertext back to the original plaintext, the inverse set of operations and round keys are applied, and the same encryption key is used [1].

Table 7 - Key lengths and number of rounds for AES

key length	# rounds = n_r
128 bits	10
192 bits	12
256 bits	14

Brute-force attacks, which consist of a trial-and-error exhaustive search over all possible combinations for a certain key size, with the goal of discovering the correct key, are always a possibility against cryptographic algorithms. For this reason, AES implements key sizes of 128, 192, and 256 bits, meaning that for AES-128, the key space is 2^{128} which amounts to roughly 3.4×10^{38} possible combinations, making it unfeasible to crack with today's limited computing

capabilities of modern computers [24]. So far, the only attacks on AES, feasible with current technology, are side-channel attacks. Side-channel attacks do not attack the cipher as a black box, and thus are not related to cipher security as defined in the classical context, as they attack implementations of the cipher on hardware or software systems that inadvertently leak data.

For a detailed description of the Advanced Encryption Standard the reader is referred to [27] or [28].

2.4.1 High-level description of the algorithm

The AES cryptographic algorithm is made of five operations. In the Key Scheduling operation, (n_r+1) 128-bit round keys are generated from the secret key K, either before starting the encryption/decryption operations, or in parallel with each round. Each round is composed of the AddRoundKey, SubBytes, ShiftRows, and MixColumns operations [28]. In the AddRoundKey operation, a round key is XORed with the current state bytes. Then, in SubBytes, each state byte is replaced with a corresponding byte from a fixed 8-bit lookup substitution table called the S-box, with the goal of cancelling the linear relation between the input and the output of the AddRoundKey, associated with the XOR operation, introducing so-called *confusion*. The ShiftRows operation cyclically shifts the rows of the state array, increasing *diffusion* by ensuring that a change in one byte of a row affects multiple columns in the subsequent round. In MixColumns, the columns of the state array are mixed using a linear transformation (i.e., matrix multiplication operation) so that each byte in a column of the state array becomes dependent on all column bytes, hence a change in one plaintext byte spreads to all ciphertext bytes. This operation along with ShiftRows, increases the overall confusion and non-linearity of the encryption process.

Confusion and diffusion are two fundamental properties of block ciphers like AES. Confusion refers to the process of making the relationship between the plaintext and the ciphertext as complex and non-linear as possible, so that an attacker cannot deduce any information about the plaintext or key. Diffusion refers to the process of spreading the effect of each input bit throughout the output, so that a small change in the input affects many bits in the output. This combination of confusion and diffusion ensures that even small changes in the input or the key result in a completely different output, making it extremely difficult for an attacker to deduce any information about the plaintext or the key, with cryptanalysis [1].

Figure 8 provides a high-level description of the AES encryption algorithm. Observe that the MixColumns operation isn't applied in the last round of the encryption algorithm. Be aware that, the first subkey K_0, that is XORed with the plaintext in the initial AddRoundKey operation, before the first round, corresponds to the first 128 bits of the actual secret key K, from which all other round keys are generated, and because (n_r+1) AddRoundKey operations are performed in total, (n_r+1) subkeys are required instead of just n_r. For the purpose of attacking AES using side-channel information, it's relevant to notice that, in the AddRoundKey, SubBytes, and ShiftRows operations, each byte of the state is manipulated individually and independently from the other bytes. Only in MixColumns, the output of each byte (8 bits) depends on four bytes (32 bits).

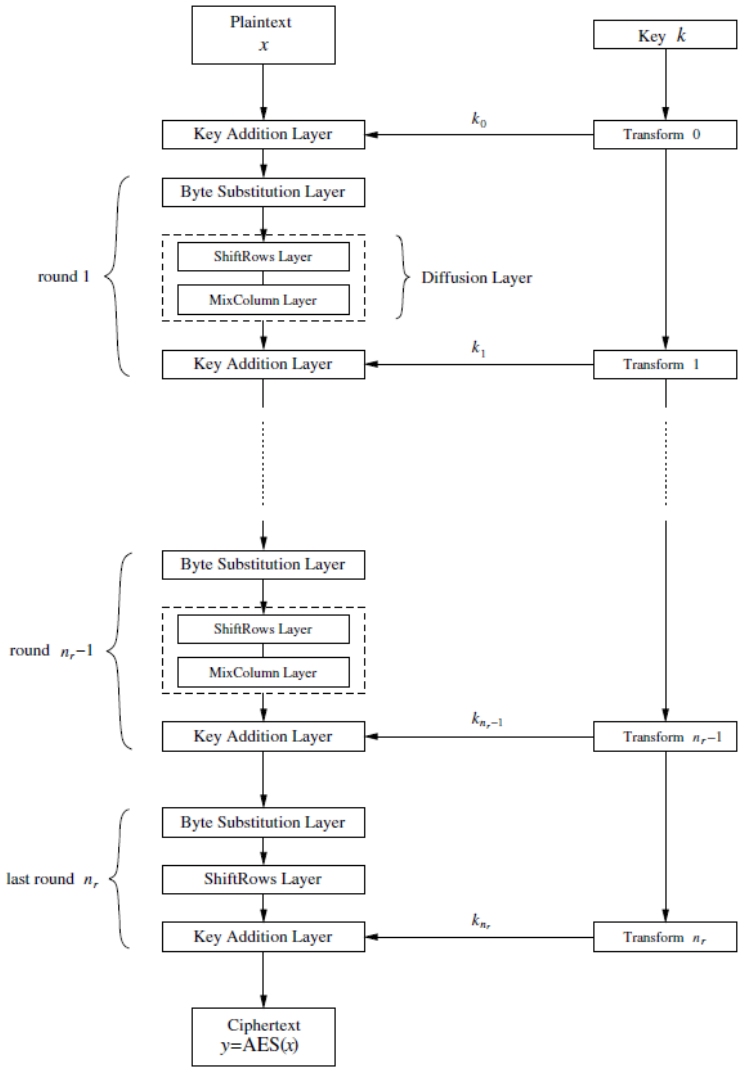

Figure 8 – Advanced Encryption Standard encryption block diagram [1].

In the following subsections each AES operation is briefly detailed.

2.4.1.1 Key Expansion layer

AES uses a key schedule algorithm to derive different 128-bit subkeys for each of the rounds, from the secret key K. However, the round key K_0, that is XORed with the 128-bit plaintext block before the first round, describes exactly the first 128 bits of the secret key K. In AES-192, the remaining 64 bits of the 192-bit key K are the first 64 bits of K_1 fed to the AddRoundKey operation of round 1, while for AES-256, K_1 corresponds to the full second half of the 256-bit key K [1]. Therefore, bear in mind that the output of the AES-128's first SubBytes operation corresponds to SBox(plaintext $\oplus$ K). Being aware and understanding this type of implementation details is particularly relevant when conducting key extraction attacks based on side-channel information.

The Key Expansion layer, also referred to as the Key Scheduling operation, is represented in Figure 8 by the "Transform *i*" blocks, at the right. In the high-level description of Figure 8, this operation is represented in parallel to each round, which can be implemented

in practice when the device has enough resources and supports concurrency or parallelism, to speed up the execution time of the algorithm. Otherwise, all (n_r+1) round keys can be computed before the encryption or decryption is executed, and then provided as input to each AddRoundKey operation.

The AES key schedule is word-oriented, where 1 word = 32 bits. Subkeys are stored in a key expansion array *W* that consists of words. The key expansion algorithm is different, but fairly similar, for each of the three different AES key sizes. Since the exploitation of the leakage information from this operation was not pursued in the experimental work of this dissertation, details on the key scheduling algorithms are not provided, but the interested reader is referred to section 4.4.4 of [1] and section 5.2 of [28]. Nonetheless, note that this operation manipulates the secret key bytes and therefore it can be a valid source of exploitable leakage.

2.4.1.2 Key Addition layer

In the so-called AddRoundKey operation of AES, the current state data is XORed with the 128-bit derived subkey. As depicted in Equation (10) and Figure 9, the state byte $a_{i,j}$ is XORed with the round key byte $k_{i,j}$ to obtain the resulting byte $b_{i,j}$.

$$a_{i,j} \oplus k_{i,j} = b_{i,j} \tag{10}$$

$$\begin{bmatrix} a_{0,0} & a_{0,1} & a_{0,2} & a_{0,3} \\ a_{1,0} & a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,0} & a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,0} & a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \oplus \begin{bmatrix} k_{0,0} & k_{0,1} & k_{0,2} & k_{0,3} \\ k_{1,0} & k_{1,1} & k_{1,2} & k_{1,3} \\ k_{2,0} & k_{2,1} & k_{2,2} & k_{2,3} \\ k_{3,0} & k_{3,1} & k_{3,2} & k_{3,3} \end{bmatrix} = \begin{bmatrix} b_{0,0} & b_{0,1} & b_{0,2} & b_{0,3} \\ b_{1,0} & b_{1,1} & b_{1,2} & b_{1,3} \\ b_{2,0} & b_{2,1} & b_{2,2} & b_{2,3} \\ b_{3,0} & b_{3,1} & b_{3,2} & b_{3,3} \end{bmatrix}$$

Figure 9 – In AddRoundKey, the round key is added to the state with an exclusive-or operation [6].

In Figure 8, (n_r+1) AddRoundKey operations are shown: the state bytes are first combined with the round key bytes before the first round, and then the operation is applied in every round.

2.4.1.3 Byte Substitution layer (S-box)

The byte substitution operation, which is applied to each byte $a_{i,j}$ of the current state separately, is referred to as the SubBytes. In the SubBytes operation, $a_{i,j}$ is substituted with another byte $b_{i,j}$, according to a substitution table (S-box), with the aim of minimizing the correlation between the input and the output of the function, and protect AES against linear and differential cryptanalysis [6].

$$\text{SBox}(a_{i,j}) = b_{i,j} \tag{11}$$

The S-Box is the only non-linear step in the round transformation. It is usually realized as a lookup table with fixed entries, as described in Figure 10, because the function that defines the S-box requires computationally expensive operations [6].

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

Figure 10 – AES S-box: Substitution values in hexadecimal notation for input byte (xy) [1].

Take, for instance, $a_{i,j} = (xy)_{hex} = (C2)_{hex}$, where $x = C_{hex}$ and $y = 2_{hex}$. The output of the S-box for $a_{i,j} = (C2)_{hex}$ is $b_{i,j} = SBox((C2)_{hex}) = (25)_{hex}$. This mapping is represented in Figure 11.

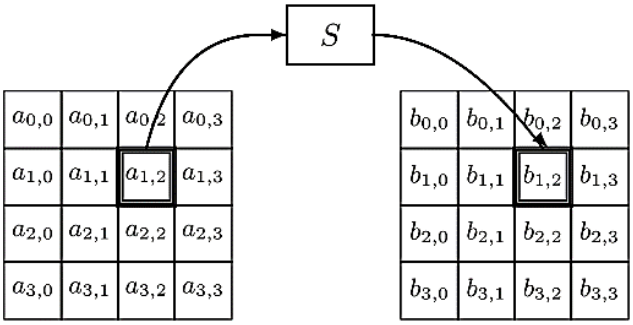

Figure 11 – SubBytes works on the individual bytes of the state [6].

The S-box has a strong algebraic structure, and it's implementation can be achieved on-the-fly by mapping polynomials over the Galois field $GF(2^8)$ to their multiplicative inverse using an affine transformation [27], as pictured in Figure 12.

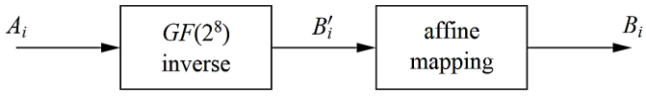

Figure 12 - The two operations that constitute the S-box

The SubBytes operation is applied in every round and is represented, in Figure 8, by the "Byte Substitution layer" block, always after the "Key Addition layer" block.

2.4.1.4 ShiftRows layer

The ShiftRows transformation cyclically shifts the 2nd row of the state matrix by 3 bytes to the right, the 3rd row by 2 bytes to the right, and the 4th row by 1 byte to the right. Note that the 1st row is not changed by this transformation. The shift positions have been chosen according to two criteria that are both related to the diffusion. Diffusion means that the individual bytes of the state get dispersed all over the state. For optimal diffusion, the offsets of ShiftRows need to be chosen differently from each other [6].

In Figure 8, the ShiftRows operation is represented by each "ShiftRows layer" block, in every round, after the "Byte Substitution layer" blocks. In practice, the ShiftRows operation can be executed before the SubBytes operation because in SubBytes each byte is mapped to another byte value, independently, the ShiftRows shifts each position, independently, thus the output from doing SubBytes before ShiftRows or ShiftRows before SubBytes, is the same.

2.4.1.5 MixColumns layer

The MixColumns step is a linear mixing transformation which operates on the columns of the state matrix, combining the four bytes in each column. The four bytes of each column of the state are combined using an invertible linear transformation. It takes four bytes as input and outputs four bytes where each input byte affects all four output bytes. The combination of the ShiftRows and MixColumns layer makes it possible that after only three rounds every byte of the state matrix depends on all 16 plaintext bytes [1]. The mixing of the elements is achieved by taking each column of the state and multiplying it with the fixed polynomial in (12), which can be represented by the matrix multiplication in (13), where the 4-byte column *j* is represented as a vector $[a_{0,j}, a_{1,j}, a_{2,j}, a_{3,j}]$, and multiplied by a fixed 4x4 matrix, to obtain the result $[b_{0,j}, b_{1,j}, b_{2,j}, b_{3,j}]$. The choice of this particular multiplication polynomial has been motivated by the so-called wide-trail design strategy, which provides high resistance against linear and differential cryptanalysis [6].

$$c(x) = 03x^3 + 01x^2 + 01x + 02 \; mod \; x^4 + 1 \tag{12}$$

$$\begin{bmatrix} b_{0,j} \\ b_{1,j} \\ b_{2,j} \\ b_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} a_{0,j} \\ a_{1,j} \\ a_{2,j} \\ a_{3,j} \end{bmatrix} \tag{13}$$

2.4.2 Software implementations

There are many ways of efficiently implementing software-based AES for embedded devices. Determining which implementation is the most suitable, highly depends on the device's resource constrains, and on the use-case requirements that often result in the trade-off between memory, execution time, and power consumption [1]. These constraints are taken into account when evaluating certain implementation details, for instance, when memory capacity is scarce, having round keys computed on-the-fly can be advantageous in the sense that only 16 bytes of memory are required to keep the current round key accessible in memory, instead of the $16*(n_r+1) = 176$ bytes, if they were to be pre-computed in AES-128. Also, the usage of a lookup table for the S-box, stored at a fixed memory location and taking 256 bytes, greatly reduces the execution time of an encryption run, compared to the alternative of computing the

necessary S-Box outputs during runtime using arithmetic operations. Even the way memory is managed by the program, impacts execution time and power consumption [6].

In a naïve implementation, all time-critical functions (i.e., SubBytes, ShiftRows, MixColumns) operate on individual bytes [1]. While processing one byte per instruction is well-suited for 8-bit processors, it is deemed inefficient for modern 32-bit or 64-bit processors. So, the Rijndael designers proposed a method for a faster software implementation, which consisted of merging all round functions (except the rather trivial key addition) into one table lookup. This requires four tables, referred to as T-Tables or T-boxes, each of which contains 256 of 32-bit wide entries. Four table accesses yield 32 output bits of one round, meaning one round can be computed with 16 table lookups [1]. Even though this optimized approach results in considerably faster execution times, it comes at the significant cost of requiring increased memory capacity, which might not always be supported or tolerable with every device. This implementation is used, for example, by Golang's crypto library, mBedTLS, OpenSSL, and its original proposal can be found in Section 5.2 of [40].

Software-based implementations are commonly executed on the general-purpose CPU of the microcontroller. They are designed and coded in programming languages such as C, C++, Java, or Assembly [30].

2.4.3 Hardware implementations

AES is not only suitable for software implementations. It can be more efficiently implemented in hardware. In fact, many publications are available with greatly optimized hardware implementations of AES for all types of different applications: [30], [29]. These range from ultra-low-power implementations for RFID devices, to high-performance implementations for Internet servers. Contrary to software-based implementations that are executed on the general-purpose CPU of the microcontroller, hardware-based implementations are implemented and executed on a co-processor referred to as "crypto accelerator" that consists of a piece of hardware specifically made to execute cryptographic intensive operations, either in the form of a FPGA or ASIC. Hardware implementations are designed and coded in hardware description languages, such as VHDL and Verilog HDL.

FPGAs are integrated circuits that can be configured and reprogrammed to perform a wide range of digital logic functions. They contain thousands of configurable logic blocks and a set of programmable interconnects that allow the designer to connect blocks and configure them to perform everything from simple logic gates to complex functions [6]. On the other hand, ASICs are custom-built integrated circuits designed to perform a particular task or application, which cannot be reconfigured after manufacturing. The primary difference between FPGAs and ASICs in terms of performance is the slower speed of FPGAs caused by the delays introduced by the circuitry required for reconfiguration. As a result of this speed penalty, any digital circuit implemented in an FPGA is typically slower than the same circuit in an ASIC, assuming that both ICs are fabricated using the same semiconductor technology (in particular, using the same transistor size) [30]. Choosing between implementing AES on a FPGA or an ASIC will depend on factors such as the required performance, power consumption, and cost of the target application. FPGA-based implementations can provide flexibility and ease of development, while ASIC-based implementations can provide higher performance and lower power consumption, but at a higher cost due to the need for custom design and manufacturing. FPGAs do not require the physical design (layout), fabrication, and testing for physical defects. Both FPGAs and ASICs can make full use of parallel processing and pipelining, and operate on arbitrary size words, while for general purpose microprocessors, parallel processing and pipelining is limited by the number and internal structure of the processor functional units and

by the instruction level parallelism, and all functional units operate on fixed-sized arguments only [30].

The S-boxes needed for the SubBytes operation are the most critical part of the hardware implementations of AES. This is due to the fact that the S-boxes require the most resources compared to other operations of AES. In practice, there are essentially three different ways to implement AES S-boxes. These can be implemented using read-only memory (ROM), they can be synthesized as lookup tables, or they can be implemented based on composite field arithmetic. Actually, the amount of memory required to implement SubBytes can be reduced to zero by utilizing the internal logic structure of inversion in GF(2^8), as in Figure 14. This approach makes particular sense for ASIC implementations, in which memory is typically costly in terms of the circuit area. In FPGAs, memory blocks are always present independently of whether they are used or not. In [41], two hardware-based implementations of AES, for FPGA hardware, are proposed, both based on lookup tables, one employing S-boxes and the other, T-Tables.

Figure 13 shows the block diagram of the 32-bit encryption core of the ASIC AES hardware implementation detailed in [6]. Note that this example serves only as a reference of interest to provide an insight on how these implementations can be designed and thought about. The actual hardware implementation targeted in the Experimental Evaluation section of this work, is completely closed-source so no information regarding its design or implementation details is available, therefore the attack will solely be based on assumptions concerning the underlying algorithm and hardware design that are corroborated using leakage assessment techniques detailed in Section 3.2. This aspect introduces an additional layer of complexity and difficulty to the attack. Hence, it is reasonable to acknowledge that being aware of how hardware implementations are typically designed and accomplished in practice, can facilitate the process of making informed assumptions and predictions regarding this matter.

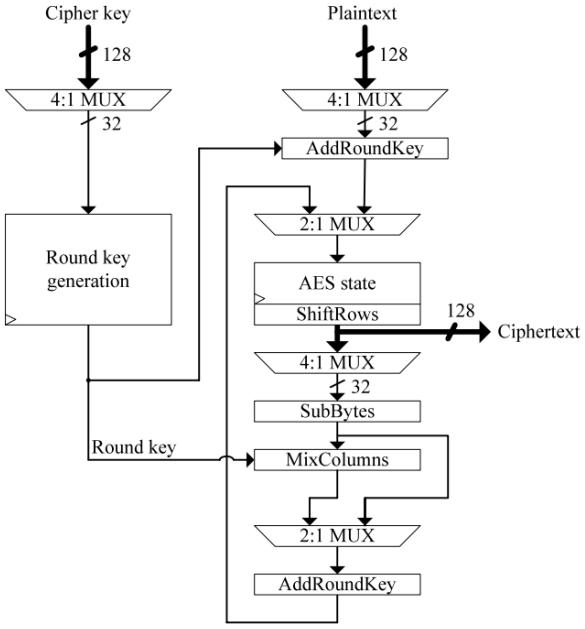

Figure 13 – Block diagram of the data path on ASIC-based AES implementation, from [6].

An overview of the hardware-based AES implementation, described in [6] and partially represented in Figure 13, is provided: Initially, the key and the plaintext are loaded into the

encryption core of Figure 13. This is done in chunks of 32 bits, so it takes four clock cycles to load the key into the module labeled “round key generation”, and another four clock cycles to load the plaintext into the register labeled “AES state”. During the loading of the plaintext, the initial AddRoundKey operation is performed. After loading the key and the plaintext into the encryption core, nine normal encryption rounds are performed. In each round, essentially two things are done. It is necessary to perform the key expansion and it is necessary to update the AES state. In most encryption cores, these two operations are done in parallel. However, in this specific example, the key expansion is performed before updating the AES state, during the ShiftRows operation. The ShiftRows is done during the first clock cycle of the key expansion, and in total the key expansion takes four clock cycles. After performing the key expansion and the ShiftRows operation, the AES state is updated. In this case, that is done column by column. This means that in each clock cycle, the SubBytes, the MixColumns, and the AddRoundKey operation are performed for a 32-bit column of the AES state. Notice that in order to perform the SubBytes operation for 32 bits, four parallel S-boxes are needed. It takes four clock cycles to update the AES state in each round, therefore, each round of this AES implementation takes eight clock cycles in total. In the last round of AES, the MixColumns operation is bypassed using the multiplexer before the last AddRoundKey operation.

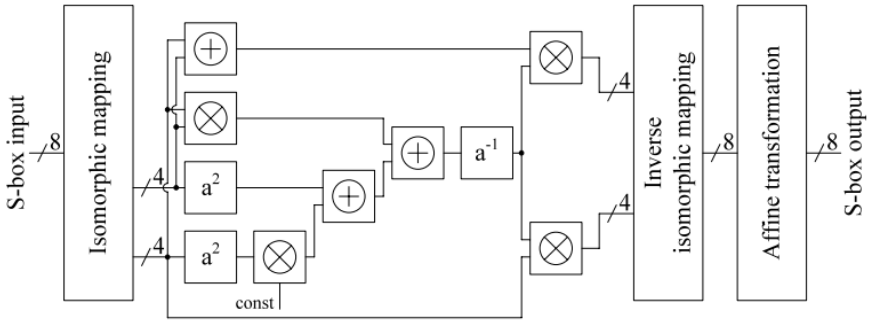

Figure 14 – Block diagram of an AES S-box, based on composite field arithmetic, according to [6].

Hardware implementations of AES can vary a lot from each other, depending on system requirements and objectives. In the previous example, the knowledge of the fact that the AES state is stored/updated to the register represented by block “AES State”, in Figure 13, after the AddRoundKey operation, provides good information that either the output of the first AddRoundKey, or the input of the last SubBytes operation (when the ciphertext is known) of the encryption algorithm should be targeted in a Power Analysis attack, because these attacks exploit the dependency between the power consumed by the device and the data it is processing, and when the data is stored to or loaded from a register, charge is applied in the data bus, and this amount of charge that can be measured in the power consumed by the device, depends on the values of the data that was processed. Due to the non-linear property of the SubBytes operation, the last option (i.e., input of last SubBytes operation) would be preferred. The purpose of mentioning this case is to demonstrate the relevance of possessing prior knowledge regarding the algorithm and hardware design under attack. Nonetheless, having the intuition to critically reason about possible implementation details is essential to successfully conduct side-channel attacks against cryptographic devices, as further explained in Section 3.1.2.

In addition, the interested reader is referred to another ASIC-based AES implementation, in [42], where a custom compact 8-bit data-path architecture core for a single-chip VLSI AES crypto-hardware acceleration is proposed.

2.4.4 Block cipher mode of operation

AES is a block cipher that operates on fixed-size blocks of data. A block cipher mode of operation is a way of using a block cipher to encrypt data that is larger than the block size, describing how to repeatedly apply a cipher's single-block operation to securely transform various blocks of data. These modes can be divided into two major categories:

1. Non-feedback modes, such as ECB (electronic codebook) mode and CTR (counter) mode.
2. Feedback modes, such as CBC (cipher block chaining) mode, CFB (cipher feedback) mode, and output feedback mode (OFB).

In the non-feedback modes, encryption of each subsequent block of data can be performed independently from processing other blocks. In particular, all blocks can be encrypted in parallel. In the feedback modes, it is not possible to start encrypting the next block of data until encryption of the previous block is completed. As a result, all blocks must be encrypted sequentially, with no capability for parallel processing. The limitation imposed by the feedback modes does not concern decryption, which can be performed on several blocks of ciphertext in parallel for both feedback and non-feedback operating modes [30].

Before employing any block cipher mode of operation in a production system, it is strongly recommended to have a thorough understanding of NIST's "*Recommendation for Block Cipher Modes of Operation*" (e.g., SP 800-38A, in [45]), for the chosen method.

The diagrams presented in the following subsections to describe each covered mode of operation are highly elucidating.

2.4.4.1 ECB

ECB is the most straightforward way of encrypting data, but also the least recommended. In ECB mode, the data is divided into blocks, and each block is encrypted (or decrypted) separately and independently. Figures 13 and 14 describe ECB mode of operation. For AES, the "block cipher encryption" and "block cipher decryption" boxes correspond to AES encryption and AES decryption, respectively. With regards to AES, these can be matched with Figure 7.

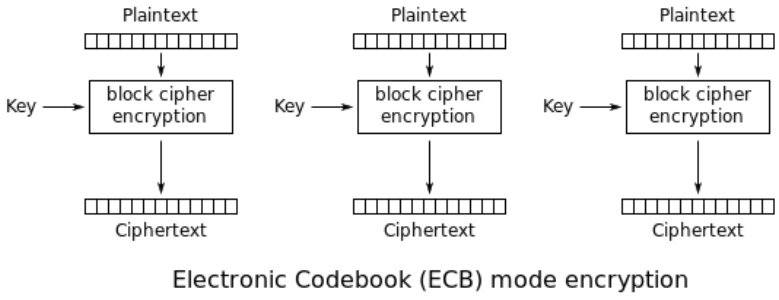

Figure 15 - ECB mode encryption [26].

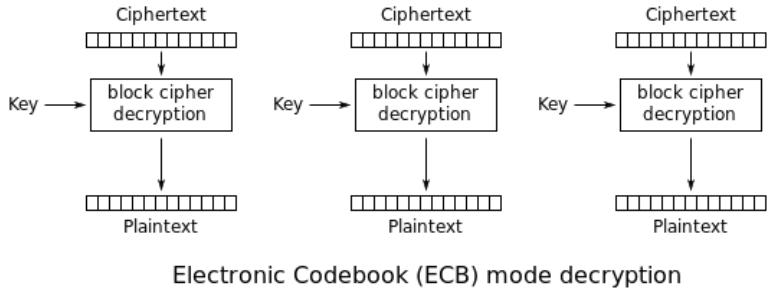

Figure 16 - ECB mode decryption [26].

The use of ECB mode is not recommended due to the fact that it is not semantically secure, meaning that ECB-encrypted ciphertext can leak information about the plaintext, because encrypting the same block of 128 bits always yields the same block of ciphertext [26].

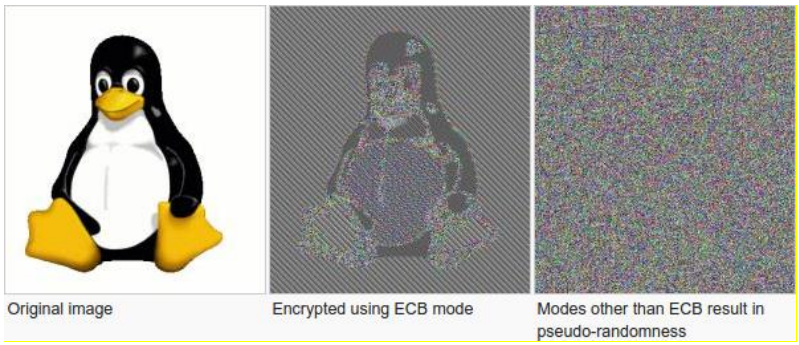

Figure 17 - ECB mode weakness [26].

Figure 17 is an illustration of the ECB mode weakness where the original image is encrypted using both ECB mode and a semantically secure cipher mode such as CBC, CTR, CFB, or OFB. It demonstrates how encrypting the original image, that contains repetitive areas, using ECB mode, results in repetitive patterns in the encrypted output which leaks information about the original image.

2.4.4.2 CTR

In CTR mode, a unique counter value is encrypted for each block of plaintext, and the resulting ciphertext is XORed with the plaintext to produce the final ciphertext. The counter value is typically a block number, which is incremented for each block of plaintext. The main advantage of CTR mode is that it allows for parallel encryption and decryption, making it faster than other modes of operation, in compatible hardware [26].

The input to the block cipher is a counter which assumes a different value every time the block cipher computes a new key stream block. Figures 16 and 17 show this principle. Note that “block cipher decryption” is not used, instead “block cipher encryption” is applied for both encryption and decryption. The desired effect is to use the “block cipher encryption” algorithm (i.e., AES-128 encryption) to produce, from the Nonce and Counter values, a keystream which is somehow similar to a one-time pad, to be XORed with the plaintext (or ciphertext) block. Similarly to OFB and CFB modes, CTR uses the block cipher as a stream cipher by computing the keystream in a block wise fashion [1]. So, a device that only supports the encryption

algorithm of AES, can encrypt and decrypt data in CTR mode. In this diagram, the Nonce is identical to the initialization vector (IV) of other modes of operation.

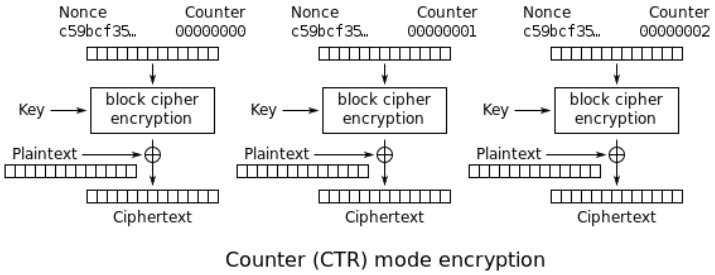

Figure 18 - CTR mode encryption [26].

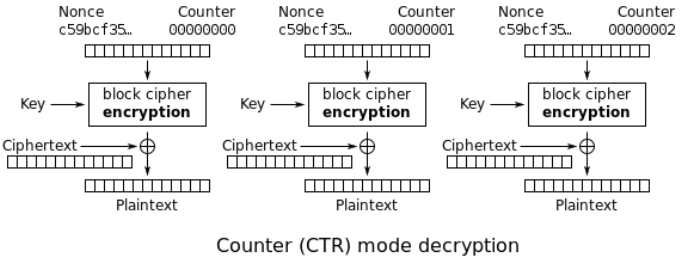

Figure 19 - CTR mode decryption [26].

2.4.4.3 CBC

There are two main ideas behind the Cipher Block Chaining mode. First, the encryption of all blocks is "chained together" such that the current ciphertext block depends not only on the current plaintext block but on all previous ciphertext blocks as well. Second, the encryption is randomized by using an initialization vector (IV) [1]. The idea is to XOR the previously computed ciphertext block, C_{i-1}, with the subsequent block of plaintext, P_i, that is fed to the encryption algorithm, to create the earlier mentioned dependency.

Figures 18 and 19 describe the encryption and decryption of data, respectively, using CBC. Due to the fact that the ciphertext of the previous block is required in the computation of the next block, encryption cannot be parallelized. For the decryption of the CBC encrypted data, the ciphertext of the previous block is needed for the decryption of the current block, but in this case, as the ciphertext is already available, therefore it can be parallelized. Note that, a one-bit corruption of a ciphertext block causes complete corruption of the corresponding plaintext block, and it inverts the corresponding bit of the following block of plaintext, with the rest of the blocks remaining intact [26]. Also, decrypting with an incorrect IV causes the first block of plaintext to be corrupt, while the subsequent blocks remain correct.

In the context of AES in CBC mode, the "block cipher encryption" in Figure 20 and "block cipher decryption" in Figure 21, would correspond to the AES encryption and decryption algorithms, respectively.

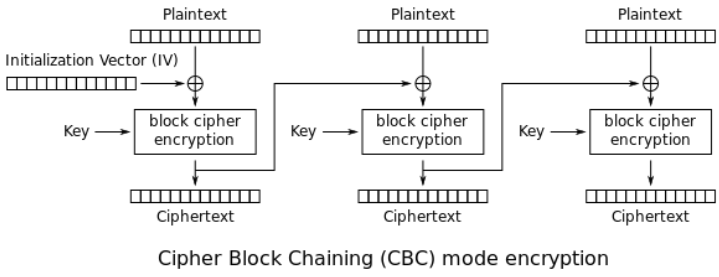

Figure 20 - CBC mode encryption [26].

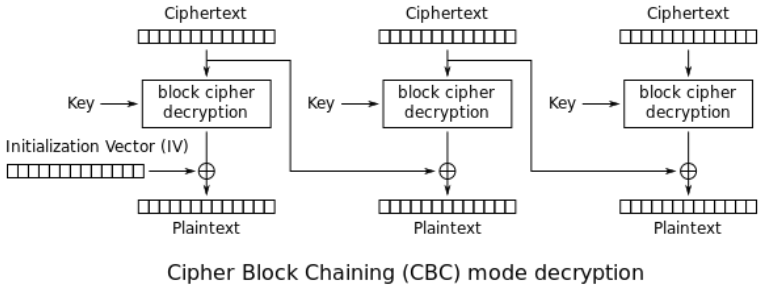

Figure 21 - CBC mode decryption [26].

2.4.4.4 CFB

Similarly to CTR and OFB, the CFB mode uses a block cipher as a building block for a stream cipher, therefore only the encryption algorithm is needed for both encryption and decryption of data using these modes of operation. The first key stream is generated by encrypting an IV using the block cipher encryption algorithm (e.g., AES-128 encryption). Then, this block is XORed with the first plaintext block to produce the first ciphertext block output. For the computation of the following ciphertext blocks, the key stream is generated by encrypting the past ciphertext block. To revert the encryption (i.e., decryption process) of the data, the same key stream that was computed in the encryption process, must be generated for each block and XORed to the ciphertext, to retrieve the plaintext. As in CBC mode, only decryption can be parallelized. Both encryption and decryption processes are illustrated in Figure 22 and Figure 23, respectively.

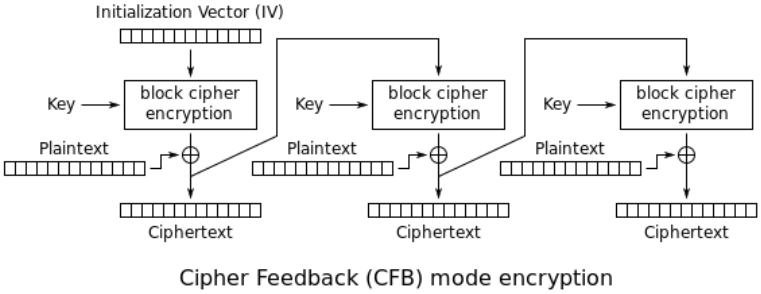

Figure 22 - CFB mode encryption [26].

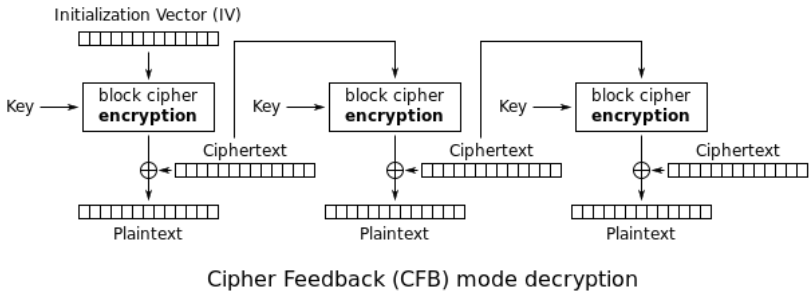

Figure 23 - CFB mode decryption [26].

2.4.4.5 OFB

The OFB mode transforms a block cipher into a synchronous stream cipher. It generates key stream blocks, which are then XORed with the plaintext blocks to get the ciphertext. The first key stream is computed by encrypting the IV using the block cipher encryption algorithms (e.g., AES-128 encryption). Then, the next key stream is computed by encrypting the key stream that was previously computed for the former block. This process is repeated for the following key streams, thus one advantage of the OFB mode is that the block cipher computations are independent of the plaintext and so it's possible to precompute all the key streams [1], however this computation cannot be parallelized since each key stream relies on the past key stream block (i.e., chaining). As depicted in Figure 24 and Figure 25, the idea of OFB consists in the computation of chained key streams that are XORed to the plaintext to be encrypted, and then XORed to the ciphertext to be decrypted. As a result of the use of an IV, the OFB encryption is non-deterministic, hence, encrypting the same plaintext twice results in different ciphertexts. As in the case of CBC and CFB modes, the IV should be a nonce [1]. Identically to other stream ciphers, flipping a bit in the ciphertext flips the same location bit of the plaintext [26].

Only the implementation of the block cipher encryption algorithm is required for OFB.

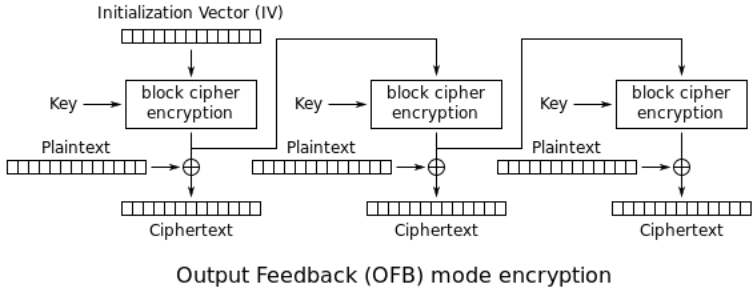

Figure 24 - OFB mode encryption [26].

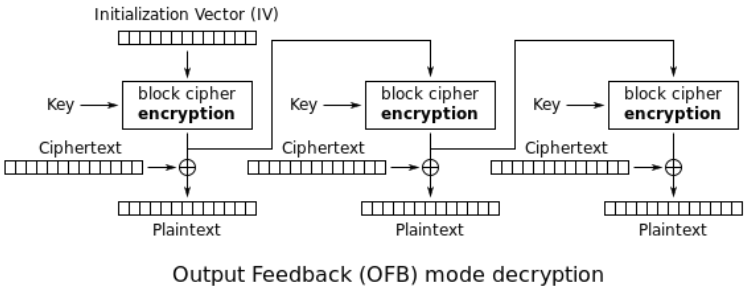

Figure 25 - OFB mode decryption [26].

2.5 Conclusions

In the present chapter, essential background information on firmware encryption has been provided, laying the foundation for the subject matter in this dissertation. Then, the concept of SCAs was introduced, highlighting their significance in the context of cryptographic devices. Given their particular relevance to the research conducted in this work, two types of sources of data-dependent leakage, namely, power consumption and EM emanation, were described. Additionally, strategies for measuring these leakage sources were contemplated along with the intuition behind leakage models, that is at the core of the PA techniques covered in Section 3.1. Within this scope, a comprehensive overview of the AES cryptographic algorithm is presented, including insights on how it is accomplished in software or hardware. Block cipher modes of operation are briefly addressed due to their applicability to firmware encryption and importance for the discussion on protective countermeasures.

The following chapter explores the state-of-art techniques for PA whose intuition can potentially be applied not only to exploit the data-dependent leakage of power consumption but also of associated EM emanation measurements. Methods for leakage assessment and pre-processing of traces are presented, along with a succinct list of open-source tools that facilitate part of the covered state of the art. Ultimately, countermeasures are categorized and elaborated upon according to their level of application. The primary objective of Chapter 3 is to provide readers with the essential knowledge and comprehensive perspective over the current state of the art, that is crucial for making well-informed and critical decisions when selecting the most suitable and effective techniques based on specific experimental circumstances and constraints.

Chapter 3
State of the Art

Prior to assessing whether a cryptographic implementation running on a device is exposed to key recovery attacks via power analysis, one must be fully aware of the existing state of the art techniques and the means to effectively accomplish them. Likewise, to reason about possible countermeasures it is essential to understand what the attacks consist of and how they are carried out, along with being familiar with countermeasures that have recently been proposed.

In this chapter, Power Analysis techniques are introduced, categorized, and further detailed, with a specific emphasis on AES. Leakage assessment mechanisms to identify and quantify the data-dependency in the acquired signals are presented, as well as pre-processing procedures that can be especially important when conducting power and EM-based side-channel attacks in practice. The state of the art countermeasures against PA techniques are explored, with a focus on their application for securing firmware encryption. Finally, a set of available open-source tools to facilitate some of the covered techniques and mechanisms for PA are briefly referenced.

3.1 Power Analysis techniques

In a Power Analysis SCA, the varying power consumption is measured and analyzed in order to extract the device's sensitive information or keys. These attacks exploit the fact that the instantaneous power consumed by a device depends not only, but also on the data being processed and instruction being executed [6]. PA techniques have been formerly employed to compromise the security of systems running strong and widely used robust cryptographic algorithms like the AES [5], Speck [23], Twofish, RSA [8], or even Elliptic-Curve Cryptography. Section 2.3 emphasizes that other sources of leakage, namely EM emanations, can be exploited using identical techniques as those employed for power fluctuations, since they leak similar information about the device's inner processing. As such, EM-based attacks can be considered a particular case of Power Analysis [37]. The reader is strongly encouraged to read Section 2.3 as it offers relevant context on this subject.

Conducting PA attacks in practice requires a proper measurement setup. The main components of a typical measurement setup are a power supply, a clock generator, the device under attack, a measurement circuit or an EM probe, an oscilloscope, and a PC. The measurement circuit or the EM probe provides a signal to the oscilloscope that is proportional to the power consumption of the device [6]. The PC controls the target device and the oscilloscope. It also gathers and stores the measured power traces for posterior analysis.

There are various state-of-the-art PA techniques that can be selected depending on the characteristics of the device and cryptographic implementation under attack, as well as on existing constraints (e.g., quality and number of measurements) and requirements of each approach (e.g., full control of device, perfectly aligned traces, etc.), as discussed next.

3.1.1 Simple Power Analysis

The most basic power side-channel attack, SPA (Simple Power Analysis), consists of directly observing and interpreting the electrical activity trace during cryptographic operations. The goal is to infer sensitive information from the operation-dependent component of power, $P_{operation}$ in (2), by understanding the algorithm being operated [7]. The reason why SPA mostly leverages operation-related leakage, instead of data-dependent leakage, is because $P_{operation}$ variations are generally more easily discernible and expressive than of P_{data}.

Take, for instance, the small program described in (14). Suppose that the power consumed from executing operation "op1" is different from "op2", either in terms of amplitude or duration. Under these circumstances, the power trace associated with the execution of the program will reveal perceptible differences for when "b" is equal to 0, or when it is not. Now, assuming b is a secret key bit, the mere observation of the power trace would suffice to deduce the value of b, which could ultimately be exploited to extract the full secret key. This conveys the idea behind SPA.

```
if b==0:
      r = op1(x)
else:                                                    (14)
      r = op2(x)
```

Figure 26 illustrates the instantaneous power consumption for an entire AES-128 encryption run. In this figure, the fact that the power consumed at a given moment in time depends not only but also on the instruction being executed, makes it possible to identify the instants where each round of AES occurs, due to the evident pattern leaked by the operation-dependent component of the consumed power. The 10 first peaks correspond to the rounds and the last one to the final AddRoundKey operation.

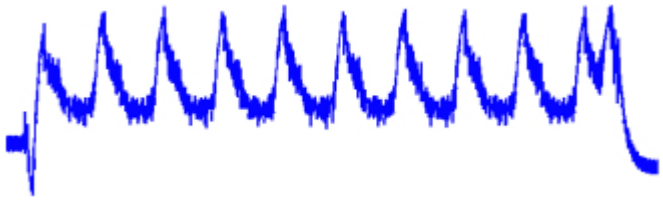

Figure 26 – Instantaneous power consumption of an AES-128 encryption run [7].

Adjusting the acquisition window illustrated in Figure 26, to include only the first and second rounds of AES-128, we're able to identify each round operation, namely the AddRoundKey, SubBytes, ShiftRows, and MixColumns steps, as in Figure 27.

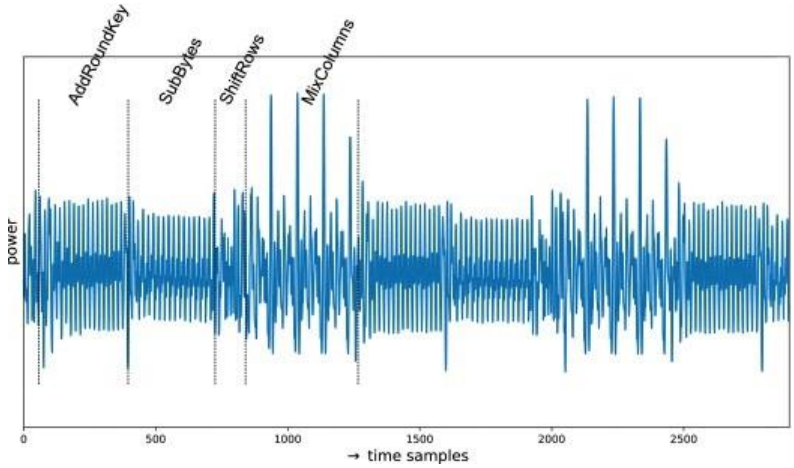

Figure 27 – Closer look at AES-128's first round and second round power consumption [8].

On the other hand, the data-dependent power component variation across measured traces is depicted in Figure 28: by changing a byte value of the AES key, from decimal 0 to 47, a different power level is obtained for the same time sample in which the byte is processed.

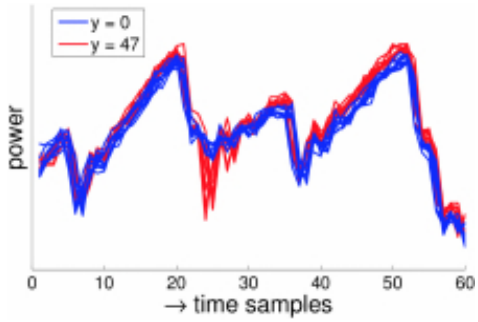

Figure 28 – Comparison of data-dependent power samples, when key-byte is decimal 0 and 47 [7].

Even though relevant features of the algorithm, such as the positions of the rounds, can be obtained from the observation of the power consumption associated with the execution of AES-128, it is unfeasible to extract the key bytes of AES solely through visual inspection of power traces. AES is safe against SPA attacks since it relies on executing the same sequence of operations, regardless of the value of the key [7].

Nevertheless, it's worth mentioning the classical attack against the RSA public key cryptosystem, using SPA, published in 1996 by Paul Kocher [8], that demonstrated how a seemingly secure implementation of a cryptographic algorithm (i.e., RSA) can expose sensitive information (i.e., secret key bits) to attack. The encryption portion of the RSA algorithm consists in raising the plaintext p to the power of the public key e modulo n: $(p^e)\ mod\ n$, while the decryption portion takes the ciphertext c and raises it to the power of the private key d modulo n: $(c^d)\ mod\ n$. RSA is secure because of its underlying security premise, that of factorization of large integers, is a hard problem. In some RSA implementations, the exponentiation operation was accomplished efficiently, in both software and hardware, through an algorithm known as "square and multiply" that can be described as in (15).

```
for bit in key:
      if bit == 1:
            square()
            multiply()
      elif bit == 0:
            square()
```
(15)

By observing the amount of power consumed by the device during the "square and multiply" algorithm, it can be determined when a square operation followed by a multiply operation occurred, meaning the key contains a 1 bit at that position, or when a single square operation was performed, leaking a 0 bit. Figure 29 shows how each distinct operation is clearly distinguishable from the corresponding power trace, which makes it possible to deduce the full secret key, bit-by-bit, through mere observation. The open-source cryptographic library [31] for the AVR microcontroller, is a real use-case example of this vulnerable function being employed.

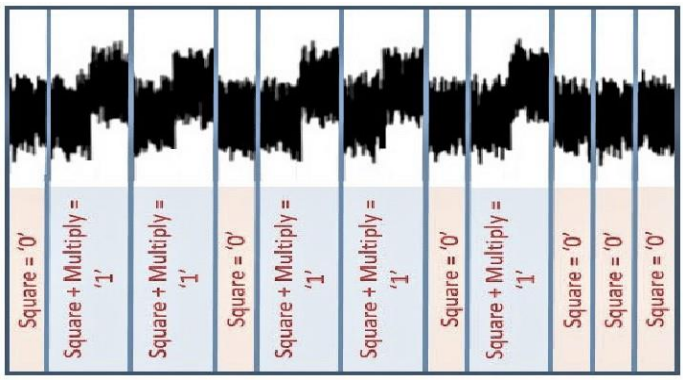

Figure 29 – Power trace of a portion of an RSA exponentiation operation [10].

When the device's power consumption is randomized internally by noise or by including dummy clock cycles, extracting secret information via SPA becomes almost impossible. Information about secret keys is often difficult to observe directly and is subject to interpretation, which led to further research and development of more automated power analysis strategies.

Randomizing the power consumption of a device through internal noise or the addition of dummy clock cycles makes it extremely difficult to extract secret information using SPA. Moreover, deducing secret keys from direct observation can be challenging and subject to interpretation, which motivated further research and development of more sophisticated and automated power analysis strategies.

3.1.2 Model-based

Over time, PA attack techniques have converged into two categories: Model-based and Profile-based.

In model-based approaches, such as DPA and CPA, a leakage model is assumed that defines a relationship between the power consumption of the device and the secret key it is employing [10]. These models are essentially functions used to map a data value to an associated, hypothetical, power consumption value, and as it will become clearer in the following subsections these are leveraged to make predictions on the expected power

consumption, more specifically the data-dependent power component, for given data values related to the secret key bytes. Section 2.3.3 elaborates on potential leakage models such as the Hamming-Distance model, the Hamming-Weight model, the Least Significant Bit model, and Other leakage models.

The efficacy of the attacks under this category is highly dependent on the accuracy of the assumptions made regarding technical details of the microcontroller (e.g., data bus width, register size, power management, etc.) and implementation details of the algorithm (e.g., whether it is implemented in software or hardware, how data is handled, the approach used to accomplish certain operations like the use of S-boxes or T-boxes lookup tables, etc.), that strongly influence the selection and adjustment of the assumed leakage model. Both model- and profile-based attacks require knowledge of the plaintext or ciphertext being processed.

Generally, model-based PA attacks consist of acquiring various power or EM measurements of the device under attack while it is encrypting or decrypting known random blocks of data, with the same secret key. Then hypothetical power consumption values are computed, using the leakage model, typically for all possible values over the key-byte space. To do so, a favorable intermediate value related to the plaintext or ciphertext and the secret key bytes is chosen. For instance, if the input of the last AddRoundKey operation of AES is selected as the intermediate value, it is said that the attacker is targeting the input of the last AddRoundKey operation since the hypothetical power values will be calculated based on such intermediate value, and these are ultimately compared to the actual measured power to deduce which possible key-byte value was most likely used in the instant where the intermediate value was processed. To sum up, the leakage model is used to predict the expected power of processing an intermediate value, for all possible key-byte guesses, and these are compared to the acquired traces, using a statistical distinguisher or other, to determine if the modeled guess matched the observed output, resulting in the leakage being useful for determining the key [10].

It is common to see, in the literature covering PA attacks against devices running AES, the output of the first SubBytes operation, depicted in Figure 30 by *z*, being chosen as the intermediate value whose leakage is modeled [5]. Usually, this is a particularly wise choice as it takes advantage of the non-linearity property of the S-box which results in enhanced distinguishability between the correct and incorrect key guesses, as attested in [6].

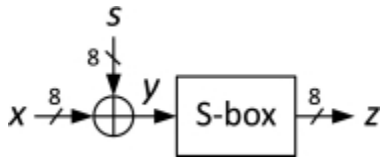

Figure 30 – AES AddRoundKey and SubBytes operations on a byte of data [7].

To provide a better understanding on why targeting the output of the first AddRoundKey isn't as beneficial as targeting the S-box output, the following explanation is provided: A one-bit difference at the input of S-box leads to a difference of several bits at the output. Consequently, if the key-byte guess differs in one bit from the correct key-byte, the output of the S-box will yield a value that is different in several bits from the one yielded for the correct key-byte. As a result, the correlation for all wrong key-byte guesses is significantly smaller than the correlation obtained for the correct one [6]. To illustrate the difference between the outputs of the first AddRoundKey and SubBytes operations, in function of all possible key-byte guesses, a fixed value was picked for byte *x* (i.e., plaintext-byte, with respect to encryption), and both *y* and *z* (i.e., first AddRoundKey and first SubBytes operation output, respectively) were obtained for all 256 possible values of *s* (i.e., key-byte) ranging between 0 and 255

decimal, inclusive. The results are presented in Figure 31 and Figure 32, where the x-axis represents the key-byte values, and the y-axis expresses the result of the operation. They show that, for the first AddRoundKey, close key-byte values result in values that are also close to each other. In contrast, for the first SubBytes operation, close key-byte values produce completely distinct and unrelated values, due to the non-linearity property of the S-box (e.g., in Figure 31, key-byte with decimal values 8, 9, 10, 11, 12, 13, 14, and 15 yield 175, 174, 173, 172, 171, 170, 169, and 168, respectively, while these same key-byte values are instead mapped to 121, 228, 149, 145, 98, 172, 211, and 194, in Figure 32).

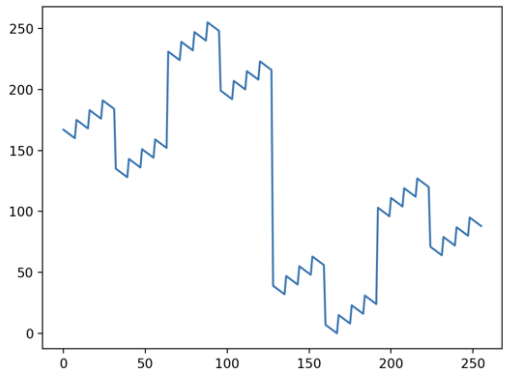

Figure 31 – First AddRoundKey output for plaintext-byte valued decimal 167 over all 256 possible key-byte values.

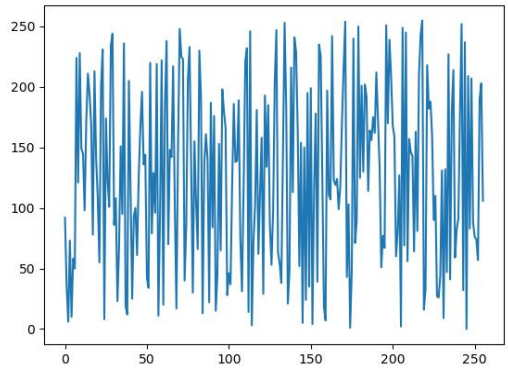

Figure 32 – First SubBytes output for plaintext-byte valued decimal 167 over all 256 possible key-byte values.

Thus, when computing hypothetical power consumption values in a PA attack, proximate key-byte guesses will produce similar hypothetical power consumption values for the AddRoundKey output. However, if the output of the S-box is considered instead, these will be completely distinct. This makes it easier to distinguish for which key-byte guess the hypothetical power consumption values better match the actual power consumed by the device, and consequently determine the correct key-byte more accurately, with a reduced number of traces. The output of MixColumns is not a common choice for the intermediate value because four bytes of the key are implied in each output byte, thus it would require exploring 2^{32} guesses, instead of 2^8 guesses as for SubBytes. In [37], the author explains why simply modeling the power consumption of the key hypothesis, instead of an intermediate value of the algorithm that depends on user-controlled input (i.e., plaintext), isn't viable: because the same key-byte guess produces the same power hypothesis.

Note that, if the attacks were to be conducted against AES-192 or AES-256, choosing the output of the first SubBytes operation wouldn't be enough to recover the full 192-bit or 256-bit key, respectively. Since AES operates on 128-bit blocks of data, larger keys must be fragmented to fit a 128-bit block. On AES-192, the first 128 bits of the 192-bit secret key are XORed with the plaintext block in the first AddRoundKey operation, while the remaining 64 bits of the key are then prepended to an extra 64 bits from the AES key scheduler, to be XORed to the output of first MixColumns operation. To extract the full 192-bit key, the strategy used to commonly target AES-128 implementations is applied to retrieve the first 128 bits of 192-bit secret key. From these first 128 bits, the real output of the first SubBytes, ShiftRows, and MixColumns of AES-192 can be calculated. Knowing the output of the initial MixColumns operation, whose first 64 bits are XORed to the last 64 bits of the key in the second AddRoundKey, the attacker is able to consider the output of the second SubBytes operation as

an intermediate value and apply the same technique that was used to deduce the first 128-bits of the key. As for AES-256, the same principle holds for the extraction of its full 256-bit key.

As previously noted, the comparison between hypothetical power and observed power consumption is achieved through a statistical model, also referred to as a "distinguisher". For each key-byte guess, the distinguisher quantifies how well its corresponding hypothetical power consumption values match the actual measured consumption of the device. Typically, the highest absolute value produced by the distinguisher reveals the correct key-byte guess. These can be based on the Difference-of-Means, Pearson correlation coefficient, Bayesian classification, mutual information, or others that allow to determine whether a set of values follow a hypothesized distribution (i.e., Goodness of Fit tests).

As it will become clearer in the specification of the following model-based attacks, their efficacy is limited by the extent of the employed leakage models and distinguishers [7]. In contrast to SPA, these focus exclusively on the data dependency of the power traces.

3.1.2.1 Differential Power Analysis

DPA attacks were first introduced by *Kocher et al.*, in 1999, targeting DES [4], and later demonstrated on both software and hardware implementations of AES [6], by *Mangard et al.*, in 2007.

DPA is a statistical technique that involves partitioning the set of acquired traces into subsets and comparing the difference of the averages of these subsets at each point in time [11]. Like all model-based PA attacks, in DPA, the adversary starts by capturing and storing the power consumption of the target device running the cryptographic algorithm, with a constant secret key, for a set of known random input (i.e., encrypting multiple random known plaintexts using a fixed unknown key). Thus, for each i = 0, ..., N cryptographic run, with a random known input D_i and fixed secret K, a trace T_i, describing the instantaneous power consumption of the device over the execution period, is acquired. The goal is to deduce K by analyzing the power consumption variation for the different inputs while K was constant. Assuming K to be a single byte, it can be any of 256 possible values. Designating K' as the guess of K, each possible K' results in a different output of the operation $C'_i = f(D_i, K')$, the so-called hypothetical intermediate value, for each known input D_i. So, for each guess K', the attacker computes the corresponding $C'_i = f(D_i, K')$ values for every i = 0, ..., N known input. Then, the hypothetical power consumption P'_i values, associated with the hypothesized processing of the output of operation $f(D_i, K')$, is computed from all previously calculated C'_i using a binary leakage model (e.g., the Least Significant Bit model). Subsequently, the actual measured power traces are partitioned into two separate subsets based on these hypothetical power values. Accordingly, a trace T_i is placed on subset '0' if the corresponding hypothetical power (e.g., LSB($f(D_i, K')$)) value is bit 0, elsewise it is inserted in subset '1'. Finally, for each time instant $t = 0, \ldots, \#T_i$, the mean of each partition is computed over all samples corresponding to the same time instant t, and the difference between the mean values of samples at instant t of subset '0' and of subset '1' is calculated, in what's referred to as the DoM (Difference-of-Means). At last, a total of $256 \times \#T_i$ DoM values have been calculated. The correct guess for K is the one that yields the highest absolute DoM value. Figure 33 describes the DoM value across samples, obtained in [5], for the correct key hypothesis. The large peak reveals the instant where the processor manipulated the intermediate value based on which the DoM was calculated, while for all other samples the DoM approaches zero.

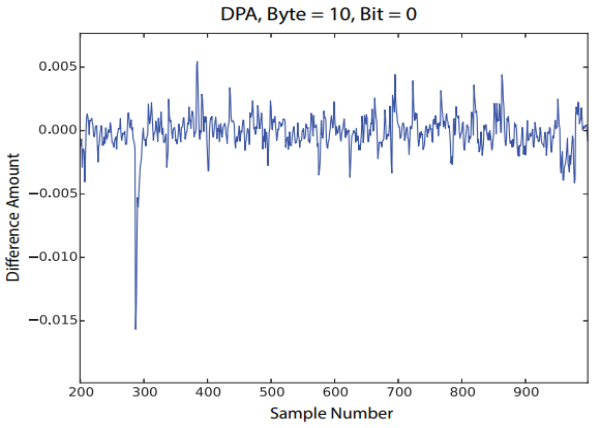

Figure 33 – DoM over samples for a key-byte guess ($K' = 10_{dec}$), based on LSB model [6].

The DoM is a statistical test used to determine whether there is a significant difference between two populations. As such, given the assumption that the data-dependent power component of a device follows a leakage model, this statistical test allows to determine which key guess produces the power hypothesis that better matches the actual observed power consumption of the device. For example, if an attacker considers the LSB model, it is being assumed that processing data whose least significant bit is set to 1 consumes differently than data whose least significant bit is set to 0. For each key guess, the measured traces are partitioned accordingly, and so, the key guess that results in the grouping for which the absolute DoM is the highest, at a particular instant, is the most compatible with the assumption that the instantaneous power significantly different depending on the LSB of the processed value. If the grouping of the acquired traces, resultant from a key hypothesis, is uncorrelated to the actual measurements, the DoM between the subsets will approach zero as the number of traces increases. Elsewise, if the grouping is correlated to the trace measurements, the DoM will tend to a non-zero value, perceptible as a peak similar to the one of Figure 33. Even if the difference of the means is very small (such as the effect of a single transistor within one chip in a complex device), the difference will eventually become statistically significant given enough traces [11]. In [6], a statistical test named Distance-of-Means was proposed as an improvement over the Difference-of-Means method, for the purpose of DPA, with the advantage of, in addition, taking variances into account. Note that, in the available literature, in this context, the leakage model can appear referred to as the "selection (or partition) function", due to the fact that it is used to partition the captured traces into separate subsets.

The AES algorithm operates on 16-byte blocks of data. Typically, each byte is processed individually, which allows an attacker to target each key-byte individually, circumventing the impracticability of guessing and checking all key candidates in a 128-bit, 198-bit, or 256-bit search-space, and reducing the complexity of performing the key and check operation [5]. As such, a general strategy for DPA attacks on AES-128 encryption, to extract a secret key-byte K, can be delineated as follows [6]:

1. **Choosing an intermediate result of AES**.

   The intermediate result $C'_i = f(D_i, K')$ must be described as a function of a plaintext (or ciphertext) byte and a key byte. As justified in Section 3.1.2, the output of the first SubBytes operation is a particularly advantageous choice: $C'_i = f(D_i, K') = \text{SBox}(D_i \oplus K')$

2. **Capturing traces and corresponding known data**.

   Measure the power consumption of the device for N encryption of random known 16-byte blocks of data. Record each trace i = 0, ..., N and corresponding plaintext-byte D_i.

   DPA is a univariate attack, meaning each sample is analyzed independently. Therefore, proper alignment of traces is crucial for the analysis. Specifically, each sample of column $t = 0, \ldots, \#T_i$, in list of traces T, should describe the power consumption of the device at precisely the same moment of execution. Synchronization techniques covered in Section 3.3.2 can be applied to misaligned traces to provide the necessary alignment.

3. **Calculating hypothetical intermediate values**.

   The key-byte K can be any of 2^8=256 possible values $K' = 0, \ldots, 255$. Given the plaintext-byte D_i and a key hypothesis K', all ($N \times 256$) hypothetical intermediate values can be computed and stored in a matrix V, as in (16).

```
for i in range(N):
      for K' in range(256):                    (16)
            V_i,K' = f(D_i, K')
```

4. **Mapping intermediate values to corresponding hypothetical power consumption values, based on assumed leakage model**.

   The hypothetical power consumption values associated with the hypothetical processing of the intermediate values from each key-byte candidate K', are computed using the leakage model δ and stored in a ($N \times 256$) matrix H, as $H_{i,K'} = \delta(V_{i,K'})$ in (17). Recall that the absolute values of power consumption aren't relevant in the scope of Power Analysis, but rather the relative differences between the hypothetical power consumption values.

   For DPA, the Least Significant Bit (LSB) power leakage model is a potential option. The requirement is to have a leakage model that yields one of two values, so that subsequently the captured traces can be partitioned between two groups. Hence, it is not possible to use the Hamming-Weight model directly, for example. A possibility is to reduce it to a binary model [6], which can be done by setting $H_{i,K'} = 1$, if $HW(V_{i,K'}) \geq 4$, and $H_{i,K'} = 0$, if $HW(V_{i,K'}) < 4$. However, it is clear that such a binary model can't describe the power consumption of a target device as accurately as a non-binary model. Also, a disadvantage of a model like the LSB is the loss of information due to ignoring all other bits. To cope with this limitation, a practical strategy could be to perform a separate attack for each of the eight bits, instead of considering only the LSB.

```
for i in range(N):
      for K' in range(256):
            H_i,K' = δ(V_i,K')                 (17)
```

5. **Comparing hypothetical power consumption with actual power consumption using a statistical test, to determine the key candidate *K'* that is most likely correct**.

   In this final step, the hypothetical power consumption values in H are compared with the samples of the collected traces, using a statistical model. Each column of H (i.e., the hypothetical power values corresponding to a given key hypothesis) is compared against each column $t = 0, \ldots, \#T_i$ of the list T of acquired traces (i.e., the actual power consumption for all traces at instant t). The result is a quantification of how well the

hypothetical power consumption associated with the key hypothesis *K'* agrees with the actual measured power at instant *t*. As such, these final results can be stored in a $(\#T_i \times 256)$ matrix *R*. For this purpose, *Kocher et al.*, in its original work on DPA, proposed the use of the DoM statistical test, as stated earlier. In essence, this step can be mathematically described by (18), and implemented as in (19).

$$R_{t,K'} = \frac{\sum_{i=0}^{N}[\delta(V_{i,K'}) \times T_{i,t}]}{\sum_{i=0}^{N}[\delta(V_{i,K'})]} - \frac{\sum_{i=0}^{N}[(1-\delta(V_{i,K'})) \times T_{i,t}]}{\sum_{i=0}^{N}[(1-\delta(V_{i,K'}))]} \tag{18}$$

```
for K' in range(256):
      for t in range(#T_i):
            #Compute DoM
            subset1_sum, subset1_size = 0, 0
            subset0_sum, subset0_size = 0, 0
            for i in range(N):
                  if (H_{i,K'} == 1):
                        subset1_sum+=T_{i,t}
                        subset1_size+=1
                  elif (H_{i,K'} == 0):
                        subset0_sum+=T_{i,t}
                        subset0_size+=1
            subset1_mean = subset1_sum/subset1_size
            subset0_mean = subset0_sum/subset0_size
            R_{t,K'} = subset1_mean-subset0_mean
```
(19)

The five-step process shall be repeated for each byte of the 16-byte AES-128 secret key.

While the DPA attack was the first presented automated method for Power Analysis, more efficient and sophisticated methods have been proposed, such as the CPA, discussed next.

3.1.2.2 Correlation Power Analysis

The CPA attack was first presented in 2004 by *Brier et al.* in [14]. Whereas DPA looked at simple differences between two groups of data, the CPA attack enables more accurate and general assumptions regarding the data dependency of the power, meaning more accurate leakage models may be considered: the Hamming-Weight model, where a linear relationship between the Hamming weight (number of bits set to 1) of the processed data and the power consumption is assumed, and the Hamming-Distance model, where the leakage is considered to be related with the number of bits changing states during processing. These models suit most devices well in practice, especially for leakages from registers and data bus. A general understanding is that the HW model is suitable against software-based implementations while the HD model is applicable for hardware-based implementations [49]. Nevertheless, some might leak other properties of the processed data. In such cases, it may be worth exploring other leakages models, as discussed in Section 2.3.3.4, that better reflect the actual behavior of the device under attack.

To compare hypothetical power values with the actual observed power consumption, CPA employs a statistical distinguisher, called the Pearson's correlation coefficient, to quantify the linear correlation between the two datasets. DPA's attack strategy, described in the previous section, can be adapted for CPA, by replacing the DoM distinguisher by the Pearson's

correlation coefficient, eliminating the constraint of having to assume a binary leakage model like the LSB, and instead use the HW, the HD, or other.

Once more, the attacker starts by acquiring power traces from the target device while it executes the cryptographic algorithm for a set of known random input (i.e., plaintexts or ciphertexts) using a constant secret key. Accordingly, for each i = 0, ..., N cryptographic run, with a random known input D_i and fixed secret K, a trace T_i, describing the instantaneous power consumption of the device over the execution period, is acquired. The goal is to infer K by analyzing, for distinct inputs, the varying power associated with the processing of an intermediate value $C_i = f(D_i, K)$, while K remains constant. Assuming K to be a single byte, and designating K' as the guess of K, for each possible K' the attacker computes the corresponding hypothetical intermediate value $C'_i = f(D_i, K')$, for every i = 0, ..., N known input. In the context of targeting an AES-128 encryption implementation, operation f is frequently the first SubBytes operation, as explained in Section 3.1.2. Then, for each K', the hypothetical power consumption value P'_i is calculate from the value of C'_i, using the assumed leakage model δ, for every i = 0, ..., N. Finally, the Pearson's correlation between the samples at instant t of the trace set, and the hypothetical power consumption values C', is computed for each K guess over all samples t = 0, …, $\#T_i$. The highest absolute correlation value reveals the most likely correct key guess, and instant t at which the intermediate value's leakage, according to the chosen leakage model, was detected.

As such, the general strategy to recover the secret key used in AES-128 encryption, that was delineated for DPA in the previous section, can be readjusted for CPA, as follows, while the first three steps remain as defined for DPA:

4. **Mapping intermediate values to corresponding hypothetical power consumption values, based on assumed leakage model**.

   The hypothetical power consumption values associated with the hypothetical processing of the intermediate values from each key-byte candidate K', are computed using the assumed leakage model δ and stored in a ($N \times 256$) matrix H, as $H_{i,K'} = \delta(V_{i,K'})$ in (17).

   To offer a more practical illustration of these steps, consider the two following examples:

   a) Targeting an AES-128 software-based encryption implementation running on a microcontroller, where the S-box is realized as a lookup table, each byte is processed individually, every AES operation is explicitly implemented, and the data-bus lines are precharged to 0 before transmitting a data-byte. Based on this (typically unavailable) information, it is wise to assume the HW leakage model and to select the output of the first S-box lookup operation as the intermediate value. This means that we'll be determining for which key hypothesis K' the Hamming weight of the first S-box output (i.e., expected relative power consumed from 'looking up' the value SBox($D_i \oplus K'$)) is more closely related with the actual observed power, at a specific instant t. As such, for each acquired trace i = 0, ..., N and corresponding plaintext-byte D_i, steps 3. and 4. can be summarized into (20) to compute both hypothetical intermediate values and corresponding hypothetical power values.

```
for i in range(N):
      for K' in range(256):
            V_{i,K'} = SBox(D_i ⊕ K')
            H_{i,K'} = HW(V_{i,K'})
```
(20)

b) The device under attack is a hardware accelerator running an ASIC-based implementation of AES-128, with 8-bit registers, where, in particular, the register where D_i is initially stored is overwritten, by the first AddRoundKey operation, with the value $D_i \oplus K$, and subsequently $D_i \oplus K$ is overwritten with the output of the next operation, i.e., SBox($D_i \oplus K$), and so on. With this information in mind, it is wise to assume the HD leakage model to model the data-dependent power leakage of such register. In light of the reasoning outlined in Section 3.1.2, regarding the choice of intermediate values, it is clear that considering the HD leakage between the plaintext and the initial AddRoundKey operation proves ineffective in the context of PA since, $HD(D_i, D_i \oplus K') = HW(D_i \oplus D_i \oplus K') = HW(K')$. However, exploiting the HD leakage between the output of the AddRoundKey and of the SubBytes operation is significantly advantageous due to the non-linearity property of S-box: $HD(D_i \oplus K', \text{SBox}(D_i \oplus K')) = HW((D_i \oplus K') \oplus (\text{SBox}(D_i \oplus K')))$. Accordingly, steps 3. and 4. can be accomplished as in (21).

```
for i in range(N):
      for K' in range(256):
            V_{i,K'} = (D_i ⊕ K') ⊕ SBox(D_i ⊕ K')
            H_{i,K'} = HW(V_{i,K'})
```
(21)

The closer the assumed leakage model aligns with the target's characteristics, the stronger the linear relationship between the hypothetical and the real power consumption values, for the correct key-byte guess. This makes it easier to guess the correct key-byte K, resulting in a more effective CPA attack.

5. **Comparing hypothetical power consumption with actual power consumption using a statistical test, to determine the key candidate *K'* that is most likely correct**.

In this final step, the Pearson's correlation coefficient is used to quantify the linear relationship between the hypothetical power consumption values H and the observed power consumption, exposing how strong the relationship is between the hypothetical and the observed power. The Pearson's correlation coefficient ranges between -1 and 1, inclusive, and is mathematically described by formula (22), where *cov* is the covariance, σ_X is the standard deviation of X, σ_Y is the standard deviation of Y, var is the variance, and X is the hypothetical power consumption values and Y the observed power consumption values.

$$PearsonCorr(X,Y) = \frac{cov(X,Y)}{\sigma_X \times \sigma_Y} = \frac{cov(X,Y)}{\sqrt{var(X) \times var(Y)}} \tag{22}$$

In practice, to identify the best key-byte candidate K', one must go through each column of the ($N \times 256$) matrix H of hypothetical power values (i.e., $H_{1..N,K'}$), and for each column of the ($N \times \#T_i$) list of traces T (i.e., $T_{1..N,t}$), that contains the power observed over all samples of instant t, compute the corresponding Pearson's correlation coefficient between the two columns, for each $t = 0, \ldots, \#T_i$. Each coefficient value can be stored in a ($\#T_i \times 256$) matrix R. This procedure can be mathematically described as (23) and further implemented as in (24).

$$R_{t,K'} = \frac{\sum_{i=0}^{N}\left[(H_{i,K'} - avg(H_{1..N,K'})) \times (T_{i,t} - avg(T_{1..N,t}))\right]}{\sqrt{\sum_{i=0}^{N}\left[(H_{i,K'} - avg(H_{(1..N,K')})^2\right] \times \sum_{i=0}^{D}\left[(T_{i,t} - avg(T_{1..N,t}))^2\right]}} \quad (23)$$

```
for K' in range(256):
      for t in range(#T_i):
            # Compute Pearson's corr coef …
            # … between column K' of H and column t of T
            avg_H = 0
            avg_T = 0
            for i in range(N):
                  avg_H += H_i,K'
                  avg_T += T_i,t
            avg_H /= N
            avg_T /= N
            cov = 0
            std_H = 0
            std_T = 0
            for i in range(N):
                  cov += (H_i,K' – avg_H)*(T_i,t - avg_T)
                  std_H += (H_i,K' - avg_H)**2
                  std_T += (T_i,t - avg_T)**2
            R_t,K' = cov/sqrt(std_H*std_T)
```

(24)

Finally, the entry of *R* holding the highest absolute value, exposes the correct key-byte guess and the sample where the leakage was predominantly detected.

Figure 34 portrays the general strategies delineated for the DPA and CPA attacks, with a substantial degree of abstraction.

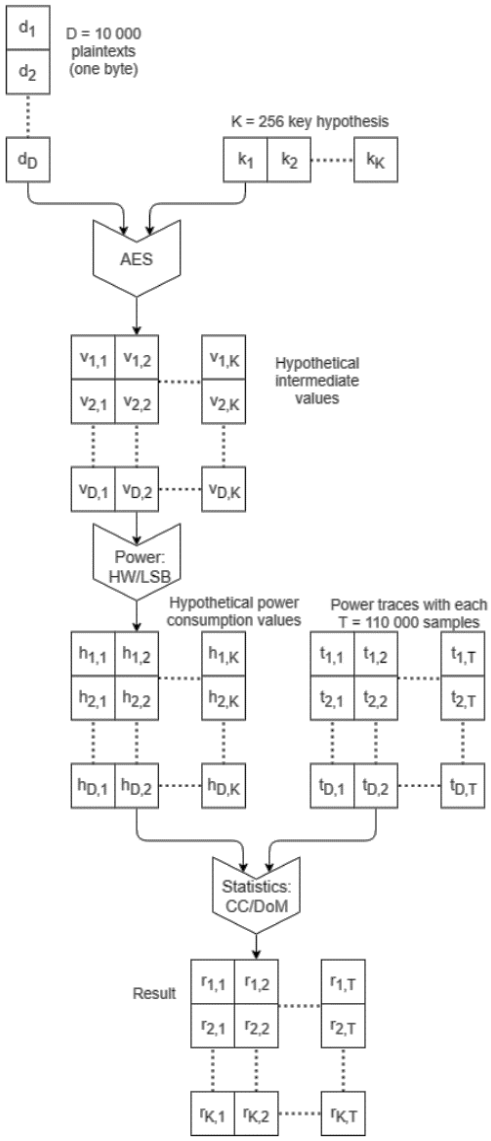

Figure 34 – High-level description of DPA and CPA attacks [6].

Experiments conducted in the existing literature have shown that CPA outperforms DPA in terms of efficiency, as it requires fewer traces to extract the same cryptographic key. For instance, in [13], it was observed that CPA achieved accurate results with only 400 traces only, while DPA required a minimum of 2150 traces to achieve a comparable level of certainty. This difference can be explained by the influence of both leakage and statistical models: the correlation coefficient, applied in CPA, takes both the differences and the variances of the variables into account, while the DoM only considers the differences [13]. Also, the fact that the LSB power model, used in DPA, only considers one bit and ignores all others, impacts its accuracy making it necessary to have a larger number of power traces in order to eventually obtain a statistically conclusive result. In contrast, the HW model, that can be used in CPA, does account for all bits, allowing CPA to be more efficient and more accurate [13].

Figure 35 and Figure 36 feature the results of a DPA and CPA attack, respectively, for all key-byte candidates, against an AES-128 software-based implementation of the first SubBytes operation running on the ATMega328P microcontroller of the Arduino Uno. The results, obtained under the conditions detailed at [3], demonstrated that both DPA and CPA succeeded at yielding the correct key-byte guess for the same number of analyzed traces,

however, the correct result is noticeably more distinct and easier to interpret for CPA than for DPA, supporting *Brier et al.*'s original finding that CPA attacks have the potential to generate more clear results [14]. Even though these were produced in a white-box testing environment, they still provide a relevant quantifiable comparison of the two attacks under the same conditions.

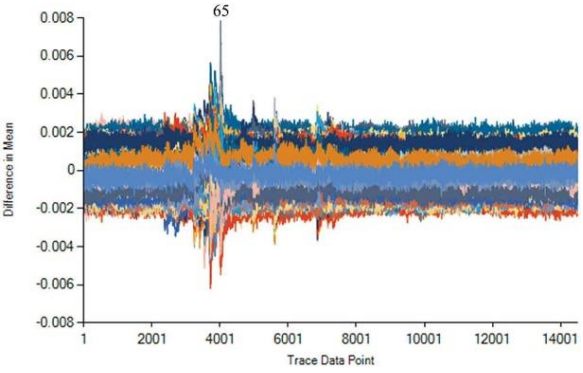

Figure 35 – DPA attack on key-byte [3].

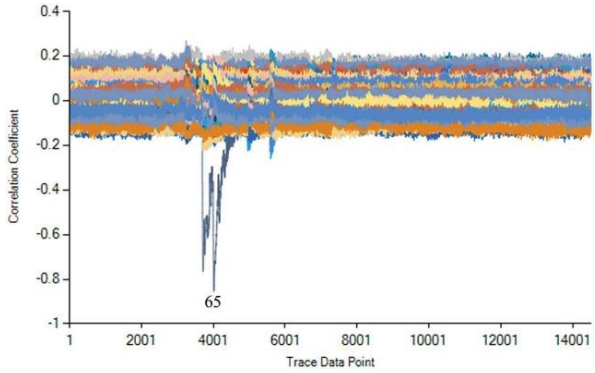

Figure 36 – CPA attack on key-byte [3].

3.1.2.3 Mutual Information Analysis

In 2008, *Gierlichs et al.*, published a paper entitled "*Mutual Information Analysis: a generic side-channel distinguisher*" [50], introducing a novel distinguisher for ranking key guesses based on the mutual information between observed measurements and hypothetical leakage. Mutual Information Analysis (MIA) consists of applying this information-based distinguisher, to perform the comparison between the observed power consumption and the hypothetical leakage. The mutual information distinguisher, as most statistical tests, is bounded in its efficiency to recover keys by the assumed leakage model. However, it aims at generality in the sense that it is expected to lead to successful attacks without requiring specific knowledge or restrictive assumptions about the device being targeted, which means that it is meant to cope with less precise leakage predictions, as opposed to other types of side-channel attacks like DPA or CPA that can only be accomplished if the leakage model assumption holds [50].

Let X and Y be random variables. The reduction in uncertainty on X that is obtained by having observed Y, is exactly equal to the information that on has obtained on X by having observed Y [50]. Mutual information measures how much knowing the value of one variable reduces the uncertainty about the other variable. This is based on the notion of entropy *E*, which is a measure of uncertainty, as introduced by Shannon. As such, mutual information quantifies the degree of dependence or association between the variables and can be formulated as in (25). The mutual information satisfies $0 \leq \mathrm{I}(\mathrm{X};\mathrm{Y}) \leq E(\mathrm{X})$. The lower bound is achieved if and only if X and Y are independent, while the upper bound is reached when Y fully determines X. Hence, the higher the mutual information, the stronger the relationship between variables is [50].

$$\mathrm{I}(\mathrm{X};\mathrm{Y}) = E(\mathrm{X}) - E(\mathrm{X}|\mathrm{Y}) = E(\mathrm{X}) + E(\mathrm{Y}) - E(\mathrm{X},\mathrm{Y}) = \mathrm{I}(\mathrm{Y};\mathrm{X}) \tag{25}$$

Similarly to the statistical model application on DPA and CPA, and referencing back to the general strategy specified in subsection 3.1.2.1 and 3.1.2.2 to conduct DPA and CPA, respectively, the MIA distinguisher is applied after the hypothetical power consumption values have been computed based on a chosen power leakage model. The intuition remains unchanged. The key guess *K'* that maximizes the mutual information between the observations and the

hypothetical power values, is most likely the correct key *K*. When considering the mutual information between the hypothetical power values for a key candidate *K'* and the actual observed power consumption at instant *t*, four possible cases can occur. When the key hypothesis is incorrect $K' \neq K$ and *t* is the wrong time instant (i.e., intermediate value was not processed by the target at instant *t*), or the key guess is correct $K'=K$ but the timing *t* is incorrect, or the key guess is incorrect $K' \neq K$ but instant *t* is correct, the mutual information value should tend to 0 since the two the hypothetical leakage and observed power consumption are independent. On the other hand, for the correct key guess $K'=K$ and at the right instant *t*, the highest mutual information value should be obtained since the variables are dependent by definition [50].

Entropy is at the core of this attack; however, in practice, computing the exact value of entropy is infeasible. Luckily there are methods to estimate this value. These rely on approximating the probability density function of the traces to estimate the probabilities of the variables to compute their corresponding entropies and ultimately the mutual information. The histograms-based, the kernel-based, and the B-spines-based entropy estimation algorithms are well explained, within the context of PA attacks, in [15]. It is outlined that the outcome of MIA is very sensitive to the choice of the estimator given that a poor approximation of the entropy can lead to the incorrect inference of the key [15]. The estimation methods trade-off precision for speed. The histograms-based approach is faster but not precise enough to result in an estimated entropy really close to the real one. The B-splines-based approach delivers a better approximation than the histograms-based method but requires a greater computation effort. The kernel-based estimator results in better entropy estimation than the B-splines method, but at an even higher processing cost [15].

MIA truly is a generic distinguisher in the sense that it can capture linear, non-linear, univariate, and multivariate relationships between models and actual observed leakages [10]. Because the DoM and Pearson's correlation distinguishers analyze a probability distribution at most by its mean and variance, they are inappropriate if the Gaussian assumption does not hold. Pearson's correlation even requires the additional assumption of a linear relation between modeled and observed leakage [50]. In scenarios where these assumptions are not valid or the choice of leakage model poses a greater challenge than usual, applying the mutual information -based distinguisher may prove effective. *Gierlichs et al.* point out that measurements shouldn't be averaged unless the Gaussian assumption is justified.

However, for standard CMOS technology these assumptions typically hold and the device's leakage behavior can be approximated by the HW or HD model [50]. In such cases, when the leakage model fits well to the physics, the CPA attack does an excellent job in characterizing the linear dependency between the modeled and the observed power quite fast, outperforming MIA by far [15]. MIA's generality comes at the price of a decreased attack efficiency, requiring a larger number of measurements for successful key recovery and more time to discriminate the correct key candidate (i.e., estimating the mutual information between sets takes significantly more time than computing the DoM or the Pearson's correlation coefficient [10]).

3.1.2.4 Linear Regression Analysis

In [2], *Schindler et al*. suggested an efficient profiling method for SCA, entitled the Stochastic Attack (SA), where an attacker with control over the key and the input to the cryptographic algorithm was able to approximate a leakage function δ from the actual observed power, using linear regression, and later apply the model in the analysis of traces of an identical device for secret key extraction. Later, in [19], a powerful non-profiled technique called Linear

Regression Analysis, based on the ideas introduced in [2], was proposed by *Doget et al.* as a form of "robust side channel attack" said to be "able to succeed with only a very limited knowledge on how the device leaks information". LRA is regarded as an alternative to CPA using the R^2 instead of the correlation coefficient ρ to distinguish key candidates [49]. The core idea of LRA is to compute, for each key candidate, a set of coefficients that linearly model the data-dependent leakage according to the observed power, and ultimately deduce the correct key guess by determining which model better fits the observed power through the *R*-squared *goodness of fit* measure.

Let ($C_i[n\text{-}1]$, …, $C_i[0]$) be the binary decomposition of the *n*-bit intermediate value $C_i = f(D_i, K)$ targeted by the attack. Let *T* be the list holding each trace T_i, with i = 0, ..., *N*, and $T_{i,t}$ the sample describing the instantaneous power consumption of T_i at instant *t*, with *t* = 0, …, $\#T_i$. The ($N \times 256$) matrix *V* contains the hypothetical intermediate value $C'_i = f(D_i, K')$ for each key guess *K'* and random known input D_i used for the acquisition of trace *i*, similarly to what was portrayed in the previously covered attacks (recall, for instance, step 3. of DPA's general strategy). In short, the LRA can be described as follows, with the three initial steps defined as for DPA:

4. **Constructing the leakage models between observed power consumption and hypothetical intermediate values**.

   For each key guess *K'*, estimate the set of coefficients (α'_{-1}, α'_0, …, α'_{n-1}) such that the distance between approximated $P'_{instant}$ and the observed power consumption at instant *t* is minimal, for all i = 0, ..., *N*, as of (26).

$$P'_{instant} = \alpha'_{-1} + P'_{data} = \alpha'_{-1} + \delta'\left(V_{i,K'}\right) = \alpha'_{-1} + \sum_{b=0}^{n-1}\left(\alpha'_b \times V_{i,K'}[b]\right) =$$
$$= \alpha'_{-1} + \left(\alpha'_0 \times V_{i,K'}[0]\right) + \cdots + \left(\alpha'_{n-1} \times V_{i,K'}[n-1]\right) \quad (26)$$

   Note the resemblances of (26) with Equation (6): α'_{-1} describes the variable *b* of (6) that encloses all power components that are considered noise to the attack, and as such, the coefficients (α'_0, α'_1, …, α'_{n-1}) represent the respective weights or contributions of each variable ($C_i[0]$, $C_i[1]$, …, $C_i[n\text{-}1]$) to the data-dependent leakage. In the case of (6), Hamming distance between the two switching values is assumed to be the only contributing factor to the data-dependent leakage. Here, an advantage of LRA over CPA becomes evident, as (26) enables the characterization of the data-dependent leakage of the device, at a bit level, where each bit contribution to the leakage is modeled through the estimation of the coefficients. Recalling that power consumption and electromagnetic emissions both result from logical transitions occurring on circuit wires, it is realistic to assume that every bit of a processed variable contributes independently to the overall instantaneous leakage. Note how having each coefficient (α'_0, α'_1, …, α'_{n-1}) contribute equally to the leakage (e.g., (α'_0, α'_1, …, α'_{n-1}) = (1, 1, …, 1)), would be equivalent to an Hamming weight leakage, while the improbable circumstance where only the coefficient corresponding to the least significant bit contributed to the leakage (e.g., (α'_0, α'_1, …, α'_{n-1}) = (1, 0, …, 0)), would imply a LSB leakage.

   The set of coefficients can be efficiently computed by performing a so-called linear regression. To do so, for each key guess *K'*, the ($N \times (n+1)$) matrix $M_{K'}$ is constructed as in (27).

$$\mathrm{M}_{K'} = \begin{bmatrix} 1 & V_{1,K'}[0] & \cdots & V_{1,K'}[n-1] \\ 1 & V_{2,K'}[0] & \cdots & V_{2,K'}[n-1] \\ \vdots & \vdots & \ddots & \vdots \\ 1 & V_{\mathrm{i},K'}[0] & \ldots & V_{\mathrm{i},K'}[n-1] \\ \vdots & \vdots & \ddots & \vdots \\ 1 & V_{N,K'}[0] & \cdots & V_{N,K'}[n-1] \end{bmatrix} \tag{27}$$

Then, the ordinary least-square method is applied, according to (28), to calculate the coefficients for each *K'* and instant *t*.

$$\alpha_{K'} = (\alpha'_{-1}, \alpha'_0, \ldots, \alpha'_{n-1}) = \ ((\mathrm{M}_{K'})^{\mathbf{T}} \cdot \mathrm{M}_{K'})^{-\mathbf{1}} \cdot (\mathrm{M}_{K'})^{\mathbf{T}} \cdot T_{1..N,t} \tag{28}$$

By definition, this method outputs the vector $\alpha_{K'}$ that optimally minimizes the Euclidean distance between vectors (26) over $V_{1..N,K'}$ and $T_{1..N,t}$.

5. **Compare the estimated models with the observed power measurements**.

Contrary to the previously covered attacks, which involved the assumption of a fixed leakage model, in LRA, for each key candidate *K'* and instant *t*, a district leakage model is computed. To distinguish the correct key guess *K'=K*, the *R*-squared value, also referred to as the coefficient of determination, which indicates how well a linear model fits a set of observations, is calculated between the estimated leakage model and the observed power measurements, in what consists of a *goodness of fit* test.

To present the reader with a more comprehensive view of the current and prior step, the following pythonic pseudo-code is provided in (29). The R^2 value is stored in the $(\#T_\mathrm{i} \times 2^n)$ matrix *R*. The entry with the highest R^2 value reveals the correct key guess and the instant *t* at which the leakage better fitted the observation. The R^2 calculation is formally expressed in (30).

```
for K' in range(2**n):
      for t in range(#T_i):
            # construct M_K'
            M = [
            [(V_i,K'[b] if b>-1 else 1) for b in range(-1, n)]
                  for i in range(N)
                ]
            # calculate the (n+1) coefficients: α_K'
            α = inverse(transpose(M)·M)·transpose(M)·T[:][t]
            # calculate the corresponding R-squared value
            R_t,K' = euclidean_dist(T[:][t] - M·α)/var(T[:][t])
```

(29)

$$R_{t,K'} = \frac{\left\| T_{1..N,t} - \mathrm{M}_{K'} \cdot \alpha_{K'} \right\|^2}{variance(T_{1..N,t})} \tag{30}$$

The R^2 is a value within the range of [0,1] and it is said to be closely related to the correlation coefficient [19]. In fact, if one was to apply the same approach articulated for LRA, but considering equation (6) in-place of (26), a similar result to CPA under the HD model assumption could be expected [49].

Depending on the device's architecture, the leakage can be 8-bit, 16-bit, 32-bit, 64-bit, and even 128-bit. With CPA, guessing a multi-byte *K* value results in high computation cost rendering the attack impractical for 32-bit, 64-bit and 128-bit architectures. A workaround is to separate *K* into different parts and conduct CPA for each part. When targeting a segment of *K*, leakages from the remaining parts are considered noise. Even though this strategy succeeds at extracting *K*, the efficiency is reduced since only part of the information is used [49]. In consideration of this limitation, the authors of [49] proposed a solution capable of targeting a full 128-bit leakage by leveraging the linear dependency between key *K* and input D_i under the HW leakage assumption in the AES's initial AddRoundKey, through Linear Regression. The interested reader is referred to [49], for further details.

The benefits of LRA applied to SCAs are remarkable. Not only is it an alternative to CPA, but it can also facilitate more effective attacks (i.e., lower number of traces are required for the real key to emerge). In [19], experiments were conducted against a real 8-bit target, to compare the effectiveness of CPA and LRA with respect to the first output of the AES S-box. It was concluded that, when the chosen leakage model exactly corresponded to the actual leakage function of the device (perfect model case), CPA and LRA results are identical. CPA is faster than LRA, as it requires less processing effort, which can be explained by the fact that LRA has to rebuild the model from data while CPA is directly provided with the optimal model function and uses the observations only to corroborate a linear dependency. However, when the model is unknown, LRA succeeds at revealing the correct key, contrary to CPA that becomes impractical. The authors conclude by stating that if one has a good linear approximation of the leakage function, CPA is an optimal way of performing the attack, elsewise LRA will always perform better than CPA.

3.1.2.5 Scatter

The vast majority of side-channel attacks require the analyzed traces to be perfectly aligned. However, this prerequisite is so often challenging in the practical realization of attacks. Some countermeasures to increase the difficulty of accomplishing SCAs successfully include the introduction of random jitter or the random execution of fake operations to cause the measured traces, associated with each execution, to be unsynchronized [46].

Scatter [16] has recently been proposed and attested with practical experiments by researchers of eShard and University of Limoges, as an attack with the potential to tackle most trace alignment issues, including random jittering and random order execution. Instead of analyzing the traces at each point in time, univariately, this technique integrates all samples within a window of interest and converts that data into their distribution depicting the number of times each value occurred. Recall that in a power trace, the samples are represented as a function of time, describing how the power changes over time. The core idea is that, instead of analyzing the samples in the time domain like for other PA attacks, in Scatter, the samples within a relevant time frame are organized and analyzed based on their frequency of occurrence (i.e., distribution of samples).

The attack procedure can be delineated as follows, with the first three steps defined as for DPA:

4. **Mapping intermediate values to corresponding hypothetical power consumption values, based on assumed leakage model**.

   Scatter's specification paper [16] mentions that the choice of the leakage model is not restrictive as the attack does not assume anything about the linearity of the model (i.e., it shall work for linear or non-linear leakages), similarly to MIA.

5. **Choosing a relevant window of interest and conversion into their respective distributions**.

   Considering T the batch of N recorded traces, the attacker starts by selecting a relevant window of interest $S_i = T_i[t : t + d]$ for i = 0, ..., N. The targeted intermediate value is believed to be processed within the (t, $t + d$) time frame, thus it's where the targeted leakage should occur. Thereafter, each window S_i is represented based on its frequency of occurrences, for all i = 0, ..., N. The premise is that the useful information counts the same wherever it stands in the window (i.e., the leakage is still present in the new representation). The window includes both leaking and non-leaking points. Figure 37 illustrates the outlined transformation.

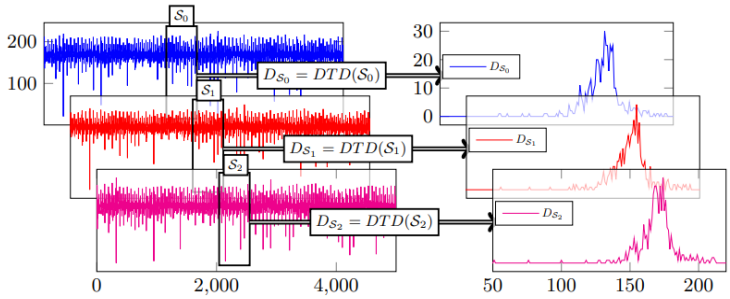

Figure 37 – Transformation of traces portions from time domain to corresponding distributions [17].

6. **Partitioning distributions and computing corresponding Probability Density Functions**.

   Scatter works by partitioning the distributions obtained from the previous step, for each possible key guess K', depending on the corresponding hypothetical power values $H_{i,K'}$ (e.g., if K is a key-byte and HW is the leakage model, the distributions would be partitioned among 256×9 possible partitions). Then, the PDF (Probability Density Function) associated with the distributions of each partition is calculated. To do so, the distributions are added on top of each other as they are included in the corresponding partition, and later normalized.

   The current and prior steps can be accomplished as in (31). This pythonic pseudo-code slightly differs from the one proposed in the original paper [16], however it conveys the same meaning. It's worth pointing out that the *histogram* function is a reference to the function from python's numpy library. Accordingly, the "density" argument set to "True" denotes the normalization of the resultant distribution which implies the computation of the PDF, while the choice of the number of bins "nbins" reportedly impacts the efficacy of the attack [51]: more bins generally lead to better results when considering a smaller window of interest (i.e., less non-leaking points are accounted for, meaning less noise), and less bins work better for wider windows of interest (i.e., more non-leaking points are contemplated meaning more noise).

   For the correct key guess, the PDFs converge towards an asymptotic distribution related to the nature of the signal, while the wrong key guesses follow a Gaussian distribution as they have not been partitioned against their real value [16]. Considering this behavior, it is possible to discriminate the PDF related to the correct key from all other PDFs.

```
# partition traces
for i in range(N):
      Si = Ti[t : t + d]
      for K' in range(256):
            partition[K'][Hi,K'].append(Si)

# compute distr of each partition and corresponding PDF
for i in range(N):
      for K' in range(256):
            hist, bins = histogram(partition[K'][Hi,K'],
            bins=nbins, density=True)
            pdf[K'][Hi,K'] = (hist, bins)
```
(31)

7. **Comparing PDFs using distinguisher to determine correct key guess**.

   To distinguish the *pdf* associated with the correct key guess from the remaining PDFs, the use of either one of two distinguishers is suggested: one based on the Pearson's chi-squared statistical test which expresses how much a distribution differs from a general distribution, and the other based on mutual information. For details on both distinguishers the reader is referred to the original paper [16]. In this paper, attacks were simulated and later conducted against an 8-bit AVR microcontroller running an AES-128 unprotected implementation, to attest and compare the efficacy of the documented strategy. In synthesis, the experiments' results demonstrated Scatter to be particularly effective for non-aligned traces. Both distinguishers were effective at revealing the correct key guess, however the mutual information -based distinguisher allowed for slightly better results than the Pearson's chi-squared, in terms of the required number of traces.

In circumstances where traces are reasonably well aligned, classical PA techniques such as CPA prove to be significantly more effective than Scatter [16]. Still, it is argued that Scatter holds genuine value in situations where the condition of alignment cannot be adequately fulfilled due to poor measurement quality or the implementation of shuffling countermeasures that, conversely, could render classical attacks unfeasible. However, researchers in [51] raised doubts on whether this novel technique was in fact an improvement over existing solutions, after conducting both simulated and real experiments under the conditions that were initially put forward by the Scatter authors. The results showed that a multivariate variant of the LRA outperformed Scatter under every tested condition. Particularly, the simulated experiments confirmed that the mutual information -based distinguished performed slightly better than the chi-squared -based distinguisher, yet for the experiments carried out against a different real target (i.e., Cortex-M4 chip) Scatter was unable to recover all 16 key bytes of an AES state while the variant of LRA succeeded at doing so.

3.1.3 Profile-based

In profile-based attacks, the adversary must possess and be in full control of a device identical to the one being targeted. Rather than assuming a leakage model *a priori*, the attacker makes use of the surrogate device to construct a "profile" that describes the behavior of the device for known random keys and inputs, and later uses it to match the observed power consumption of the target with the secret key that it is most likely associated with.

In essence, a profile-based attack can be delineated into two distinct phases:

I. In what's referred to as the *profiling phase*, the attacker acquires plenty of traces from the surrogate device while it is running the cryptographic algorithm for known random keys and known random input. Both the known keys and inputs are stored along with the corresponding traces, from which an association between the actual processed values and the observed power is established through statistics. In a simplistic way, this association can be regarded as a correspondence between power and processed data. Accordingly, in the profiling phase the characterization takes place.

II. In the *matching phase*, traces are obtained from the actual target device encrypting or decrypting known random data using the secret key, and these are compared against the "profiles" made from the surrogate device to ultimately deduce the target's secret key. Accordingly, in the matching phase, the characterization is used for an attack.

The major drawbacks of profile-based attacks, comparatively to model-based attacks, is the extent of preliminary setup and processing required to build an accurate profile (i.e., *profiling phase* is an additional time-consuming step of the attack), as well as the requirement of being in full control of an identical device to the one being targeted, which may not always be feasible or realistic.

Profile-based attacks aim to minimize the number of traces needed from the target device to successfully extract the secret key, which comes at the expense of the *profiling phase* and associated constraints. This is of special interest when the attacker is bounded by the number of measurements that can be obtained from the target device.

This category includes the Template attack, Stochastic model-based attack and, more recently, Machine Learning models for classification.

3.1.3.1 Template attack

Template attacks [52] were first introduced by *Pankaj et al.* in 2002, and were, at the time, believed to be the strongest form of side channel attack possible due to the fact that it leveraged all available information in each sample in a multivariate manner [52]. Particularly, in the profiling phase of the template attack, a precise multivariate characterization of the signal and noise, that is fully defined by a mean vector 'm' and a covariance matrix 'C', is attained through the analysis of the traces from the surrogate device, producing the so-called *templates* that consist of the pair (m, C) [6]. Unlike other attacks that propose the removal of noise by averaging the traces, the authors of [52] argue that, especially for cryptographic algorithms implemented in CMOS devices, the use of such characterization in remarkably advantageous to classify even a single sample. The relevance of considering noise is substantiated by the analogy that, in the context of signal communications, it's the accurate characterization of both signal and ambient noise that enables a receiver to successfully extract very weak signals.

Even though the attacker undertakes an extra level of processing to create the templates which, in practice, may require dozens of thousands of power traces from the surrogate device, the final matching phase should succeed for a much reduced number of traces from the target device.

In brief, the template attack can be described as follows (refer to [17] for implementation details):

1. Utilize the surrogate device to record a considerable number of traces by subjecting it to numerous random keys and inputs. Ensure that enough traces are collected in order to gather ample information regarding each subkey value.
2. Build the template from the acquired information. A template notes a few points of interest in the power traces and multivariate Gaussian distribution of the power traces at each point. By modeling a power trace as a multivariate normal distribution, we're able to take into account the correlation between neighboring points [6].

   To build the templates, traces corresponding to the same pair (*D*, *K*), where *D* and *K* are input and key, respectively, are grouped together. Then, the mean vector 'm' and the covariance matrix of the multivariate normal distribution 'C' are computed for each group to obtain a template (m, C) for every pair of data and key (*D*, *K*). Note that the (*D*, *K*) can be described as a function of the two values, i.e., intermediate value, such as the output of the S-box, or even as the hypothetical intermediate value of a surely suitable leakage model [6], as it will be elaborated further.
3. Record a small number of power traces from the target device executing the cryptographic algorithm for random known input and fixed secret key.
4. Match each trace from the target device against the template to limit the choices for the key to a small set and deduce the best key candidate.

   The matching is done by comparing the probability density function of the multivariate normal distribution of each template with the trace of the device under attack. In other words, for a trace T_i of the target device and for each template $(m, C)_{(D,K)}$, the probability that measures how well each template fits a given trace is computed through expression (32).

$$p\left(T_i; (m, C)_{(D,K)}\right) = \frac{\exp\left(-\frac{1}{2} \cdot (T_i - m)^T \cdot C^{-1} \cdot (T_i - m)\right)}{\sqrt{(2 \cdot \pi)^{\#m} \cdot determinant(C)}} \tag{32}$$

   Intuitively, the highest probability should indicate the correct template for T_i. After having computed the highest probability of each trace T_i for all i = 0, …, *N* traces, a maximum-likelihood decision rule is applied to the results, to ultimately determine the correct key [6]. In practice, the values calculated in the exponent tend to be very small which often results in numerical problems. As such, to avoid the exponentiation, one typically applies the logarithm to (32), evaluating $\ln(p(T_i; (m, C)_{(D,K)}))$ instead of $p(T_i; (m, C)_{(D,K)})$. For a more comprehensive explanations, readers are directed to reference [6].

As previously stated, in the template attack the power traces are characterized by a multivariate normal distribution. Since the goal is to create a multivariate probability describing the power traces for every possible key, if the entire power trace was modeled (with, say, 3000 samples), then the result would be a 3000-dimension distribution, as the size of the covariance matrix depends on the number of analyzed samples. For this reason, only points of interest (i.e., leakage points) are modeled. Section 3.2 provides insights on how POIs can be calculated prior to an attack. Another good strategy to reduce the necessary number of models is to only model a sensitive part of the cryptographic algorithm such as the S-box in AES. In addition, if a device

leaks according to the Hamming weight of the processed data, for instance processing a decimal value 1 will virtually lead to the same power consumption as processing the decimal value 2. Consequently, the template that is associated with the decimal value 1 will match to a trace in which the decimal value 1 is processed, but also to the template associated with the decimal value 2. This implies that it is irrelevant to build separate templates for values with identical Hamming weights [6]. Now, in such case, instead of building 256 templates for the AES's S-box output, we can simply build 9 templates, one for each Hamming weight of the S-box output. Typically, this is the preferred approach, where there's the trade-off between less memory to store the templates and less processing in both profiling and matching phases, at the cost of a higher number of traces being required to infer the correct key [17].

Generally, the leakage of an intermediate value is not present only in one sample, it is spread through the surrounding samples, given that the power is measured at a higher sampling rate than the device's clock speed. However, univariate SCAs analyze each point in time independently. Template attacks, however, exploit the multivariate leakage by considering a set of samples as a whole, which can prove beneficial, when the leakage is indeed spread, and the Gaussian assumption holds. Apparent downsides of the template attack include the requirement of having complete control over a surrogate device, as well as the need for a substantial quantity of traces to build each template. This need arises mainly for statistical reasons: in order to come up with a good distribution to model the power traces for every key, a large sum of traces is imperative for every key.

3.1.3.2 Stochastic attack

The Stochastic profile-based attack was first introduced in [2] by *Schindler et al.*, for the purpose of Power Analysis. The proposal aimed at achieving the efficiency of template attacks in the matching phase but with the requirement of far less measurements for the profiling phase. Given that LRA was primarily derived from the stochastic attack, the linear expression (26) still applies, as the essential idea behind stochastic models is to find a good set of coefficients that matches well the actual observed leakage of the processed values [18]. As for the stochastic attack, these coefficients are computed during the profiling phase based on the known processed values related to the known key and input and associated power consumption of the surrogate device. Unlike the template attack where, for each possible key-byte value, multiple traces needed to be acquired from the surrogate device, for the profiling phase of the stochastic attack only a total of N leakage traces from a uniform distribution of key bytes should suffice [18].

To summarize, in the profiling phase, the coefficients (α'_{-1}, α'_0, …, α'_{n-1}) are estimated for the known intermediate values based on the actual power consumption of the surrogate device under control. Recall that the intermediate values are known because the key K_i and input D_i, associated with trace T_i, are controlled by the attacker. One set of coefficients (α'_{-1}, α'_0, …, α'_{n-1}) is computed per point of interest and placed in a 'profile list' where each entry t represents the approximated leakage model of the device at instant t. So, the profiling phase of the stochastic attack consists of building a profile of the surrogate device, and this profile corresponds to the approximations of the leakage model at each point of interest t, that is estimated using the same methods described in Linear Regression Analysis' Section but based on the actual real processed values and associated power consumption.

At last, to extract the key from the traces of the actual target device, in the matching phase, for each key guess, the set of coefficients (α'_{-1}, α'_0, …, α'_{n-1}) is estimated based on the associated intermediate values and power traces from the target, for the same set of temporal points of interest considered in the profiling phase, and the resultant sets of coefficients are

compared against the profile computed in the profiling step. The closest match discloses the best key candidate.

According to [2], the stochastic attack exhibits comparable efficiency, in terms of key extraction, to that of the template attack. Nevertheless, the profiling phase of the stochastic attack comes with the advantage of requiring significantly fewer traces from the surrogate device to approximate a profile of the device that later allows extracting the secret key.

3.1.3.3 Machine Learning

More recently, machine learning models based on neural networks, commonly used for image classification tasks, have been proposed and applied in the context of side-channel analysis. Practical results from literature [20], on several datasets, have demonstrated the ability of these new profiled attacks to perform successful key recoveries, even outperforming the Template Attack when applied to highly noisy traces.

Researchers in [20] emphasized deep learning algorithms based on MLP (Multi-Layer Perceptron) networks and CNNs (Convolutional Neural Networks). One of the goals of [20]'s study was to discuss and validate several parametrization options of these models for distinct attack conditions such as noise and desynchronization. For the purpose of this dissertation, it is most pertinent to mention that the study showed that CNNs are almost as efficient as MLP networks in the context of perfectly synchronized observations, and that CNNs, which are more difficult to train, proved to be more efficient than MLPs in the presence of desynchronization/jitter.

Deep learning, and in particular deep-neural networks, are nowadays the privileged tool to address classification problems. A neural network has an input layer, an output layer, and all other layers are called hidden layers. The neural network training consists of an iterative approach that applies the stochastic Gradient Descent algorithm to minimize a loss function that quantifies the classification error of the function over a training set. One iteration over all the training sets during the stochastic gradient descent is called an epoch. The number of epochs is an important parameter to tune because small values may lead to under-fitting (i.e., the number of steps of the Gradient Descent is not sufficient and the model is too poor to capture a trend in the training dataset) and high values may lead to over-fitting (i.e., the model is too complex, it perfectly fits the training dataset, but is not able to generalize its predictions to other datasets). Two parameterized model architectures (i.e., MLP and CNN) are selected from [20]'s paper, based on their experimental results, and are made available through an open-source power analysis tool entitled "lascar" (see Section 3.5), to serve as a base ground for researchers to explore further parameterization options or network designs, for practical use.

In this profiled attack based on ML classifiers, the adversary acquires a large and diverse set of profiling traces, from a device identical to the target device, and uses this dataset to train the classification model, during the profiling/training phase. Then, he acquires power measurements from the actual target device while it is encrypting or decrypting data using the secret key and feeds these target traces into the trained model that classifies each key candidate K' with a confidence score. The highest confidence score determines the most likely correct key guess.

These models can process more data samples than the classical template attack; however, finding the optimal parameters and the most suitable architecture for a specific dataset is a particularly challenging task that heavily impacts the efficacy of the model for the attack.

3.2 Leakage assessment

The leakage assessment is generally performed before conducting any PA attack, as it allows to quantify the data-leakage at each point in time, in the power traces. By assessing the power leakage of the cryptographic execution of the device, adversaries can learn information about the underlying cryptographic implementation, particularly the manner in which data is handled, whether the power consumption exhibits detectable data-dependency and if so, where, in time, such leakage occurs, as well as gaining insights into the device's leakage model [6].

By leveraging this supplementary information, an attacker can engage in more targeted attacks. For instance, focusing on attacking only the samples where data-dependent leakage was detected, the so-called POIs (Points of Interest) or leakage points, to reduce both memory and processing requisites ultimately accelerating the attack, rather than targeting the entire trace. Recall that the instantaneous power consumption, described by each sample of the trace, depends on the processed data at that instant. Moreover, the intermediate values targeted for key extraction are not handled at all instants, which means that only a restricted set of samples is exploitable, the leakage points.

Leakage detection techniques rely on relatively simple metrics that are knowingly efficient and don't require vast prior knowledge regarding the target device or implementation.

When the adversary has control solely over the plaintext that is encrypted with AES, he's able to do a leakage assessment to identify where the plaintext bytes leak. In such case, depending on AES implementation and the leakage of the device, the adversary might be able to spot the leakage not only from the operation where the plaintext bytes are initially loaded from memory but also of other operations that depends on these bytes. Conversely, an attacker with complete control over the plaintext and the key (i.e., case of having a surrogate device running the same AES implementation as the target), has the ability of computing any intermediate value of the AES encryption and consequently assess where the power consumption depends on such value and so when it is most likely processed. Furthermore, it can serve to validate assumptions pertaining to the leakage model that can be determinant for the success of attacks.

3.2.1 Signal-to-Noise Ratio

The SNR (Signal-to-Noise Ratio) (33), generally defined as the ratio between the signal and the noise component of a measurement, is a commonly used metric in electrical engineering and signal processing.

$$SNR = \frac{Var(Signal)}{Var(Noise)} \tag{33}$$

In the context of PA, the signal corresponds to the component of the power consumption that is exploitable (i.e., that contains relevant information for an attacker in a given attack scenario), while the noise corresponds to every other component that holds no significance to the attack. Accordingly, the SNR of a power trace's point can be described as the ratio between the variance of the data-dependent component of the instantaneous power, and the variance of the remaining parts, with consideration to (2). So, the SNR quantifies how much exploitable information is leaking from a point of a power trace [6]. Hence, the higher the SNR, the higher the leakage.

$$SNR = \frac{Var(P_{data})}{Var(P_{instant} - P_{data})} \quad (34)$$

The variance of P_{data} quantifies how much a point of a power trace (i.e., $P_{instant}$) varies because of P_{data}. On the other hand, $Var(P_{instant} - P_{data})$ quantifies how much the power at a given instant varies due to power components unrelated to the processed data, which are regarded as noise in the context of a PA attack. Therefore, the higher the SNR, the easier it is to detect P_{data} from the noise [6]. Note that, as we're calculating the SNR at a given instant for properly synchronized traces, the instruction executed over all traces at that fixed instant is the same, thus the variance of $P_{operation}$ is zero.

The assumption that is made by the attacker, regarding how P_{data} is modeled (i.e., leakage model), when computing the SNR, highly influences SNR results that will be obtained from the assessment, since the variation of P_{data} is computed based on such assumption [6], meaning that the SNR can be leveraged to validate certain suppositions about the target.

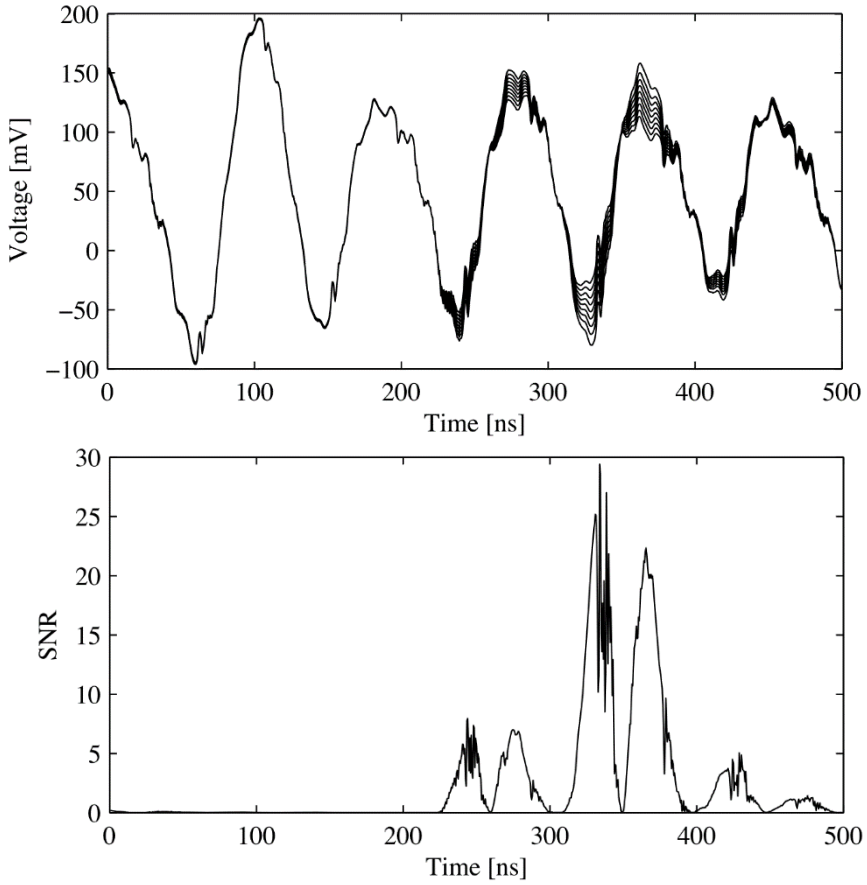

Figure 38 - Mean traces for the 9 distinct Hamming weights and corresponding Signal-to-Noise ratio [6].

The first plot of Figure 38 shows the power consumption of a microcontroller, in [6], while transferring uniformly distributed values from its internal memory to a register, whose P_{data} distribution was demonstrated to be consistent with the HW model. Accordingly, traces obtained from transferring a value with the same Hamming weight were averaged, and the 9 resultant traces are presented in the top plot, where the data fluctuations related to processing become evident. The SNR at each point in time was derived from the 9 traces and displayed in

the final plot of Figure 38. It clearly exposes the instants where power varies the most according to the Hamming weight of the processed data and thus where the leakage is most prevalent. The SNR provides a straightforward measure of the signal (i.e., P_{data} leakage) strength and is relatively easy to compute and interpret. In contrast to the Welch's *t*-test, the SNR doesn't provide insights into the significance of observed differences between traces associated with the processing of different values.

In summary, the SNR is a useful metric to assess the strength of the data-dependency at each instant, relatively to all other components of power (or EM field) that are considered noise to the attack, and thus detect the points of exploitable leakage.

Readers seeking implementation details about the computation of SNR in the context of PA are encouraged to consult the "lascar" [60] github repository.

3.2.2 Welch's *t*-test

A fundamental question in many different scientific fields is whether two sets of data are significantly different from each other. Statistically speaking, the most common approach to answer such a question is the Welch's *t*-test, which is an extension of the Student's *t*-test but for unequal sample sizes and variances, and whose aim is to provide a quantitative value as a probability that the mean μ of two sets are different [21]. In other words, a *t*-test determines whether a dataset provides sufficient evidence to reject the null hypothesis [53]. The null hypothesis is that the two sets of power traces have identical means and variance. A high positive or negative value of t_{test} at a point in time, indicates a high degree of confidence that the null hypothesis is incorrect.

In [53], the NIST standardization body established a leakage assessment methodology, based on the Welch's *t*-test, for side-channel resistance validation. This approach uses statistical hypothesis testing to detect whether a certain sensitive value (i.e., intermediate value) significantly influences the measurement data. Each trace is assigned to one of two groups based on its corresponding sensitive value. As the traces must be partitioned between either of two groups, the partitioning process aims to ensure that the sensitive values associated with the traces in one group differ significantly from the sensitive values in the opposite group [53]. For example, the traces can be partitioned depending on whether the Hamming weight of a plaintext-byte is greater than 4, or for instance based on the LSB of the plaintext-byte or other intermediate value. Alternatives of partitioning explored in the literature [54] include the *fixed vs. random* test where one of the classes has a fixed intermediate value and the other has random valued intermediate values instead (e.g., for a device running AES, acquire traces for the first S-box outputting 0 and traces for the S-box outputting a random value), and the *fixed vs. fixed* test where traces are captured for different fixed intermediate values (e.g., plaintext with all bytes set to 0x00 and plaintext with all bytes set to 0xFF). Several options for partitioning can be explored depending on the leakage we want to assess.

Let Q_0 and Q_1 indicate the two groups under test, μ_0 and μ_1 the sample mean, s_0^2 and s_1^2 the sample variance, and n_0 and n_1 the cardinality of the set Q_0 and Q_1, respectively. The *t*-test statistic is computed as (35).

$$t_{test} = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}} \tag{35}$$

For each point in the traces, the *t*-test is calculated. The points with the highest absolute *t*-test values are those where the mean value is the most different between the samples of the two groups at that instant, which is a clear evidence of leakage [37]. The *t*-test provides a statistical measure of the significance of differences between the two groups while taking into account the variability within each groups, allowing for more accurate comparisons.

In summary, the Welch's *t*-test is a viable option for comparing sets of observations, assessing the significance of their differences, and identifying potential points of leakage.

3.3 Pre-processing

Most PA attacks and leakage assessment techniques require the traces under analysis to be perfectly aligned. The traces are captured based on a trigger signal from the target device. However, in practice, having a reliable hardware trigger signal presents a significant challenge due to a variety of potential factors such as the source of the trigger (i.e., is it triggering on specific USB packets or was the device programmed to trigger right before the execution of the cryptographic algorithm), the stability of the clock, or the presence of countermeasures intended to induce misalignments. Also, noise induced, for example, by neighboring components can degrade the quality of the measurements and impact the effectiveness of the analysis. To tackle some of these issues, it may be worth exploring the adoption of pre-processing mechanisms.

3.3.1 Filtering

Filtering is mainly used to filter out undesirable or irrelevant frequencies from the measured power signals. Most devices have integrated filters inside their electronic circuit, to remove high-frequency or low-frequency noise that may affect the performance of certain components.

Filtering the traces for required frequencies can improve the SNR in terms of correlation [39]. To do so, after the acquisition has been performed, one should start by detecting the predominant frequencies in the signal, by computing the discrete Fourier Transform using the FFT algorithm and identifying the frequencies that correspond to the noise. The frequency at which the target operates is usually known, and if not, it can be measured.

There are four distinct categories of filtering algorithms, namely low-pass, high-pass, band-pass, and band-stop filters. They should be selected depending on the intended outcome: low-pass filters remove frequencies above a specified threshold and only allow frequencies below such threshold (i.e., lower frequencies) to pass, thus they can be used to cut high-frequency noise. On the contrary, high-pass filters remove, from the signal, frequencies below a specified threshold, letting frequencies above such threshold to pass. Band-stop filters are used to remove a specific frequency, while band-pass filters let frequencies within a certain range to pass and reject/attenuate frequencies outside such range. Many algorithms exist for each type of filter. Examples are the Butterworth filter that can be used for all four mentioned types, or the second-order IIR band-stop filter to reject a narrow frequency band while minimally affecting the remaining spectrum.

Trace "compression", either performed by adding together successive measurements or by averaging, is said to help reduce high-frequency and measurement noise, and amplify signal resolution [11].

3.3.2 Synchronization

The proper alignment of traces is crucial for most Power Analysis techniques. However, in practice, capturing traces for which samples at instant *t* correspond precisely to the same moment for all traces is not always feasible. As such, a collection of synchronization mechanisms exists, to enable the alignment of traces after the traces have been gathered.

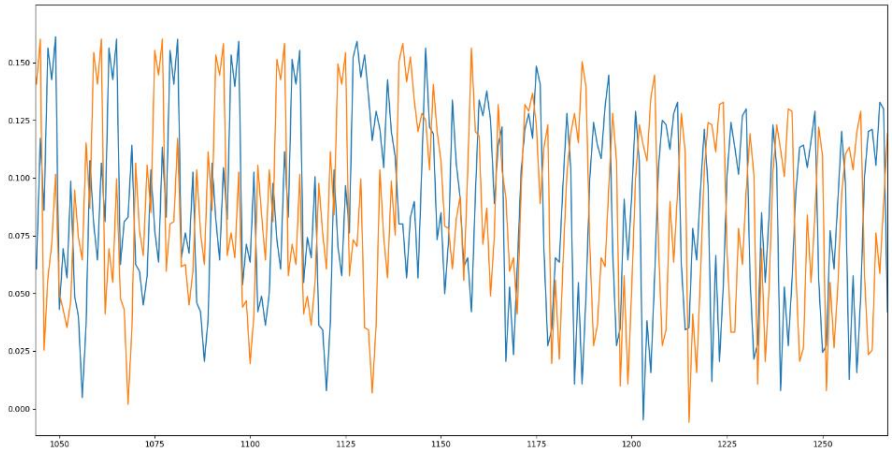

Figure 39 – Desynchronization between two power traces.

Figure 39 portrays the temporal window, from sample 1025 to sample 1300, of two power traces acquired from the execution of an AES-128 encryption algorithm. In the plot, the evident power consumption pattern enables a clear identification of the present desynchronization between the two traces. To synchronize the orange trace, using the blue trace as a reference, we would have to calculate the desynchronization offset between both traces and apply that offset to the orange trace by shifting it - in this case, to the right - so that the trace pattern would fit perfectly on top of the reference trace.

The following subsections analyze the various synchronization algorithms that can be used for the purpose of computing such synchronization offset. Particularly, in DTW, a distinct offset can be applied per sample, rather than having a single offset value applied to all samples of the trace.

3.3.2.1 Sum-of-Differences

The synchronization mechanism based on the sum of differences relies on a sliding window approach. Accordingly, a sliding range is specified, and a reference window is selected from a trace belonging to the dataset. Then, for each trace in the dataset, we slide a window within the specified sliding range and subtract the current sliding window to the chosen reference window. The result is an array of differences between samples of the two windows. Finally, we sum the absolute values of this array and obtain a scalar value referred to as the sum-of-differences value. The offset of the sliding window that maximizes the sum-of-differences value is added to the current trace, in order to horizontally align him with the reference trace. For all traces, this operation is performed, resulting in the synchronization of each trace relative to the reference trace. The synchronization algorithm can be described by (36).

```
for trace in traces:
      max_sum = 0
      sync_offset = 0
      #find best sliding win between [start:end] range
      for i in range(start,end-window_size+1):
            curr_window = trace[i:i+window_size]
            abs_diff = abs(reference_window - curr_window)
            sum_of_diffs = sum(abs_diff)
            if sum_of_diffs > max_sum:
                  max_sum = sum_of_diffs
                  sync_offset = reference_window_start - i
      apply_offset(trace, sync_offset)
```

(36)

Figure 40 illustrates the sliding window approach. The red straight arrow line, below the trace, represents the sliding window range, while the red box represents the current sliding window, and the blue double-sided arrow line represents the comparison that is made between sliding window and reference window.

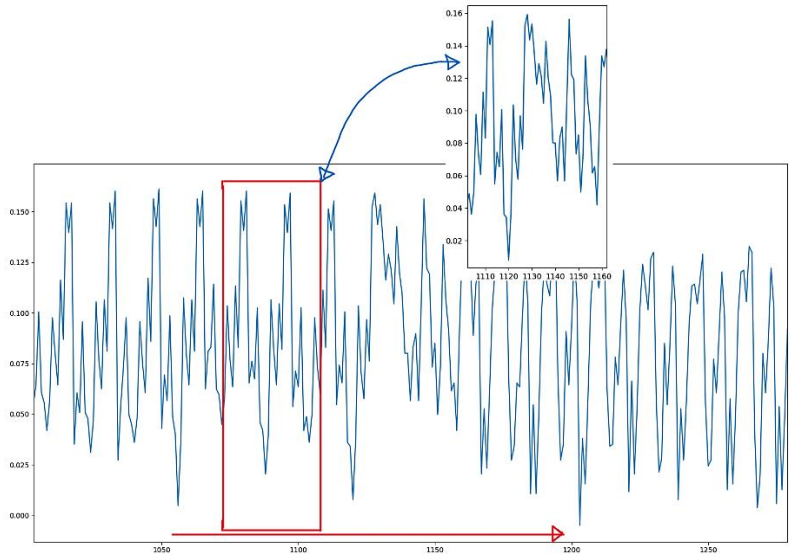

Figure 40 – Sliding window synchronization approach.

3.3.2.2 Pearson's correlation coefficient

The synchronization of traces can be achieved by employing the same sliding window approach portrayed in Figure 40, but rather than determining the offset of the sliding window that maximizes the sum-of-differences value as for the previously presented mechanism, the offset of the sliding window that maximizes the Pearson's correlation coefficient between sliding window and reference window is considered, as in (37).

```
for trace in traces:
      max_correlation = 0
      sync_offset = 0
      #find best sliding win between [start:end] range
      for i in range(start,end-window_size+1):
            curr_win = trace[i:i+window_size]
            correlation=pearson_corr_coef(reference_win,curr_win)
            if correlation > max_correlation:
                  max_correlation = correlation
                  sync_offset = reference_win_start - i
      apply_offset(trace, sync_offset)
```
(37)

3.3.2.3 Cross-correlation

In the field of signal processing, cross-correlation serves as a metric to assess the similarity between two series, considering the displacement of one series relative to the other. Therefore, it can be used to compare two time series and objectively determine how well they match up with each other, including the point at which the best match occurs.

The efficient computation of the time-delay offset between two time series can be achieved by using a fast Fourier Transform-based implementation of cross-correlation, instead of the previously covered sliding window approach. As such, this synchronization mechanism can be especially advantageous for synchronizing a significant number of power traces. Traces are still synchronized relatively to a reference window of a reference trace.

3.3.2.4 Dynamic Time Warping

In time series analysis, dynamic time warping (DTW) is an algorithm for measuring the similarity between two temporal sequences that may vary in speed. In the case of power traces, saying that the trace varies in speed means that the instructions executed by the processor took a varying amount of time (i.e., clock cycles), which can render it unfeasible to synchronize.

The goal of DTW is to provide an optimal global alignment between two time series, exploiting temporal distortions between them, allowing to determine how similar the two signals are and which points in one signal correspond to the points on the other signal. From this mapping, an alignment cost for each point is computed and applied to the trace that is being synchronized, which eventually leads to the distortion of the trace, contrary to what happened with the previous methods where a single offset was applied to the whole trace. DTW is considered an elastic alignment method due to the distortion that can result from its application. This distortion can either turn out to be unfavorable for power analysis, or beneficial, depending on the dataset. In the presence of unstable clocks, DTW should be considered and tested as a possible workaround to counter the problem. An efficient implementation of this algorithm, referred to as FastDTW, was proposed by Salvador S. and Chan P. in [23].

3.4 Countermeasures

In this section, possible countermeasures to prevent or mitigate side-channel attacks leveraging power or electromagnetic leakage analysis are presented and briefly discussed. To address the primary objective of this thesis, particular focus is directed towards the application of such countermeasures for the purpose of securing AES-based firmware encryption.

Numerous countermeasures against SCAs have been researched and proposed. However, in the present state-of-the-art, none of the existing techniques can guarantee perfect security [46]. Protecting implementations against SCAs generally consists of increasing the difficulty of the attacks, making them more challenging to execute. In this context, the implementation cost of countermeasures and associated degradation of performance, is of great importance and should be evaluated with regards to the additional security obtained. Delving into the exhaustive list of all possible solutions to protect cryptographic devices against side-channel adversaries would, by itself, require an extensive survey, which falls outside the scope of this thesis. Accordingly, the goal of this section is to provide an intuitive understanding of the existing countermeasures, including their security foundations and objectives, and how security can be reasoned about and incorporated at different levels of abstraction [46].

3.4.1 Physical level

At the physical level, measures like EM shields or conforming glues to difficult modifications to the PCB are mentioned in [46]. Additionally, countermeasures at the circuit level based on electronic filters or in the usage of voltage regulators, zener diodes, operational amplifiers, or others, have been explored in the literature [23]. However, these have been demonstrated to be ineffective at rendering SCAs infeasible. Although they may introduce an additional layer of complexity to the attack, they can typically be overcome by the removal of the associated physical components.

When referring to circuit-level countermeasures, these can also include changes to the design of the IC (integrated circuit) comprising the hardware implementation of the cryptographic algorithm, with the aim of enhancing its resistance to SCAs. Since these are adjustments to the way the cryptographic algorithm is implemented, they can also be categorized as security at the algorithmic level.

3.4.2 Technological level

At the technological level (i.e., at the level of the gates used in the physical circuit), dynamic and differential logic styles have been proposed (e.g., [55]) as an alternative to static CMOS gates, to minimize the data-dependencies of the power consumption. In essence, to break this relation at the gate level, two different mechanisms are widely used: hiding and masking techniques [56].

Hiding attempts to have the same power consumption at the gate, circuit, or algorithm level, regardless of the data being processed. Ways to achieve data-independent power consumption at a gate-level include having differential gates where the true and complemented outputs are simultaneously generated, which implies full symmetry. However, in practice, the goal of hiding can only be achieved to a certain degree [6].

In masking, critical data is masked with random data such that operations on the masked data are indistinguishable from random data. At gate-level, this consists of computing both the inputs and the mask inside the gate itself. The simplest way to perform masking is through Boolean masking, where data is masked by being XORed with a random value. Arithmetic

masking involves more complex arithmetic operations within specific algorithms. As such, Boolean masking is preferably used at the gate-level, while at the algorithmic or circuit level, the use of dedicated arithmetic masking techniques that best suit the algorithm are recommended [56]. In [6], examples of masked logic styles such as the masked dual-rail precharge logic are mentioned.

In [56], countermeasures applied at gate level, based on hiding and/or masking, are compared against PA attacks. These are mainly compared, in a highly pertinent manner for cryptodesigners, based on cost, performance (delay and power), and estimated security level. In contrast to Section 3.4.1, tampering with the device in order to cancel the countermeasure is impossible.

Attacks on devices protected with masking or hiding countermeasures are still possible and have been thoroughly discussed across multiple publications (e.g., [6][57]). However, in most cases, attacks against devices protected by these countermeasures require significantly more effort than attacks on unprotected devices [6].

3.4.3 Algorithmic level

A variety of countermeasures can be employed at the algorithmic level. Time randomization through fake cycles insertion, unstable clocking, random instruction injection, or other random delays can be employed for the purpose of introducing desynchronization so that traces within the same acquisition set no longer align [10], random shuffling, random precharge of bus, the encryption of the buses, masking, or the addition of noise, are usual countermeasures [46].

Countermeasures at the algorithmic level can either be implemented for protecting software-based or hardware-based implementations of a cryptographic algorithm. Integrating countermeasure mechanisms in software-based implementations is relatively simple since these are designed and coded in programming languages such as Assembly, C, or Java, to be executed on a general-purpose CPU, and therefore don't require any additional hardware modifications nor cost [23]. The same applies for FPGAs since they are programable. On the other hand, ASIC-based implementations require the actual underlying hardware to be modified which can incur significant costs.

For all these measures that are based on some kind of randomness, the quality of the randomness source is of great importance [23]. The time taken for a certain instruction can often be calculated using the clock frequency of the device, therefore if the attacker knows the random number sequence determined by the device, he can bypass the countermeasure. Random numbers in software are deterministically generated using pseudo-random generators depending on a seed value. If the seed is constant (i.e., hard coded), the generated random number sequence will always be the same for a given pseudo-random algorithm. As such, [23] suggests the use of a true hardware random generator to obtain the seed to be fed to the pseudo-random generation.

Creating a block cipher implementation that verifiably resists all side-channel attacks in all usage scenarios poses a considerable challenge. Some countermeasures introduce substantial overheads in terms of gates, code-size, and performance. In addition, it is difficult to be certain that advanced side-channel analysis techniques, such as high-order PA, will not recover enough information to deduce the key [58]. Although no countermeasure alone can provide absolute security against statistical analysis, they do increase the amount of effort and level of expertise required to achieve an attack [14]. Combining defenses by implementing at least two of these

countermeasures (i.e., hiding and masking), prove to be very efficient [56] and practically dissuasive [14].

3.4.3.1 Hiding

The methods that consist of degrading the side-channel SNR to "hide" the internal activity of the chip are considered hiding techniques. These can include the addition of noise in amplitude due to, for example, parallel processes or peripherals, or in timing such as the insertion of dummy operations, shuffling, and other time randomization techniques [7]. The addition of noise can be implemented at both circuit level, technological level, or algorithmic level [46].

The shuffling countermeasure consists of shuffling the order in which an operation happens for each execution of the algorithm and has a similar effect to the one of time randomization techniques. In the context of AES, randomly shuffling the execution of the S-box lookup operations has been experimented in [6] and [23]. The idea is to, rather than doing an S-box lookup for each byte of the state in a fixed sequential way, for each cryptographic run the sequence from which the bytes all looked-up in the S-box table is randomly shuffled. For instance, in the smart card industry, it is common to see up to 15 fake executions of an algorithm along with a genuine one hidden amongst them, which results in a considerable increase of noise at the cost of a severe performance drawback [36]. *Mangard et al.* [6] mathematically proved that attacks against misaligned traces due to countermeasures based on time randomization would still succeed for a sufficiently large number of traces. Again, for a sufficiently large set of traces, [23] demonstrated that in practice CPA sufficed at extracting the key for the S-box lookup shuffling implementation. In certain cases, the effect of these countermeasures can be corrected by applying appropriate signal processing [14]. Also, the Scatter attack is presented as a potential solution against implementations whose supposed resistance to SCAs is based on these types of induced misalignment.

In the context of firmware encryption using AES-CBC, [36] suggests that the sequence of block decryptions can be shuffled randomly without any performance loss. Doing so, the attacker is not able to tell which block is being handled at which time. Given the length of a firmware image and assuming that the attacker is not able to recover the sequence by which the AES blocks are decrypted, the leakage becomes significantly diluted within the entire decryption process. However, if an attacker has control over the firmware blocks that are decrypted by the device, this countermeasure becomes irrelevant since he's able to send one single 16-byte block to the device (as if it was the full encrypted firmware image), and consequently capture one trace per firmware flashing request.

3.4.3.2 Masking

Data can also be ciphered dynamically during a process by hardware (e.g., bus encryption) or by software means (e.g., data masking with random values), so that the handled variables become unpredictable and consequently no correlation can be expected anymore [14].

In a masked implementation, each intermediate value v is concealed by a random value m that is referred to as the mask: $v_m = v * m$. The mask m is generated internally, unknown to the attacker, and varies from execution to execution. The goal is to decorrelate the processed data from the power consumption. The operation * is typically defined according to the operations that are used in the cryptographic algorithm, hence it is most often the XOR, the modular addition, or the modular multiplication. Commonly, masks are directly applied to the plaintext or the key. The implementation of the cryptographic algorithm needs to be slightly

changed in order to process the masked intermediate values and in order to keep trace of the masks since the masks need to be removed at the end of the computation, to obtain the final ciphertext [6]. Be aware that effective masking implementations are not trivial and badly implemented masking schemes may lead to a false sense of security [7]. For details and examples of both software and hardware-based masked AES implementations, as well as considerations on bus encryption, random precharging, or masked table lookups, the reader is encouraged to consult [6].

In theory, sophisticated attacks such as high-order analysis can overcome the data masking method, however, employing masking together with hiding countermeasures can provide a greater level of security [14]. High-order attacks exploit joint leakages of the intermediate variables and require a pre-processing of the traces, using a combining function over distinct pairs of samples along the leakage trace. The pre-processing goal is to catch the pairs where both the masks and the masked variables leak. Two main pre-processing methods that have been proposed and employed in the literature (e.g., [57]) include the "absolute difference" (38) and the "product combining" (39). Due to the computational expense associated with such pre-processing, first determining the sample points where the mask and masked values leak and then applying the pre-processing only to these points is of utmost importance [57].

$$f(x, y) = |x - y| \tag{38}$$

$$f(x, y) = x \times y \tag{39}$$

Studies have been carried out on the security of masked bitsliced implementations of AES [57]. Bitsliced implementations involve decompositions of the algorithm into bit-level instruction, using logic gate operations (e.g., AND, OR, NOT, NAND, NOR, XOR). Instead of storing the values in the AES state as whole bytes, the bits of the state are spread across multiple registers. Even though these implementation types offer resistance against cached timing attacks they are still vulnerable to SCAs like PA if no countermeasures are taken [57].

3.4.4 Protocol level

Countermeasures can also be implemented at the protocol level. Within the context of firmware encryption, the underlying cryptographic algorithm (e.g., AES-128) is just one of several aspects that contribute to the overall process. The integration of block cipher modes of operation, the size of the firmware image, or the way the firmware procedure is implemented, all impact the approach that an attacker must undertake to break firmware encryption through side-channel analysis. For instance, in the hypothetical case of a device whose firmware cannot be tampered, holding a 512 KB encrypted firmware image that is decrypted using an AES-128 secret key *K*, the attacker may be limited by the amount of distinct blocks (i.e., (512*1024)/16 = 32768) that he can leverage for PA. In contrast, if the attacker is able to tamper the encrypted firmware image (i.e., a), in Section 2.1) and have the device decrypting it using *K*, he faces no limitations regarding the number of decryptions for distinct blocks that he can observe and exploit for PA. Considering this line of thought, it is wise to deliberate on protective countermeasures at the protocol level.

For PA attacks to be effective, the attacker must monitor many block encryption or decryption operations with the same secret key. If a single encryption key was used to encrypt only a few data blocks, then statistical attacks become impractical [58]. In fact, following secure key management guidelines, such as avoiding embedding master keys in end devices or

frequently updating keys are common protocol-level countermeasures [11][36][58]. This can consist in limiting the key usage to a single firmware by using a firmware-dependent key derivation algorithm seeded for example on a firmware version number, as suggested by [36], or updating the key after every few block encryptions for example by using a hash function as suggested by [58]. Under the crucial assumption that the attacker is not able to trick the device into decrypting arbitrary input, the goal of the first approach is to reduce the scope of the attack (i.e., retrieving the key would only affect a single firmware version), while the second approach aims at limiting the amount of useful information that an attacker can obtain about any particular key (i.e., each key is used so little that it is no longer possible to extract by SCA). Yet, protocols must be designed with care and expertise; otherwise, there is a risk of introducing new attack vectors, for example on the key derivation algorithm [36].

The idea of limiting the number of transactions that can be performed with any given key was introduced and strongly supported by *Kocher et al.* in [11], either through the implementation of key update procedures (i.e., dynamic re-keying protocols, in [36]) or the use of key derivation algorithms, to reduce the amount of data processed per key.

In [58], *Pankaj Rohatgi* wrote a design note where he defined a protocol-level approach to secure data encryption and decryption operations that can use existing unprotected hardware block cipher implementations while achieving provable security against SCAs. Given the crucial necessity of obstructing the attacker's control over the ciphertext, the author proposes both a leakage-resistant message authentication construction to enable the decrypting device to authenticate each ciphertext block before decrypting it, and a leakage-resistant decryption procedure for block ciphers. To put it briefly, the idea is to have the firmware encrypted by the manufacturer using an ordinary block cipher chaining mode but in such a way that a different key is used to encrypt every *nblocks* of plaintext. The choice of the key update frequency (*nblocks*) depends on the security and performance penalty trace-off. Each key is generated based on a proposed key derivation algorithm that relies on hashing the previously used key with a fixed constant to generate the next key. In the context of securing firmware encryption with AES, in a company that holds a range of distinct products, a different key per device model would be generated based on a secret, for example unique per group of products, managed in a so-called "key trees" construct. Note that it is impractical for a company to store a unique key per device, for key management reasons and because it would require the company to produce a unique encrypted firmware image per device every time it launches a new firmware update. At the moment of decrypting the firmware to be executed, the device uses its embedded key to generate the initial key, it uses it to decrypt the first *nblocks*, then it generates the second key from the initial key, to decrypt the following *nblocks*, and so on. The implementation is efficient, can be applied to various use-cases (i.e., loading encrypted firmware, secure storage, message exchange, and others) and has low overhead associated to it, making it an excellent reference to understand how security at the protocol-level can be reasoned and achieved. For further details, the reader is referred to [58].

Protocol level countermeasures are recognized as an effective and simple way of addressing side-channel exposure on less than perfect hardware [11].

3.5 Tools

A range of open-source tools are available for power analysis, leakage assessment, and pre-processing, but as expected, none of them implements all addressed techniques. Some have clear advantages over the others, such as concurrent implementations, optimizations, ease of use, or increased functionality. The reader is advised to explore the tools mentioned below, and

depending on his use-case, either use the most suitable tools independently or build his own with the possibility of integrating the components of interest from these tools into his own tool and customizing or adding extra functionality from scratch. All tools are quite different and require the user to read the available documentation and get comfortable with it.

a) *lascar*

Ledger's Advanced Side-Channel Analysis Repository is a python3 library, available at [60]. It contains classes and functions to easily conduct pre-processing, leakage assessment, and power analysis attacks. In brief, it has optimized implementations for DPA and CPA attacks, for leakage assessment with t-test and SNR. It's relatively easy to use, it is an actively maintained repository and it is flexible to be integrated with other tools. Many tutorials and examples are made available in its repository.

b) *qscat*

Qt Side-Channel Analysis tool is an intuitive GUI tool for conducting Correlation Power Analysis on AES, with various target options of intermediate values (e.g., output of the first SubBytes operation). Available at [61], "qscat" is a valuable tool for visually inspecting traces, supporting the Sum-of-Differences synchronization mechanism, which benefits from the great visualization feature.

c) ChipWhisperer library

This python3 library, available at [62], has a lot of functionality, from trace acquisition, to flashing target devices, to side-channel power analysis and glitching attacks. It is well documented but given the fact that it has so many available functionalities it may be more challenging to master than "lascar" when the only purpose is to analyze already acquired traces, but on the other hand, the fact that it supports trace acquisition might be perceived as an advantage to eventually have most functionality integrated in a single tool. The CW (ChipWhisperer) library toolchain was developed to facilitate the interaction with the CW's hardware.

d) scared

Similarly to *lascar*, the "scared" framework is a rich option for side-channel power analysis. It implements filtering, synchronization, leakage assessment, and power analysis mechanisms. It's available at [63].

e) side-channel-analysis-toolbox

This repository, developed by members of the Artificial Intelligence and Security Lab at TU Delf, contains the direct implementation of some power analysis attacks and leakage assessment techniques, based on purposed algorithms from state-of-the-art literature. Made available at [64].

To conduct the experimental part of the work proposed in this document a custom Python tool was developed for the purpose of handling large amounts of measurements data, conducting power analysis attacks, leakage assessment methodologies, and pre-processing of power traces. In this tool, support for the CPA, SNR, t-test, and Machine Learning classifier was based on "lascar" python classes. Functionality missing in the previously mentioned tools, such as the Scatter PA attack, were implemented based on the available literature. The "qscat" tool was useful for visualization and multi-step synchronization of traces. The "chipwhisperer"

python library was used to instrument and interact with the ChipWhisperer hardware device for trace acquisition.

3.6 Conclusions

In this chapter, state of the art Power Analysis techniques have been presented and compared. From the reviewed literature, it became evident that SPA is not an effective approach for attacking AES implementations. However, it remains valuable for identifying patterns associated with cryptographic operations, within power or EM field measurements. The CPA attack was introduced as an advancement over the precedent DPA. It requires a smaller number of traces than DPA to extract a key, as it allows to employ leakage models that are more aligned with the actual behavior of the target device contrary to DPA whose statistical distinguisher only supports binary leakage models. MIA employs a mutual information distinguisher to compare the real power consumption or EM emission of the target with the hypothetical power values of each key candidate. LRA allows to better model the data-dependent leakage of the device by employing a statistical approximation method thereby enabling to better exploit the actual leakage. Scatter is considered pertinent in cases of misaligned traces.

Profile-based attacks, which rely on the characterization of the device's power consumption, are then presented. These require the attacker to be in full control of a device identical to the one being targeted, in order to derive the so-called profiles that are used to determine the correspondence between keys and observed power.

Leakage assessment techniques provide a means to quantify the leakage associated to data processing at each moment in time. They allow to identify leakage points and consequently to conduct a more targeted attack. Conducting attacks in practice poses considerable challenges. In certain cases, pre-processing methods can be employed to address noise or the desynchronization of traces.

Finally, countermeasures are introduced and discussed at the physical, technological, algorithmic, and protocol levels. It is acknowledged that no approach offers absolute protection against SCAs. However, by combining countermeasures, better security guarantees can be achieved. Evaluating the security of side-channel resistant implementations is not a straightforward task. Caution should be exercised during the implementation or design of protective countermeasures, as certain decisions can inadvertently open new attack paths.

The existing literature on these types of attacks and countermeasures is widely accessible and conveyed in a clear language that facilitates a comprehensive understanding of the subject matter.

Obtaining a solid background knowledge and a clear understanding of the current state of the art was essential for carrying out the experimental work presented in the following chapter and justifying some of the decisions made throughout the process.

In summary, the *t*-test is useful for comparing well-defined groups and determining significant differences in means, while the SNR provides a general measure of signal strength relative to noise without relying on specific groupings. To assess the leakage, the SNR was preferred over the *t*-test with the aim of reducing acquisition time, as it can be applied to the datasets that are analyzed during attack. Since the final experimental setup allowed the acquisition of properly synchronized traces, no pre-processing mechanisms were necessary for obtaining the results presented in this document. Given the quality of the measurements and detected leakage, the CPA, LRA, and MIA attacks were thought to be advantageous due to their

traits and differentiating aspects documented in the literature, such as supported leakage model assumptions and characteristics of the analyzed dataset (i.e., synchronization requirement). For instance, DPA was overlooked due to CPA being portrayed as an advancement over it. The Template and ML classifier attacks were conducted with the aim of attesting the projected reduction in the minimum number of traces necessary for extracting the secret key bytes. Since profiled attacks are more demanding in terms of resources, they were directed towards exploiting known leakage points. No experiments were performed for the Stochastic attack due to the claimed superiority of LRA.

Chapter 4
Experimental Setup and Evaluation

This chapter serves to elaborate on the experimental work conducted during my internship at *Logitech Europe S.A.*, as well as to present and discuss the achieved outcomes.

In line with the motivation for this thesis, the task of assessing the feasibility of breaking firmware encryption through side-channel analysis, in a specific device, was assigned. To accomplish it, the emphasis was placed on demonstrating how the power consumption of the target device exposed its cryptographic secrets to attack.

Accordingly, and given that the device is a System-on-Chip comprised of an ARM-Cortex M4 general-purpose CPU and of a dedicated hardware co-processor for AES-128 ECB encryption, the interest was directed towards targeting both software- and hardware-based implementations of AES-128 running on the device. For this purpose, PA attacks were conducted considering two distinct adversarial models: the case where an adversary only has control over the plaintext, thus he is limited to non-profiled attacks, and the case where an adversary has full control over a surrogate device, thus he can acquire traces for random plaintexts and keys and subsequently construct a profiled attack.

Showing that key extraction is possible against both implementations, through PA attacks, is not only relevant for understanding whether firmware encryption is vulnerable but also to demonstrate that the device is most likely not suitable for cryptographic applications that lack proper protective countermeasures.

As mentioned in the previous conclusion, the attacks executed against the target device, as part of the experimental evaluation, include the CPA, LRA and MIA, along with the Template and the ML classifier-based attacks. Prior to their evaluation, the leakage assessment results, based on the computation of the SNR, are analyzed.

Ultimately, protective countermeasures for the purpose of securing firmware encryption on the targeted device are discussed.

4.1 Target device

The target device is a compact yet advanced SoC that is well-suited for a wide range of applications, including smart home devices, industrial IoT sensors, fitness trackers, wireless payment devices, smart city infrastructure, and more.

The SoC is composed of a 32-bit ARM-Cortex M4 general-purpose CPU that runs at a clock speed of 64 MHz, and of a proprietary ASIC hardware cryptographic accelerator supporting AES-128 ECB block encryption (encryption only, not decryption), that is controlled by the CPU and can be used to implement other block cipher modes of operation. While the SoC is designed to receive a 32 MHz clock input, the operational clock speed of the AES crypto accelerator is not disclosed. However, subsequently, by considering the execution time of the hardware-based implementation specified by the manufacturer, and both the sampling rate and

number of samples captured between the pin trigger, this peripheral was found to operate at a clock speed of 32 MHz.

Recall that software-based implementations are executed in the 32-bit ARM processor, while the dedicated hardware accelerator executes an efficient hardware-based implementation.

Given the objective of determining whether the SoC's processing units are secure against state-of-art PA attacks, in this experimental setting it was possible to program the SoC's ARM-Cortex CPU with any desired custom firmware, through a connection to the device's debug pins. Note that, even though the CPU could be programed to execute any software-based implementation, no modification could be realized to the proprietary ASIC-based implementation that is executed in the SoC's dedicated AES-128 cryptographic accelerator, since it is implemented in hardware. Also, no information is available regarding the underlying implementation of the AES-128 hardware-accelerated block encryption algorithm. Moreover, the device manufacturer does not provide any details regarding the power management of the SoC or the circuit power lines responsible for powering the CPU or the crypto accelerator. Such information would greatly facilitate practical PA attacks as it would offer valuable insights on how to measure the power consumption associated with each processing unit.

It is worth noting that *Logitech* is not a SoC's manufacturer, and that the manufacturer of the SoC covered in the present Chapter does not claim any security protection against physical attacks.

All devices that are physically available to attackers can be targeted with Power Analysis. The device under attack in this work is of special interest given the range of applications it can be applied to, and because it has a modern architecture. Demonstrating that the processing hardware of the device (i.e., both CPU and crypto accelerator) exposes the device to key extraction attacks through side-channel information, raises awareness for the fact that widely deployed modern devices today are still vulnerable to this type of attacks, and that their false sense of security still relies on the idea that the underlying cryptographic algorithm is strong, overlooking the significance of the executing hardware.

4.2 Preparation

Prior to devoting efforts on attacking the target device, the available tools were explored and experimented with on a publicly available dataset of simulated power traces with distinct characteristics (e.g., time desynchronization).

Afterwards, a Python-based tool was developed, leveraging existing functionality from the available tools, and with additional support for all state of the art attacks, leakage assessment, and pre-processing techniques covered in Chapter 3. Furthermore, methods for the processing and analysis of data, from persistent memory, were implemented. The tool's features were initially tested against the previously mentioned dataset, and fixes and enhancements were actively implemented based on practical usage and identified areas for improvement. Given the fact that computers have limited resources, it is quite important to consider and come up with solutions that cope with such constraints. For example, the set of traces and corresponding inputs (e.g., plaintexts) that are analyzed in a PA attack take a considerable amount of memory, thus the processing of this data must be efficiently implemented to avoid exhausting resources and time. The execution time of the attacks depends significantly on the number of traces and samples being analyzed. Therefore, the attacks that were coded from scratch, based on the studied literature, were implemented in such a way that leveraged the multi-core architecture

of the computer's processor running the attacks, so that different tasks could be done concurrently.

Initially, the plan involved performing key extraction via PA, against an AES-128 software-based implementation running on the device's ARM processor, and then against the hardware-based implementation of the dedicated crypto accelerator, by measuring the power from the circuit. However, through leakage assessment it became clear that the power measured from the circuit was hard to leverage for attacking the crypto-processor peripheral, as elaborated further in Section 4.4.1. Consequently, the alternative of measuring and exploiting the near-field EM emanation of the crypto accelerator was contemplated.

To carry out the actual attacks, a significant number of traces must be acquired, with each trace corresponding to the encryption (or decryption) of a known block of data. As such, the device was flashed with an interrupt-based piece of custom firmware, written in C Language, that allowed to set encryption parameters (i.e., plaintext, key) and to choose between the execution of the software or hardware-based implementation of AES-128. The program operated according to the following logic:

1) The program waits until one of three possible byte values (i.e., 'm', 'k', or 'p') is received in the Rx pin, via UART:
   - To set the encryption mode: byte 'm' followed by a byte valued decimal 0 (for hardware-based AES-128) or 1 (for software-based AES-128);
   - To set the encryption key: byte 'k' followed by the key's 16 bytes;
   - To set the plaintext block: byte 'p' followed by the plaintext's 16 bytes.
2) When the plaintext is set, the program triggers a specific pin of the device's circuit to HIGH (e.g., 3.3V), and executes the selected encryption algorithm for the plaintext bytes using the previously set key. After finishing the encryption run, the specific pin is set to LOW (e.g., 0V) and the computed ciphertext is sent back to the host (i.e., computer) through the device's Tx pin, via UART, in the format of byte 'c' followed by the ciphertext's 16 bytes. The pin trigger indicates the start and the end of the encryption run.

The implemented software-based AES-128 consists of the open-source TinyCrypt's AES-128 encryption algorithm, available at [33]. The fact that no details are available regarding the design of the hardware-based implementation of the device's AES-128 encryption accelerator makes it particularly difficult to attack this implementation.

The effectiveness of the attacks relies heavily, though not exclusively, on the quality of the measurements and the validity of the assumptions made.

4.3 Attacking software-based AES-128

In this Section, the process and results of attacking the software-based implementation of AES-128 running on the ARM Cortex-M4 CPU of the target device are presented.

The targeted software-based implementation of the Advanced Encryption Standard is the TinyCrypt's AES-128 encryption algorithm, from the TinyCrypt Cryptographic Library [33], developed by Intel for resource constrained devices. TinyCrypt implements the AES-128 naïve algorithm based on a substitution table (S-box). The S-box is hard-coded, and the SubBytes operation is performed through S-box lookups. In the algorithm, each AES's state byte is

processed individually. This information is relevant when targeting the implementation. The code pertaining to the encryption function is provided in Figure 41.

The decision to opt for this particular implementation, as opposed to a bitsliced or T-Table, is justified by the purpose of evaluating whether the device exposes software-based implementations to attack. In this experimental segment, the goal is to assess whether the device's hardware is secure for executing software-based cryptographic implementations, and not to specifically evaluate the side-channel resistance of the implementation itself.

To conduct both non-profiled and profiled attacks, two distinct datasets of traces were recorded:

A. Fixed secret key & Random known plaintext: 500.000 traces

B. Random known key & Random known plaintext: 1.500.000 traces

Each trace corresponds to the averaging of 5 consecutive traces for the same key and plaintext, as clarified in the following subsection. The first dataset is the one that is analyzed, in all PA attacks, to deduce the secret key. The second dataset holds relevance solely within the context of the profiled attacks, particularly for the profiling phase.

```
int tc_aes_encrypt(uint8_t *out, const uint8_t *in, const TCAesKeySched_t s)
{
        uint8_t state[Nk*Nb];
        unsigned int i;

        if (out == (uint8_t *) 0) {
                return TC_CRYPTO_FAIL;
        } else if (in == (const uint8_t *) 0) {
                return TC_CRYPTO_FAIL;
        } else if (s == (TCAesKeySched_t) 0) {
                return TC_CRYPTO_FAIL;
        }

        (void)_copy(state, sizeof(state), in, sizeof(state));
        add_round_key(state, s->words);

        for (i = 0; i < (Nr - 1); ++i) {
                sub_bytes(state);
                shift_rows(state);
                mix_columns(state);
                add_round_key(state, s->words + Nb*(i+1));
        }

        sub_bytes(state);
        shift_rows(state);
        add_round_key(state, s->words + Nb*(i+1));

        (void)_copy(out, sizeof(state), state, sizeof(state));

        /* zeroing out the state buffer */
        _set(state, TC_ZERO_BYTE, sizeof(state));

        return TC_CRYPTO_SUCCESS;
}
```

Figure 41 – TinyCrypt's AES-128 encryption function [33].

4.3.1 Power measurement setup

One of the major challenges remains the signal acquisition necessary to conduct the side-channel analysis. Finding the most appropriate measurement setup to acquire leaking traces was, for both attacks (i.e., against software and hardware-based implementations), the most arduous phase of the experimental work.

As mentioned in Section 2.3.4, an oscilloscope is commonly used to measure the power consumption of a device. A variety of approaches suggested in the literature, for measuring power, were attempted. Ultimately, the most effective one involved removing decoupling capacitors (since the measured power was appearing as a steady line and because capacitors have a stabilizing effect over the power within the circuit) and inserting the oscilloscope probe into a power line of the circuit. Determining the power line of the circuit that was linked to the CPU, where the software AES-128 is executed, was a process of trial and error. During this process, for the various experimented approaches and measurement locations in the circuit, the power signal captured by the oscilloscope was visually inspected with the objective of detecting any discernible patterns related with $P_{operation}$ leakage (i.e., 10 rounds of AES-128). Subsequently, leakage assessment was carried out to ensure the capture of data-dependent leakage P_{data} in the measurements. Obstacles related to the presence of noise and jitter in the measurements lead to the study of pre-processing techniques as presented in Section 3.3. However, these were found to be solved by providing the device with an external stable clock signal, in place of the circuit's poor clock source, and by keeping essential decoupling capacitors in the circuit. Note that, if too many capacitors are removed from the circuit, the power inside the circuit might become too unstable to power the microcontroller or to keep it running properly, so this process of finding the right setup must be done with caution to avoid permanently damaging the device.

Once determined the proper measurement setup, the acquisition of power traces for several plaintext blocks had to be automated. A general trace acquisition procedure is illustrated in Figure 42. The acquisition trigger is provided by the device's I/O pin state change (i.e., low to high), described in Section 4.2, and allows to identify the time frame within the measured signal that corresponds to the execution of the encryption run. Note that having programmed the trigger highly simplifies the acquisition process. However, if such ability was not accessible, one would have to consider triggering the oscilloscope capture on other signal, for instance, from communication activity between host and target device. The procedure in Figure 42 is repeated until the desired amount of traces has been acquired.

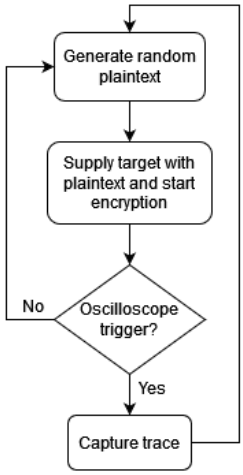

Figure 42 - Flowchart of general trace acquisition procedure

Due to limited storage capacity in the computer used to store and analyze traces, and with the intent of maximizing the amount of leakage information in the acquired traces while minimizing noise unrelated to processing, 5 consecutive traces were captured and then averaged, for the same randomly generated plaintext block, to obtain one single trace.

In the course of the attack, the following equipment was used:

a) **Target device**: As described in Section 4.1.

b) **Tektronix DPO 3014**: The digital oscilloscope includes four channels with a maximum of 100 MHz bandwidth and 2.5 GS/s sampling rate. It can be instrumented via a USB interface, using the programming instructions described in the oscilloscope's manual, to automate the power trace acquisition.

c) **HP EliteBook**: Computer used to perform the acquisition and posterior analysis of the traces.

d) **ChipWhisperer Lite**: The CW Lite is a low-cost open-source hardware platform designed for side-channel power analysis and glitching attacks. It enables interfacing with the target device via UART and allows to control the acquisition of power consumption or electromagnetic emanation. It offers precise clock generation and triggering capabilities, allowing synchronized control between the CW Lite and the target device. It has a 10-bit 105 MS/s ADC for capturing power traces and can be clocked at both the same clock speed as the target and 4 times faster. It has adjustable low-noise gain from -6.5dB to +55dB, which allows it to measure small signals. low n maximum sampling rate of 105 MS/s and a buffer size that can hold up to 24000 samples.

Instead of accomplishing the automated acquisition of traces through the Tektronix oscilloscope, the CW Lite hardware was used, because it allowed to generate and provide the target with a clock signal and thus have the measurement device (i.e., CW Lite) synchronized with the target device. Having the target synchronized with the measurement device had a positive influence on the quality of the measurements. The samples were obtained at a sampling rate equal to the frequency of the clock signal provided to the target device.

Figure 43 depicts the power measured from the target device, using the CW Lite, during a full encryption run of TinyCrypt's AES-128. The power leakage from the execution of the AES operations reveals each round of AES.

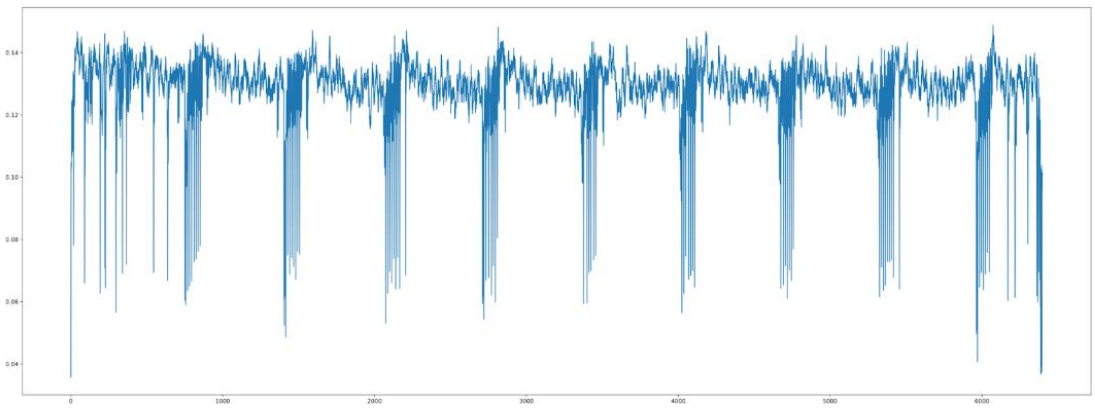

Figure 43 – Power consumption during 10 rounds of TinyCrypt's AES-128.

Figure 44 shows the first round of AES-128, as depicted on the Tektronix oscilloscope. The rising edge of the triggering signal, in blue (i.e., channel 2), indicates the beginning of the encryption run. In this case, the first round of AES (i.e., inside red dotted box) becomes evident due to the repetitive pattern that allows to deduce where the first and second AddRoundKey operations are executed.

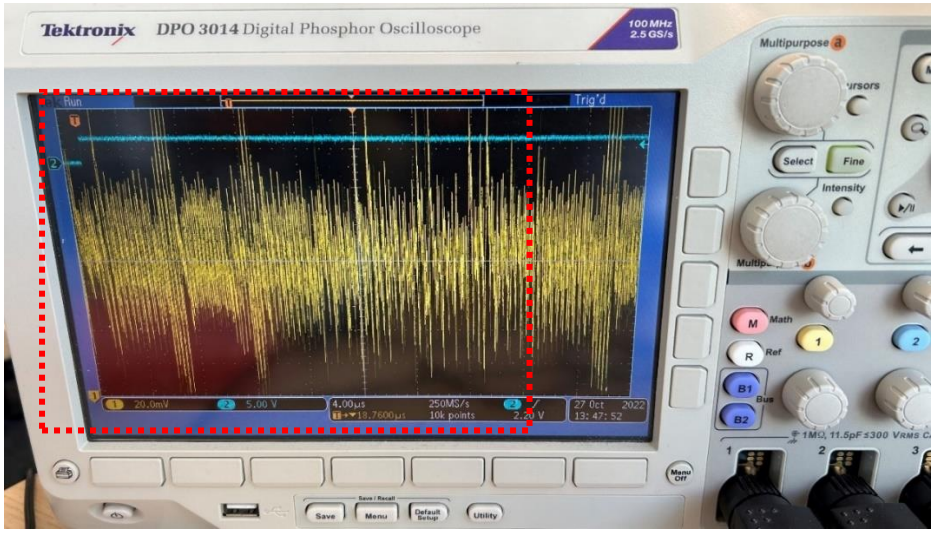

Figure 44 - TinyCrypt's AES-128 first round view on the Tektronix oscilloscope

The final power trace acquisition setup can be described by the symbolic diagram in Figure 45. Note that the target image is not representative. The computer runs a custom script that leverages CW's Python library to interact with the CW Lite hardware via USB and control its hardware. In this setup, the UART communication between the computer and the target device is accomplished via the CW Lite's USB-UART bridge. The CW Lite listens for a rising edge trigger on one of its pins that is connected to the triggering pin of the target device. For each detected trigger, the CW Lite measures the power through a probe, and sends the measured samples to the computer. This way, the computer can record both the trace and the cryptographic data it corresponds to, in separate binary files. The blue line represents the clock signal that is provided by the CW Lite to the target and that must be compatible with the specification of the device's clock input frequency. The target can be grounded to the CW Lite, which helps reduce measurement noise.

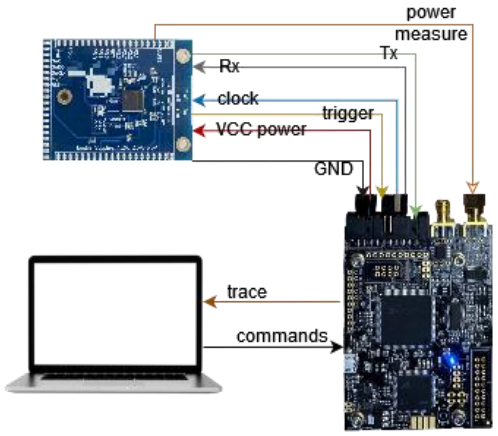

Figure 45 - Diagram of power trace acquisition setup

Functionalities like trace visualization, averaging of traces on-the-fly, and others that were found to be useful for the acquisition process, were implemented in the acquisition script.

4.3.2 Leakage assessment

As introduced in Section 3.2, leakage assessment techniques allow to identify and quantify the data-dependent leakage across each sample of the measured power. As such, it holds great significance in the context of conducting PA attacks in practice.

In the attack strategy pursued in this document, the leakage assessment played a crucial role. It was used in the discovery of the appropriate measurement point in the circuit and measurement setup, to ensure the existence of data-dependent leakage in the measured power. It also allowed for a better understanding of which leakage model could potentially be assumed for some of the attacks. The identification of leakage points (i.e., POIs) allows for more targeted attacks. Instead of analyzing the entire trace, focusing on a window of interest or specific POIs requires processing fewer samples, leading to a faster attack.

The *t*-test is especially useful when you have well-defined groups to compare, as it provides a measure of the significance of differences between the means of two groups. On the other hand, the SNR does not rely on specific groupings and provides a more general measure of the signal strength relative to noise.

With the objective of showcasing the attacks under the two adversarial models stated in Chapter 4's introduction, and to economize the time invested in acquiring traces, the traces were obtained from the encryption of random plaintexts so that they could subsequently be utilized in the key extraction attacks. As such, assessing the leakage based on the SNR instead of the *t*-test was preferred over the due to the fact that it could be applied directly to the dataset of traces subject to analysis in the PA attacks. This is because computing the *t*-test based on partitioning suggested in the literature (i.e., *fixed vs. fixed* or *fixed vs. random*) would require at least a batch of traces to be acquired for a fixed plaintext. Still, it's worth mentioning that the *t*-test was experimented with a partitioning based on the HW of an intermediate value (which isn't a partitioning strategy as beneficial as the *fixed vs. fixed* or *fixed vs. random*), in particular for the key, initial AddRoundKey output, and first SubBytes output bytes, and the obtained results clearly demonstrated the *t*-test to be effective at pinpointing the leakage points. Both the *t*-test and the SNR are known to provide valuable insights regarding the data-dependent leakage, thus, in practice, one can potentially be used to complement the other and vice-versa.

In this first setting, only the knowledge and control over the plaintext is assumed, while the encryption key remains secret and constant. Figure 46 displays the SNR computed at each point in time, based on the plaintext byte value. The visible peaks reveal the points in time where the power varied, across the traces, according to the plaintext byte variation. This allows us to understand how the AES state data is processed by the CPU and ultimately reverse-engineer parts of the algorithm.

In this particular scenario, the source-code of the implementation is available (even though the instructions of the compiled source-code can differ substantially from the actual source-code, due to compiler optimizations), thus we know for sure that the first leakage period corresponds to the execution of *memcpy(state,plaintext)*, where the plaintext bytes are copied to the state array, the second to the *AddRounkKey(state,key)*, where the plaintext bytes are XORed with the secret key bytes, and the third with the *SubBytes(state)* operation where the output of the previous operation, that depends on the plaintext bytes, is loaded from memory and goes through the S-box lookup table. Notice how, up until here, each byte is processed sequentially and then, for what is most likely the *ShiftRows(state)* and the *MixColumns*(state)

operations, the order of the state bytes changes, as expected. The leakage is identified for all these operations, even when the plaintext value is no longer being directly manipulated, because the value that is processed (e.g., the first S-Box output) depends on the plaintext byte. From here, it is possible to identify the so-called POIs.

Ultimately, the output of the first S-Box was chosen as the intermediate value. Therefore, the POIs associated with the processing of this value's bytes are recorded to eventually attempt a more targeted attack. The choice for this intermediate value is backed by the evidence covered in Section 3.1.2.

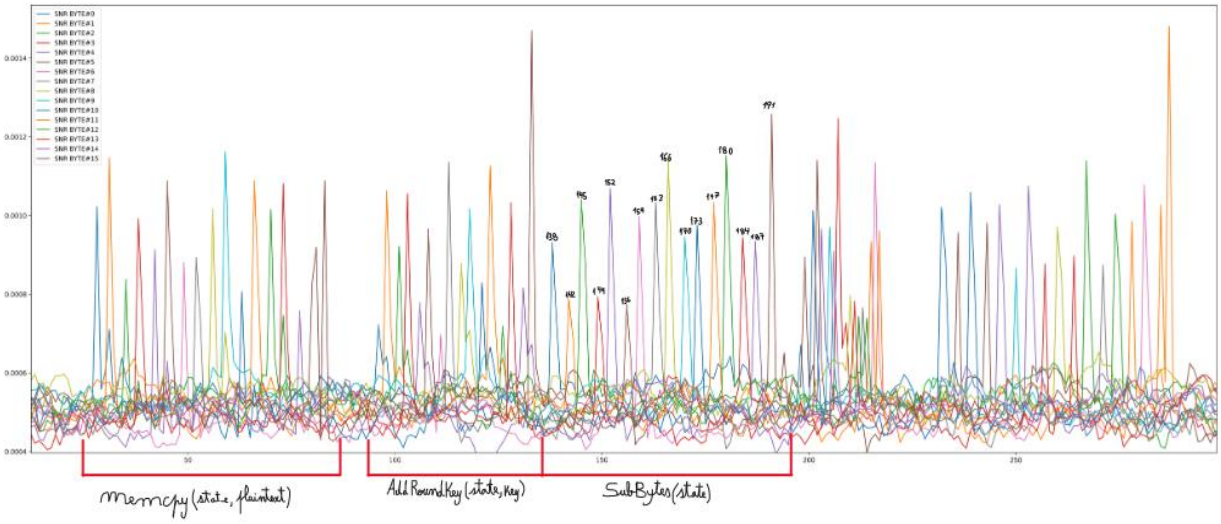

Figure 46 – SNR per sample, for each plaintext-byte: software-based AES.

Now, considering the setting where an attacker is able to obtain traces from the encryption of random known plaintexts using random known keys, instead of assessing the leakage of the plaintext bytes, he can assess the leakage for any intermediate value of the algorithm. As such, for the dataset of 1.500.000 profiling traces, in Figure 46 the SNR was computed at each point in time, based on the known output of the first SubBytes operation, and Figure 47 illustrates the result of computing the SNR based on the HW of this intermediate value.

In Figure 46, peaks are visible, not only for the time frame where the output of the first SubBytes operation is processed, but also for the first AddRoundKey operation. This is justified by the fact that when the output of the SubBytes operation varies, the output of the AddRoundKey operation varies accordingly, thus the leakage becomes apparent.

On the other hand, in Figure 47, the HW leakage of each SubBytes output byte evident in the time frame expected to be associated with the SubBytes operation, and also for the ShiftRows and MixColumns which is understandable since the bytes outputted from the SubBytes operation are indeed manipulated by the ShiftRows and MixColumns operations. No leakage for the HW of the SubBytes output bytes is detected on the AddRoundKey time frame because, while there's a bijective relation between y and x, for $y = \text{SBox}(x)$, where x is the AddRoundKey output and y is the SubBytes output, that results in the AddRoundKey leakage in Figure 46; in the case of the HW function, distinct inputs can map to a same valued output, and therefore the variation of the HW of the SubBytes output is not evident in the AddRoundKey operation's time frame. The peaks in Figure 47 are a good indication that the HW leakage model assumption can most likely be leverage in the following key extraction attacks.

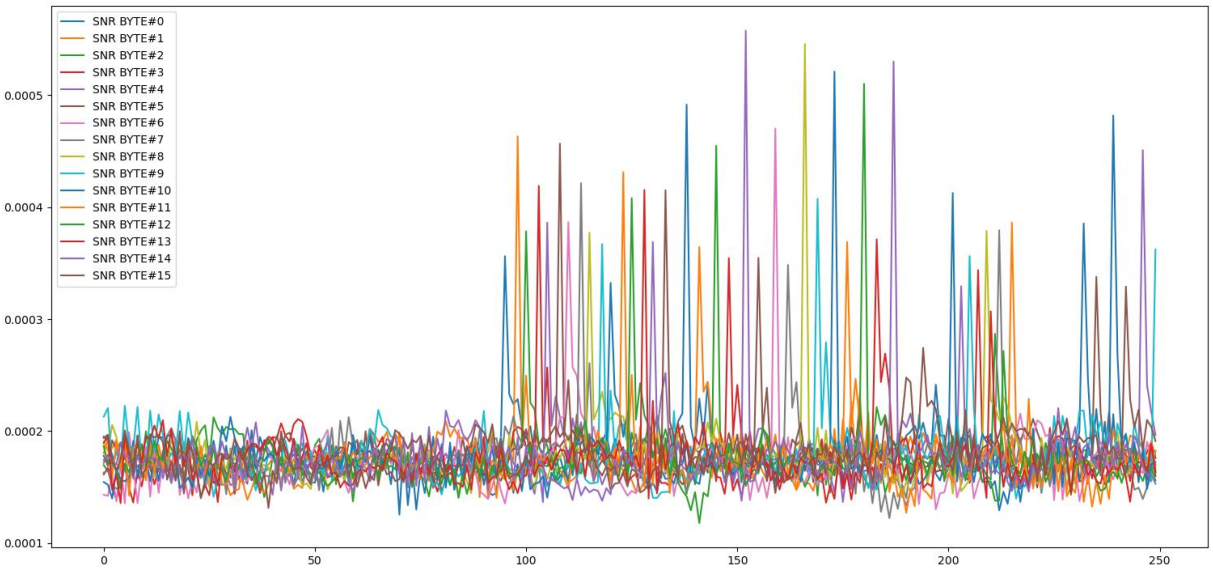

Figure 47 – SNR per sample, for each SubBytes output byte position: software-based AES.

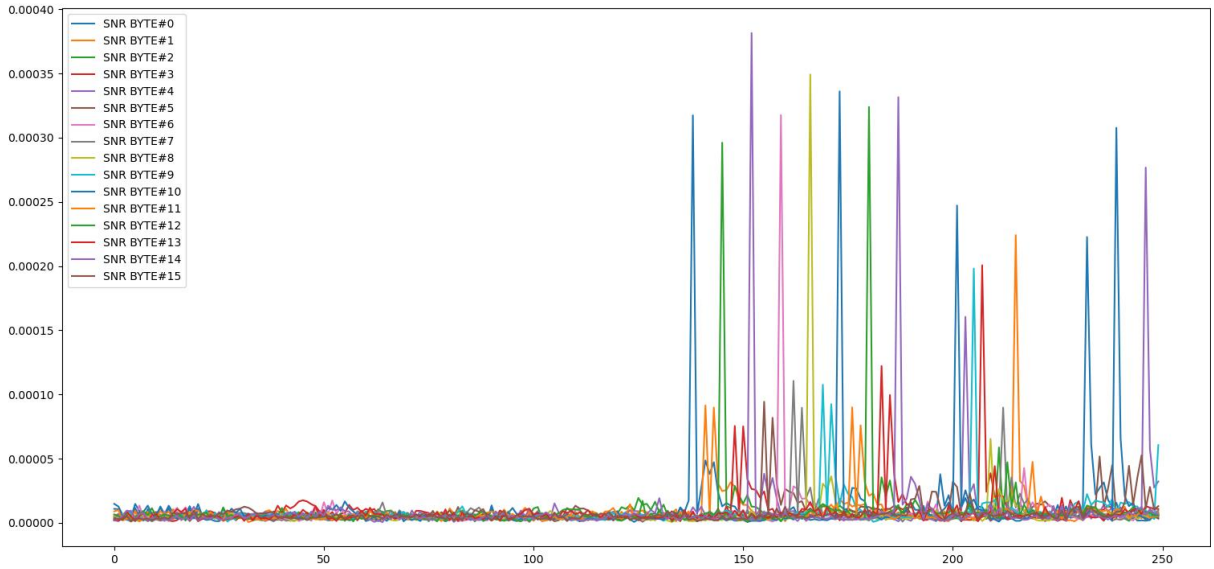

Figure 48 - SNR per sample, for each SubBytes output byte's HW: software-based AES.

4.3.3 Non-profiled attacks

As covered in Chapter 3, there are two types of attacks: the non-profiled attacks, also referred to as model-based, and the profiled attacks. In non-profiled attacks, the adversary has knowledge of the plaintext being encrypted (or resulting ciphertext) and assumes the data-dependent power to be modeled by a certain function, called the leakage model.

Recall that, the goal of both non-profiled and profiled attacks, is to deduce the secret key that is used by the cryptographic algorithm to encrypt data. As such, the dataset of 500.000 traces, acquired from the encryption of known random data using the constant secret key, is the one that is analyzed with PA techniques for the purpose of key extraction.

Favorably, the measurement setup enabled the acquisition of properly synchronized traces, thus no pre-processing mechanism had to be applied to the dataset. As such, and considering the advantages, characteristics and requirements identified for each of the attacks covered in Section 3.1.2, the CPA, LRA, and MIA were regarded as the most beneficial under

these conditions, namely the fact that the traces in the dataset were synchronized and that the leakage assessment unveiled the potential exploitability of an HW leakage assumption.

4.3.3.1 Correlation Power Analysis

The CPA attack was performed across all 250 samples of the time frame where the leakage associated with the initial *memcpy(state,plaintext)*, *AddRounkKey(state,key)*, *SubBytes(state)*, and *ShiftRows(state)* operations was detected.

As previously mentioned, the attacks against the software-based implementation targeted the output of the first SubBytes operation. For CPA, the HW leakage model was assumed. Accordingly, the hypothetical intermediate values were computed for the output of the initial SubBytes operation, for all key-byte candidates, using the known plaintext bytes. Then, the hypothetical power consumption values were computed from the hypothetical intermediate values, using the Hamming-Weight model.

Under these assumptions, CPA was successful at extracting all 16 bytes of the secret key, with a good level of confidence. The level of confidence is the ratio between the highest absolute correlation value (which reveals the most probable correct key-byte guess), and the second highest absolute correlation value. This metric indicates the degree of certainty provided by the attack, for deducing the correct key-byte of a certain key-byte position (i.e., 1^{st} up to 16^{th} byte).

Next, the results obtained from the CPA attack for the extraction of the 3^{rd} key-byte are presented. For this key-byte position, a minimum of 95.000 traces were necessary to determine the correct key-byte value. Focusing the attack solely on the points of leakage (i.e., samples 145 and 146) would allow to lower this number down to 80.000 traces. Table 8 shows the four highest absolute correlation values and corresponding key-byte guess. The key-byte candidate that yielded the highest Pearson's correlation coefficient value was the byte valued 240_{dec}, which is indeed the correct key byte.

Table 8 – CPA attack against 3^{rd} key-byte, on first 250 samples: software-based AES.

key-byte guess	correlation	sample
240_{dec}	0.0172	146
6_{dec}	0.0064	145
57_{dec}	-0.0060	145
33_{dec}	0.0059	145

The correlation values obtained for the correct key-byte guess across all 250 samples is manifestly clear in Figure 49, for the points where the leakage was exploitable. The highest peak represents the correlation value of the correct key-byte guess at sample 146.

Figure 50 describes the highest absolute correlation value of each of the 256 possible key-byte candidates, making the level of confidence, between the correct and the invalid guesses, visibly evident. In this case, the correlation obtained for the correct key-byte guess is around 2.7x higher than the one with the second highest correlation.

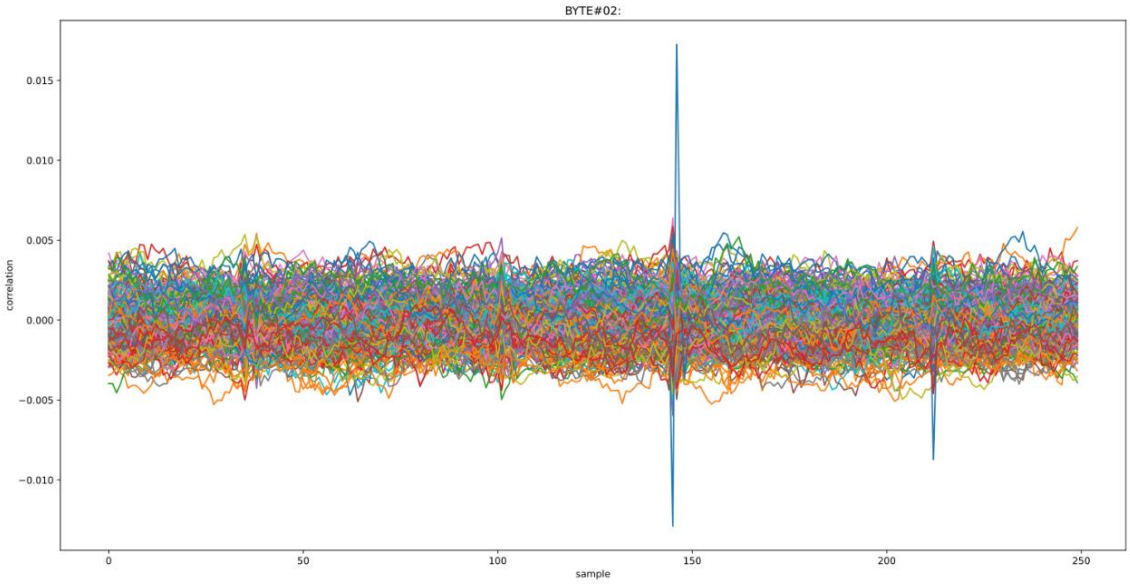

Figure 49 – Correlation value per sample: CPA attack on 3rd key-byte (software-based AES)

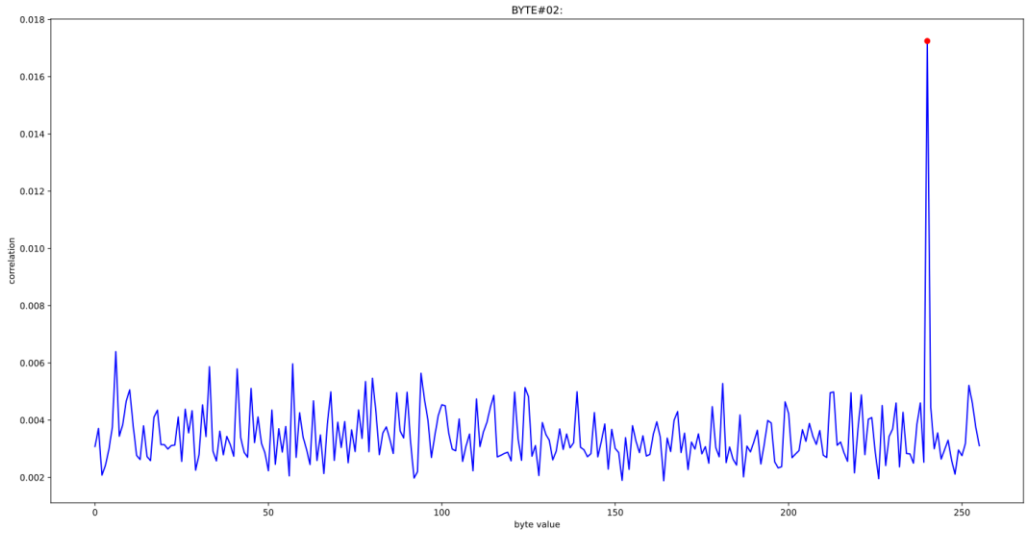

Figure 50 – Maximum correlation value per key guess: CPA attack on 3rd key-byte (software-based AES).

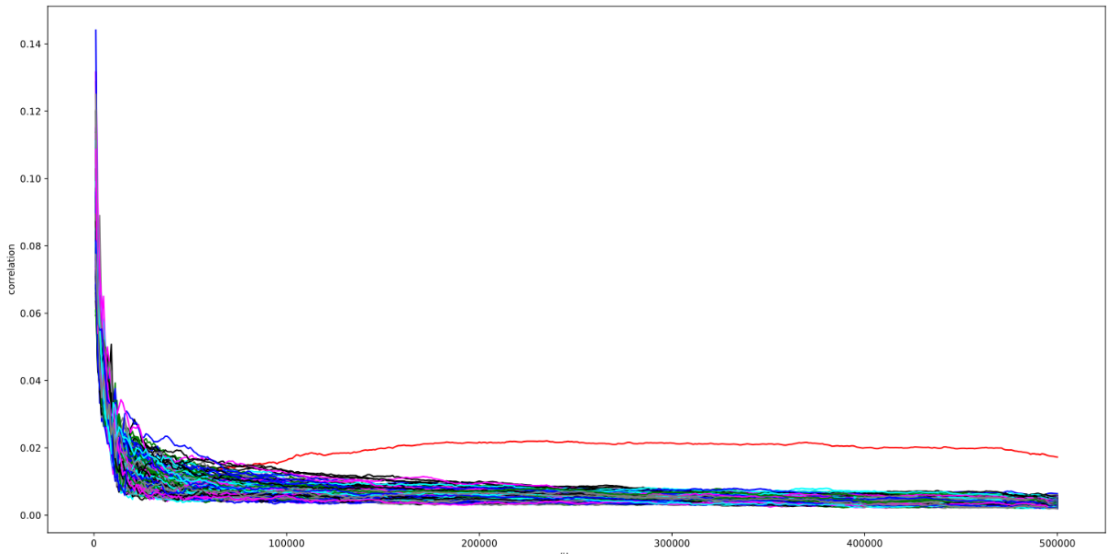

Figure 51 – Evolution of key-byte guess correlation, as the amount of analyzed traces increases: CPA attack on 3rd key-byte (software-based AES).

Figure 51 illustrates a common metric for comparing the efficacy and success of a PA attack: the evolution of the correlation for each key-byte guess, as the amount of analyzed traces increases. This plot is obtained by executing the attack on incremental batches of traces and storing the highest absolute correlation value obtained for each key-byte candidate, in every attack execution. In the attacks against both the software- and the hardware-based implementations, this was achieved by executing the attacks for incremental batches of 1.000 traces, which means that the attack starts by being performed on 1.000 traces of the dataset and the highest correlation value associated with each key-byte guess is recorded, then it is executed for 2.000 traces and the corresponding results are recorded, then for 3.000, 4.000, 5.000, and so on, until having analyzed the total dataset of 500.000 traces. The purpose of this plot is to show after how many traces the correct key-byte guess becomes evident (i.e., minimum number of traces required to extract the key-byte value), and to depict how the confidence level over the correct key-byte guess changes as more traces are analyzed during the attack.

The fact that the HW leakage assumption enabled the extraction of all key bytes with CPA can potentially indicate that the data-buses are precharged before transmitting the looked-up values. Yet, even if this assumption does not hold in reality, for a sufficiently large number of traces it becomes statistically relevant and can ultimately be leveraged for deducing the key.

4.3.3.2 Linear Regression Analysis

The results obtained from the LRA attack on the same dataset of 500.000 traces surpassed those of CPA. This is justified by the fact that LRA works by approximating a leakage model per key-byte guess, that determines the contribution of each bit of an intermediate value (i.e., first SubBytes output), to the observed instantaneous power consumption of the device. So, for each key-byte guess, the LRA computes the coefficients of the linear equation describing the instantaneous power consumption, in (26), that better fits the observed data. As such, a much more accurate leakage model of the device can be assumed and leveraged in the analysis.

Again, the attack was conducted across the first 250 samples and all secret key bytes were successfully inferred.

The results from the LRA attack on the 3rd key-byte are presented next. A minimum of 52.000 traces was necessary to infer the correct key-byte. Table 9 displays the four key-byte candidates that yield the highest R^2 values. The correct key-byte is first with the highest R^2 value of 0.5179, which is around 3.9x greater than the second highest R^2. This indicates that the LRA allows to extract the 3rd key-byte with a greater level of certainty than the one of CPA, which is analogous for the remaining bytes.

Table 9 – LRA attack against 3rd key-byte, on first 250 samples: software-based AES.

key-byte guess	R^2	sample
240_{dec}	0.5179	145
129_{dec}	0.1331	60
171_{dec}	0.1176	177
96_{dec}	0.1137	176

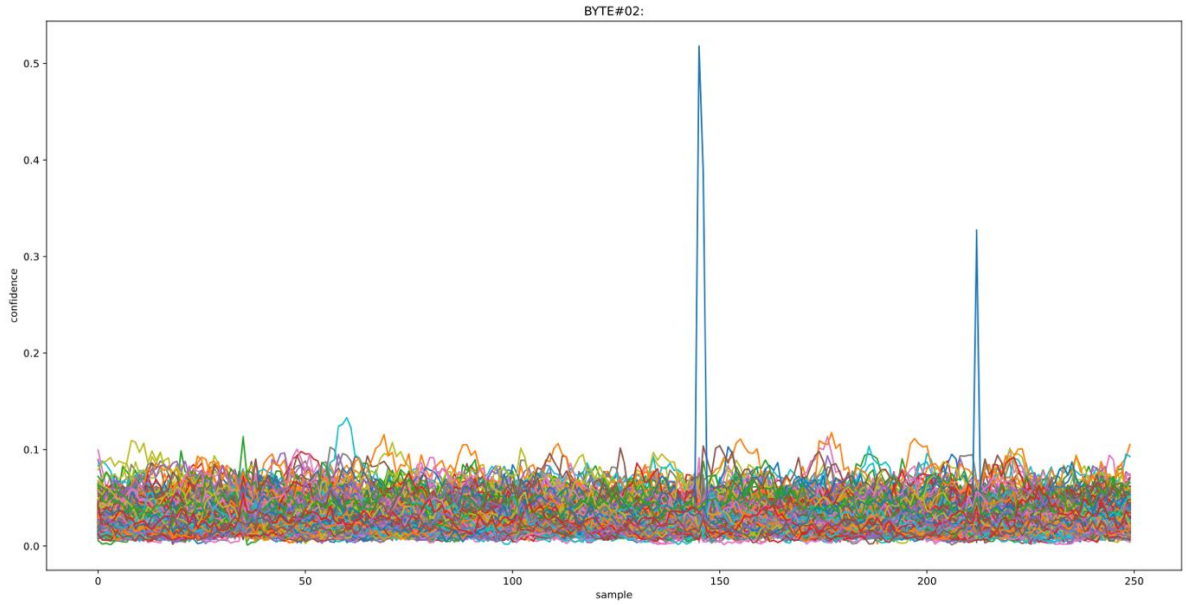

Figure 52 – R^2 value per sample: LRA attack on 3rd key-byte (software-based AES).

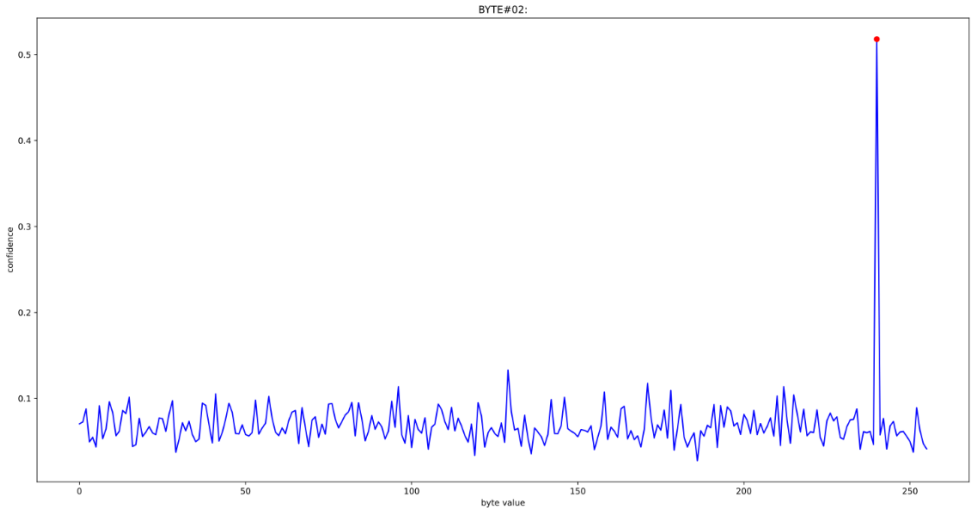

Figure 53 – Maximum R^2 value per key guess: LRA attack on 3rd key-byte (software-based AES).

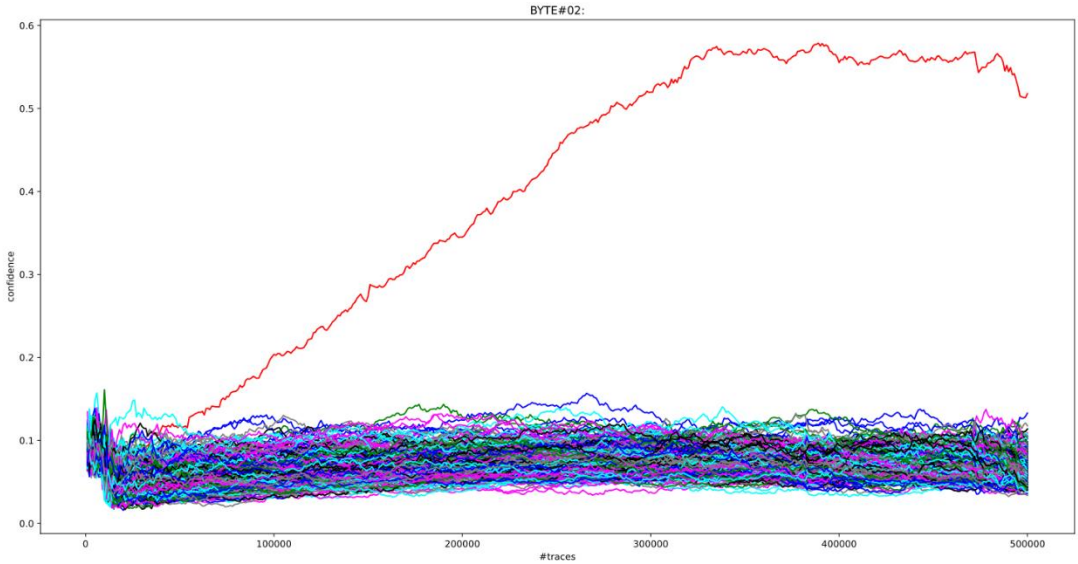

Figure 54 – Evolution of key-byte guess R^2 value, as the amount of analyzed traces increases: LRA attack on 3rd key-byte (software-based AES).

The peaks in Figure 52 reveal the samples at which the instantaneous power consumption of the device was related to the processing of the output of the initial SubBytes operation. Observe how, even the leakage from the manipulation of this intermediate value in the ShiftRows operation would have been sufficient to uncover the correct key-byte.

Figure 53 describes the maximum R^2 value of each of the 256 possible key-byte candidates. The correct key byte guess (240_{dec}), highlighted with a red dot mark, is clearly and unequivocally distinguishable from the remaining guesses. In the case of LRA, the level of confidence consists of the ratio between the highest R^2 value, that indicates the correct key-byte guess, and the second highest R^2 value that corresponds to the second most probable candidate for the key-byte.

With Figure 54, for the 3rd key-byte extraction with LRA, it becomes evident that as the number of analyzed traces increases, the confidence level increases linearly up until 315.000 traces. Then, it becomes fairly constant as the R^2 value for the correct key-byte guess stays at approximately 0.55.

4.3.3.3 Mutual Information Analysis

To perform MIA, the HW leakage model on the output of the first SubBytes operation was assumed, similarly to CPA.

The histogram-based method mentioned in Section 3.1.2.3 was used to estimate the value of the mutual information between computed hypothetical power values and the actual measured instantaneous power consumption.

Due to the fact that estimating a value for the mutual information between sets is significantly more computationally demanding than, for instance, computing the Pearson's correlation value, conducting the MIA across all 250 samples would demand a considerable amount of time, and for this reason the attack was instead conducted on the POIs identified during leakage assessment.

As for the previous attacks, MIA succeeded in retrieving all secret key bytes. However, it required a much larger number of traces than CPA to do so, while providing a substantially lower level of confidence.

The results from the attack for extracting the 3rd key-byte, based on the leakage points at sample 145 and 146, are presented below. A minimum of 177.000 traces were required to extract the correct 3rd key-byte value. Table 10 describes the four key-byte candidates with the highest mutual information value, including the correct guess.

Table 10 – MIA attack against 3rd key-byte, on POIs: software-based AES.

key-byte guess	mutual inf	sample
240_{dec}	0.4860	146
166_{dec}	0.3592	146
34_{dec}	0.3484	146
174_{dec}	0.3444	146

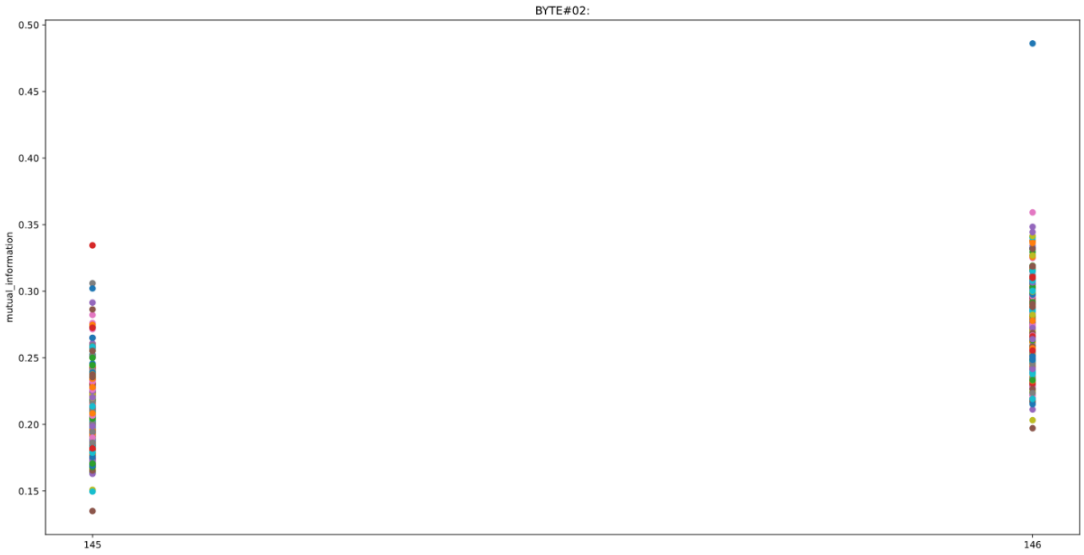

Figure 55 – Mutual Information value per sample: MIA attack on 3rd key-byte (software-based AES).

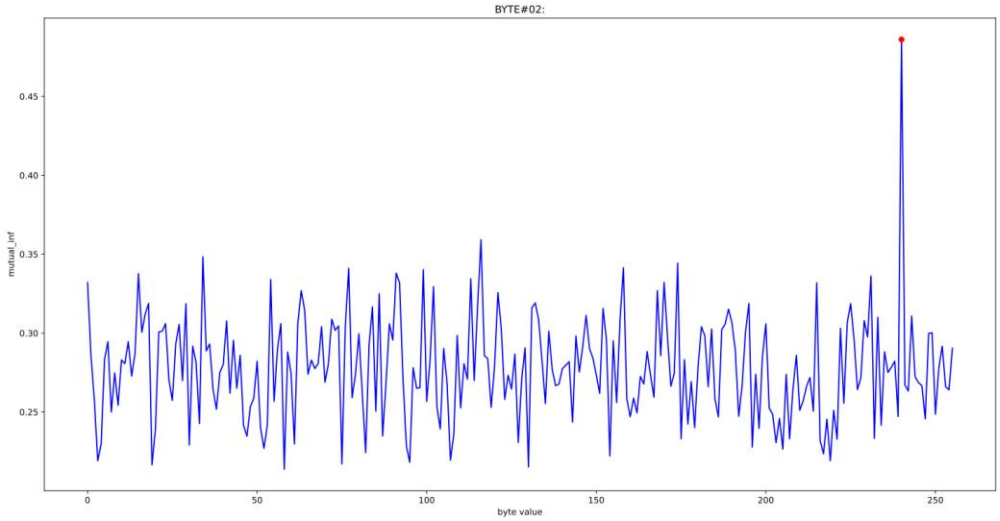

Figure 56 – Maximum Mutual Information value per key guess: MIA attack on 3rd key-byte (software-based AES).

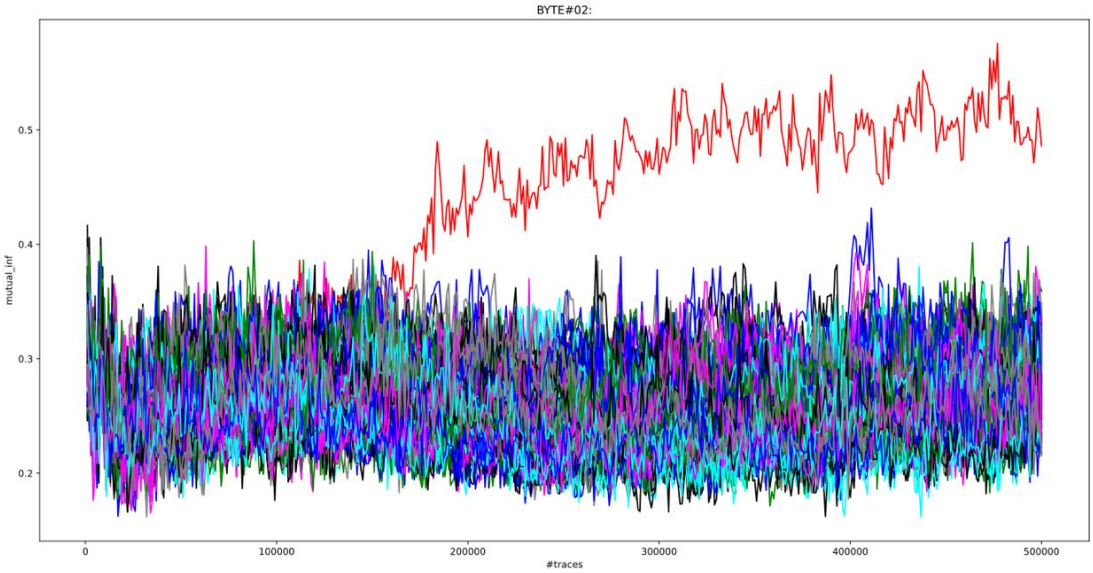

Figure 57 – Evolution of key guess mutual information value, as the amount of analyzed traces increases: MIA attack on 3rd key-byte (software-based AES).

Figure 56 describes the maximum mutual information value obtained for each of the 256 possible key-byte candidates. The level of confidence consists of the ratio between the highest mutual information value obtained for the most probable correct key-byte guess, and the highest mutual information value of the second most likely key-byte candidate.

In this experimental context, the attack fell short on expectations: it is much slower and requires a larger quantity of traces, comparatively to CPA or LRA, to extract a same key-byte, with significantly lower levels of certainty. This final aspect is easily recognized in Figure 57. This can potentially be attributed to the consideration drawn in Section 3.1.2.3, regarding the impact of the estimator's accuracy on the success of the attack.

4.3.4 Profiled attacks

Profile-based attacks, particularly the Template attack and the one based on a Machine Learning classifier, were also carried out to judge whether they allowed an attacker, in practice, to reduce the minimum quantity of traces from the target device necessary to extract the key bytes.

For the profiling phase of the attacks, the dataset of 1.500.000 traces acquired from the encryption of random known input, using random known keys, was used.

For the matching phase, only 100.000 traces were used, out of the dataset analyzed in the non-profiled attacks with 500.000 traces that were acquired from the encryption of random known inputs, using a constant secret key. This choice is justified by the main motivation for employing this type of attacks: require a smaller number of traces from the target device for successfully inferring the key, at the cost of an expensive profiling phase associated to the processing involved in building profiles and acquiring a large quantity of profiling traces.

Since profiled attacks are more demanding in terms of processing and memory, it was essential to leverage the knowledge of leakage points and focus the attacks on these POIs. Given that the LRA is claimed to be an improvement over the Stochastic attack, no experiments in this chapter were carried out for the latter.

4.3.4.1 Template attack

In the profiling phase of this attack, a multivariate Gaussian distribution was calculated for each known intermediate value, from the leakage points on the profiling dataset associated with the output of the SubBytes operation.

Then, in the matching phase, the multivariate Gaussian distribution was calculated for each hypothetical intermediate value based on the observed power at the same leakage points considered in the profiling phase, but now for the target dataset, and ultimately these were compared with the templates computed in the profiling phase.

Building the templates was relatively fast, hence the time dedicated to the acquisition of 1.500.000 traces for profiling was the most significant cost.

Even though the attack didn't succeed at extracting the last secret key-byte with less than 100.000 analyzed traces, the attack was deemed successful in the sense that it enabled to reduce the minimum number of traces required to extract some key bytes, notably five bytes were deduced from the analysis of 2.000 traces and other three from around 4.000 traces. Still, the attack was not favorable for all key-byte positions.

The result of the Template attack against the 3rd key-byte, based the POIs associated with the output of the initial SubBytes operation, is presented in Table 11. The guess that yielded the

highest confidence value was the correct key-byte valued decimal 240. This attack doesn't allow to detect the sample that yielded the highest confidence value as the Template attack is a multivariate attack, so it considers the combination of the specified POIs instead of each POI individually. Figure 58 describes the maximum confidence value obtained for each key-byte candidate.

The confidence levels obtained from the Template attack are relatively low. Still, they allow to distinguish the correct key-byte. It is important to mention that the outcome of this particular attack could have been potentially improved if the traces of both datasets wouldn't have been averaged since, as pointed out in Section 3.1.3.1, the Template attack takes advantage of the characterization of noise.

Table 11 – Template attack against 3rd key-byte, on POIs: software-based AES.

key-byte guess	confidence
240	_.5181
113	_.4787
206	_.4723
138	_.4684

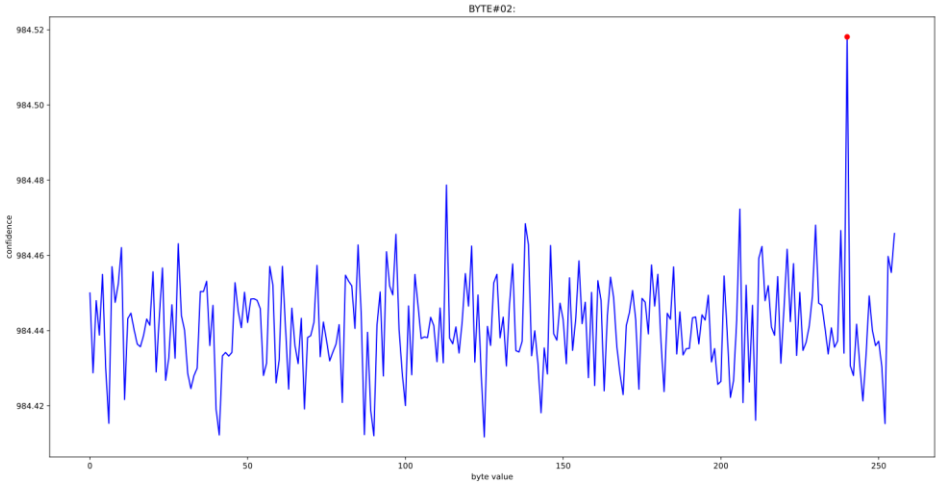

Figure 58 – Maximum confidence per key-byte guess: Template attack on third key byte (software-based AES).

4.3.4.2 Machine Learning classifier

In this attack, the profiling phase consisted of training a MLP neural network, tuned by the authors of [20], with the set of profiling traces. The traces were partitioned based on the output of the first SubBytes operation. It is noteworthy that the training phase was particularly time-consuming. Knowing the number of epochs (i.e., number of times the ML algorithm iterates over the entire dataset during the training process) that will be necessary to properly train the model is particularly challenging. Unfortunately, even for a larger number of epochs, throughout the training process the model's accuracy remained consistently low.

As such, in the matching phase the model failed to correctly deduce any of the secret key bytes from the 100.000 traces. Consequently, alternative partitioning methods (e.g., HW of SubBytes output) were tried without success. Even though this attack performed well when being tested on a dataset of simulated traces, it failed to succeed for this experimental scenario.

Two plausible causes for the lack of success of the attack in this experimental scenario can be contemplated:

- Insufficient training: the model may have been trained for insufficient epochs. A significant limitation of this attack is the amount of time needed for training the model. One model has to be trained per each of the 16 key bytes, and it is difficult to know *a priori* the effective amount of epochs needed to properly train the model;
- Poor-quality data: the two datasets (profiling and target datasets) were found to have a vertical misalignment problem, induced by the CW Lite hardware, due to the fact that they were acquired in separate moments. This went unnoticed for the Template attack since the analysis performed by such attack is based on probability density functions. However, in the case of the ML classifier, it may have prevented it from correctly classifying the traces of the target dataset, since it was trained on the traces from the profiling dataset. The misalignment between the two datasets is depicted in Figure 59.

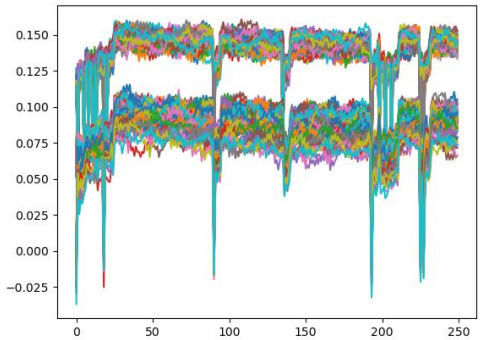

Figure 59 - Vertical alignment problem between profiling and target datasets: software-based AES.

4.3.5 Conclusion

The results presented in this Section "4.3 - Attacking software-based AES-128" have demonstrated that the target device is indeed vulnerable to side-channel attacks, particularly those involving Power Analysis. The efficacy of these attacks is evident in their ability to uncover sensitive information processed within the device's general-purpose CPU. Hence, relying on unprotected software-based implementations running on the device for safeguarding sensitive information should not be presumed secure, since it has been shown that the data-dependency in the device's power consumption exposes the processed data to attack and can indeed be exploited.

Several conclusions were drawn and interpreted for each attack. Assessing the leakage prior to the attacks proved to be an essential step, not only at determining the time periods at which each operation occurred but also to concentrate the attacks on the data-dependent instants of the traces. The effectiveness of each attack, regarding the minimum necessary number of traces to extract each key-byte, can be compared in Table 12. The values are described in a scale of x1000 traces. The confidence levels conveyed by the attacks, for deducing each key-byte, are described in Table 13. These were computed as defined in the experimental section of each attack: ratio between the confidence value (i.e., correlation, mutual information, or other) of the most likely correct key-byte candidate and of the key-byte candidate with the second highest confidence value. Regarding these results, it's worth noting that the non-profiled attacks were carried out on a dataset of 500.000 traces, while the Template attack was performed on 100.000 of those traces. The templates were built on 1.500.000 profiling traces.

Since the success of the attacks is limited by the number of traces that an attacker can acquire from the target device and analyze (for instance, in the context of breaking firmware

encryption, it might be limited by the flash memory storage capacity), it is crucial to reason how the attack could be enhanced in such a way that the number of traces needed to extract the full key could be reduced. In this particular experiment, increasing the sampling rate for which the measurements were acquired would augment the amount of exploitable information available in the traces which could lead to improved results in terms of number of traces required for full key extraction and associated confidence level.

Table 12 - Summary of minimum required traces for each attack (scale of x1000): software-based AES.

		key-byte position																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	***avg***
attack	CPA	88	95	95	59	50	155	40	85	25	90	60	45	40	95	95	67	**74**
	LRA	77	114	52	45	81	101	66	66	50	78	45	73	62	56	96	58	**70**
	MIA	310	330	170	355	290	310	230	260	150	235	180	190	140	195	285	155	**236**
	Tem.	78	91	2	4	13	44	2	41	2	45	3	3	2	37	2	-	**25**

Table 13 - Summary of confidence levels from each attack: software-based AES.

		key-byte position																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	***avg***
attack	CPA	2.7 58	2.4 13	2.6 88	2.9 06	2.6 14	2.0 77	3.4 69	2.2 92	2.6 56	2.4 29	2.4 46	3.0 88	2.5 88	2.0 27	2.2 88	3.1 29	**2.6 2**
	LRA	3.5 80	4.2 06	3.8 91	4.6 15	3.7 34	3.7 81	4.1 94	3.4 70	3.4 58	3.0 97	3.9 57	3.9 75	3.9 25	3.6 17	3.3 78	4.2 15	**3.8 2**
	MIA	1.0 52	1.3 28	1.3 53	1.2 04	1.0 08	1.1 41	1.4 82	1.3 83	1.3 04	1.4 52	1.2 85	1.3 75	1.3 13	1.3 06	1.3 75	1.6 56	**1.3 1**
	Tem.	1.0 23	1.0 05	1.0 82	1.0 04	1.1 18	1.7 19	1.1 56	1.0 08	1.7 44	1.1 12	1.0 43	1.1 09	3.0 62	1.0 35	1.0 56	~~1.0~~ ~~03~~	**1.2 9**

4.4 Attacking ASIC-based AES-128

This section presents the experimental process and results of attacking the ASIC-based implementation of AES-128 that is implemented by the dedicated encryption accelerator hardware that resides inside the target device.

The details regarding how this implementation is accomplished in hardware remain unknown. This lack of knowledge adds a significant level of difficulty to the attack, complementary to the challenges that are typically associated with targeting ASIC-based implementations, e.g., since the internal circuitry is optimized for performance and power efficiency, it is often difficult to observe the individual operations and intermediate values during execution. Even though this level of obscurity adds an extra layer of difficulty to the attack, it also serves as motivation to dedicate further efforts to attempt to break it. As emphasized by Kerckhoff's principle, the security of a cryptosystem should rely solely on the secrecy of the key, rather than on the secrecy of the algorithm or other implementation details.

To conduct both non-profiled and profiled attacks, two distinct datasets of traces were recorded:

A. Fixed secret key & Random known plaintext: 500.000 traces

B. Random known key & Random known plaintext: 1.500.000 traces

Similar to the approach used for "Attacking software-based AES-128", each trace represents the average of 5 consecutive traces acquired for the same key and plaintext. Again, this was done with the intent of eliminating normally distributed noise unrelated to processing, and to reduce the memory needed for storing and analyzing the datasets. For all PA attacks, the first dataset is the one analyzed for the purpose of key extraction. On the other hand, the second dataset is only necessary for the profiling phase of the profile-based attacks.

4.4.1 EM measurement setup

Following up the attack conducted against the software-based implementation running on the device's CPU, the successful power measurement setup described in Section 4.3.1, was attempted to measure the power associated with the data-processing of the crypto accelerator. Recall that, in this power measurement setup, the measuring probe was inserted in the power path, in the power management integrated circuit external to the SoC, that was found to be powering directly the ARM processor. Unfortunately, the hardware cryptoprocessor often works on another power domain [36], that is unknown to the attacker, and that is shared with other peripherals within the SoC. Likely for this reason, the prior measurement setup didn't allow to visualize any repeat signal between the trigger that could potentially be related to the AES computation. The leakage assessment conducted on a set of traces acquired from this setup confirmed the absence of leakage associated with the processing of the AES state array. However, it clearly revealed that the last of the acquired samples, within the timeframe of the trigger, was dedicated to the transfer of the ciphertext bytes from the cryptoprocessor to the ARM processor, and that the first samples were devoted to initially transferring the plaintext bytes set on the ARM processor to the cryptoprocessor. From here, other power lines in the circuit were tested for data-dependent leakage, but none yielded any successful outcome.

After spending a considerable amount of effort trying to find an effective power measurement setup to target the ASIC-based implementation of AES-128, the research presented in [36], against a hardware cryptoprocessor of a modern SoC in a mobile phone, elucidated to the potential of exploring the EM side-channel instead. The research conducted in [36] is of particular interest to the experimental work carried out in this section, as it closely aligns with the same objectives and conditions.

Given that silicon chips are primarily composed of transistors and metal wires, any transient current flowing through the logic gates generates an EM field. This radiated signal is directly related to the device activity and can be measured near the chip, with the adequate EM probes [36]. As covered in Section 2.3.4, the near-field EM measurement offers the advantage of enabling to concentrate on localized activity by positioning the probe in close proximity to the specific functional block. The power consumed solely by the AES computation within the dedicated encryption accelerator of the SoC is likely to be negligible when compared to the aggregate power consumption of the surrounding peripherals and CPU. Still, finding the precise location of the AES computation over the SoC remains challenging and highly depends on the target hardware. A block cipher logic is usually comprised of a few thousand transistors, whereas the SoC incorporates several billions [36].

In a wise move, the authors of [36] recorded the thermal behavior of the SoC while it encrypted data using the AES hardware, in order to precisely determine the location of the

cryptoprocessor over the chip die. Alternatively, an automated XY table equipped with a steady clamp holding the EM probe could be used to acquire a dataset of enough traces, per XY location over the surface of the chip, to posteriorly determine the XY location of the dataset that yielded the strongest leakage, and consequently identify the most favorable EM measurement setup. However, both strategies were beyond the capabilities of the available experimental equipment, and so the adopted approach was to position the probe on top of the SoC and move it across its surface while it executed hardware-AES and visually inspect the trace, using the oscilloscope, for a persistent pattern that could be related to cryptoprocessor's activity. This seemed feasible, considering the small size of the SoC's surface. Then, to determine whether the position for the probe was sufficiently satisfactory to conduct the final acquisition, the assessment of the leakage was the determining factor.

This strategy enabled an estimation of the relative position of both the cryptoprocessor and the ARM processor. Figure 60 shows the EM signal, in purple, recorded from the execution of the software-based AES implementation covered in Section 4.3, by placing the EM probe near the chip's area that was presumed to correspond to the CPU region.

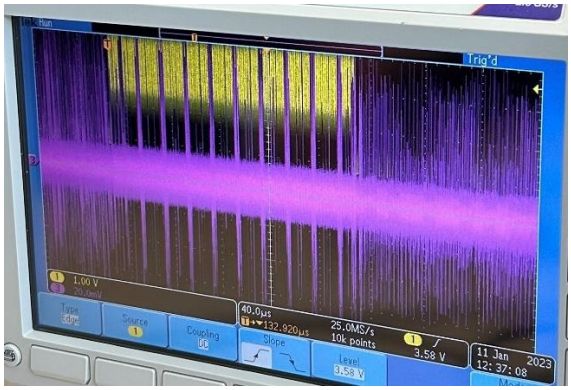

Figure 60 - EM signal on oscilloscope: 10 rounds of TinyCrypt's software-based AES-128

After determining the probe positioning and the location on the SoC's surface that was satisfactory to measure the activity associated with the AES's cryptoprocessor, the probe was fixated on the top of the device to ensure that it didn't move throughout the process of acquiring the datasets for the attack. Ensuring this condition is imperative in the acquisition of EM traces because even a minor adjustment in the probe's placement can significantly alter the characteristics of the measurements (e.g., amplitude of the data-dependent fluctuations, or capture noise from extra sources).

Maintaining the same acquisition procedure described in Section 4.3, the two datasets of traces were obtained in a very non-intrusive way, as depicted in Figure 61, using the EM probe together with a signal amplifier. The equipment used in this experiment was the following:

a) **Target device**.

b) **Tektronix DPO 3014**.

c) **HP EliteBook**.

d) **CW Lite**.

e) **Langer PA 303 SMA**: This preamplifier is designed for the amplification of measuring signals, e.g., weak signals of near-field probes with high resolution. The near-field probe is connected to the input of the preamplifier while the output is

connected to the oscilloscope or CW Lite. The frequency range is 100 kHz - 3 GHz, and the gain is 30 dB.

f) **Langer Near Field Probe set RF1**: this set consists of four passive near-field probes for measuring E-fields and magnetic fields from 30 MHz to 3 GHz. They have a current attenuating sheath and, therefore, are electrically shielded. For this attack, the RF-U 2.5-2 probe was preferred. More details can be found at [65] Langer's website.

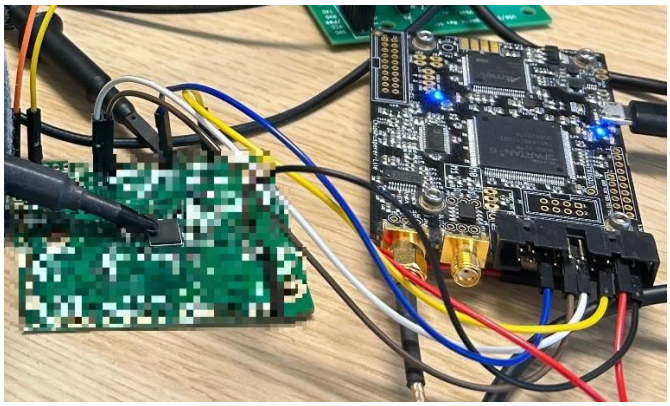

Figure 61 – Actual EM acquisition setup: probe placement.

Figure 61 shows the placement of the EM probe over the actual SoC. The probe is connected to the signal amplifier (not displayed) that connects to the CW Lite. Ten of the traces acquired for analysis, from the encryption of random data on the dedicated crypto accelerator, are plotted in Figure 62.

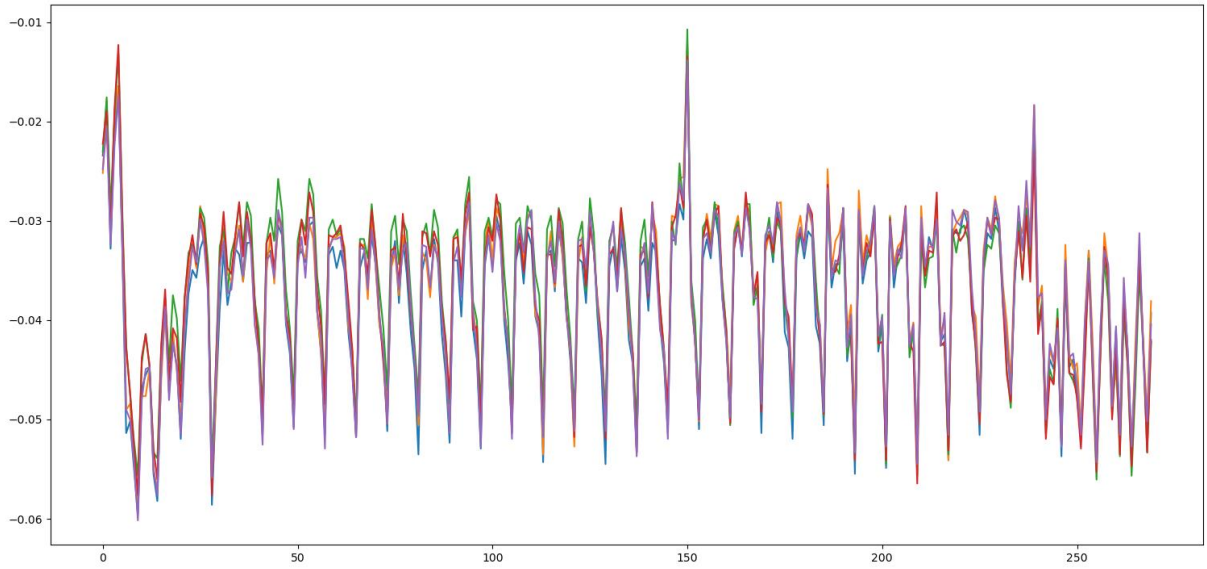

Figure 62 – EM traces from hardware-based AES-128 encryption.

It's important to recognize that the acquisition for the attack was conducted "in the wild", meaning it took place without adequate isolation (e.g., anechoic chamber) from surrounding noise sources such as radiation emitted by other devices. This lack of isolation inevitably introduces additional noise into the measurements. Ultimately, opting for this course of action allows to attest whether the attack against the target can be accomplished under a less strict adversarial setting.

4.4.2 Leakage assessment

The methodology utilized in Section 4.3.2 for assessing the leakage in the traces and consequently gain insights into the data-path of the underlying algorithm, set a solid foundation for delving into assessing the leakage of this more complex target. As mentioned in Section 4.4, targeting this ASIC-based implementation comes with the extra layer of difficulty of lacking information about the underlying design of implementation in addition to the fact that it is often difficult to observe individual operations and intermediate values of the algorithm during execution. Having knowledge of the way the dedicated IC implements the AES computation, or even a more high-level view of the AES state processing like the one depicted in Figure 13 for an ASCI-based implementation from the literature, would greatly facilitate the task of assessing the data-dependent leakage and of attacking the target implementation.

Since the EM emanation is directly related to the power consumption of the device, in particular to the current flowing through the logic gates of nearby components, the assumptions and techniques leveraged in Power Analysis may be applicable. The reader is referred to Section 2.3.2 for additional information on the subject.

Without knowledge regarding the processing of data inside the cryptoprocessor's hardware, it's complicated to decide on an intermediate value or associated leakage model, to conduct an attack. In this context, leakage assessment techniques are a valuable tool to attest assumptions pertaining to data processing and corresponding leakage. Accordingly, my strategy revolved around assessing the leakage of the datasets under various assumptions to identify the one that best aligned with the device's behavior, particularly concerning leakage. Then, this assumption would be attempted in the PA attacks to successfully deduce the secret key of the device.

Throughout the course of finding a satisfactory position and location on the SoC's surface to acquire the dataset of traces for conducting the attacks, the evaluation focused on assessing the leakage of the plaintext and ciphertext bytes. Figure 63 and Figure 64 show the SNR peaks obtained for each plaintext byte and ciphertext byte, respectively. The ciphertext leakage, detected between the 180^{th} and 220^{th} samples, is likely to be related to the transfer of the ciphertext bytes from the cryptoprocessor to a region of memory accessible to the ARM processor. On the other hand, the plaintext leakage between the 110^{th} and 145^{th} samples may be related to the writing of the plaintext bytes, from a region of memory common to the ARM processor, to dedicated registers of the cryptoprocessor, to be processed. This presumption is backed by the leakage assessment conducted on the power measurements acquired with the setup of Section 4.3.1 but while executing the ASIC-AES. The assessment revealed strong plaintext leakage prior to the 110^{th} sample, which should correspond to a software *memcpy* call that wrote the plaintext bytes into a memory region accessible by the ARM processor and whose address was then indicated to the cryptoprocessor. Additionally, the assessment on the ciphertext bytes indicated leakage also between the 180^{th} and 220^{th} samples, exhibiting an even more distinctive and clear leakage pattern for each byte.

To determine the intermediate value and leakage model that could potentially be assumed to correctly deduce the key with the PA attacks, various distinct assumptions were attested, including:

1. Hamming-Weight of output bytes from: first SubBytes, first AddRoundKey, first MixColumns.
2. Hamming-Distance between bytes of plaintext and first SubBytes output.
3. Hamming-Distance between bytes of plaintext and first MixColumns output.
4. Hamming-Distance between bytes of first AddRoundKey and SubBytes output.

5. Hamming-Distance between bytes of first SubBytes and MixColumns output.
6. Hamming-Distance between bytes of first SubBytes and ShifRows output.
7. Hamming-Distance between bytes of first ShiftRows and MixColumns output.
8. Hamming-Distance between bytes of ciphertext and last SubBytes output.
9. Hamming-Distance between bytes of ciphertext and last ShifRows output.
10. ...

During the arduous phase of discovering the (power, and then EM) measurement setup for targeting the cryptoprocessor, the gathered traces were also being assessed for possible 16-bit or 32-bit leakages, rather than just byte leakages, since it could be the case of the cryptoprocessor manipulating more than 8 bits at a time. Even in such a scenario, the 8-bit leakage assumption should suffice. Other assumptions like the HD leakage between arrangements of bytes resultant from intermediate AES operations were tested, without any relevant outcome.

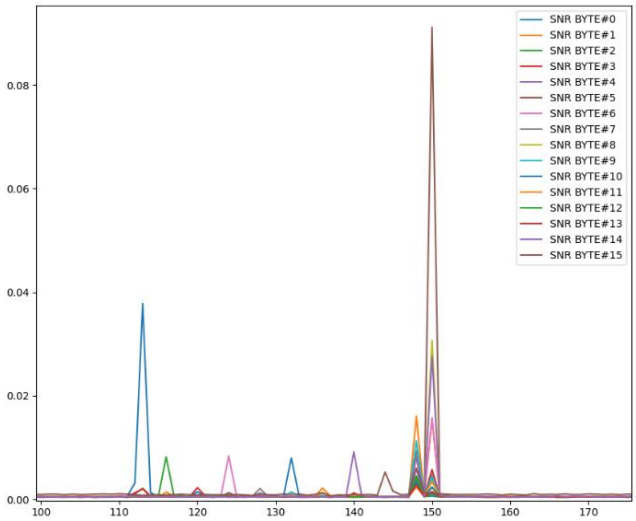

Figure 63 - Plaintext bytes' leakage: ASIC-based AES.

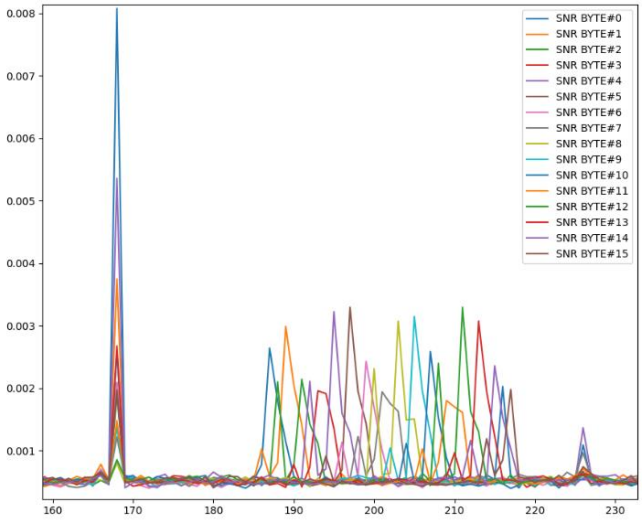

Figure 64 - Ciphertext bytes' leakage: ASIC-based AES.

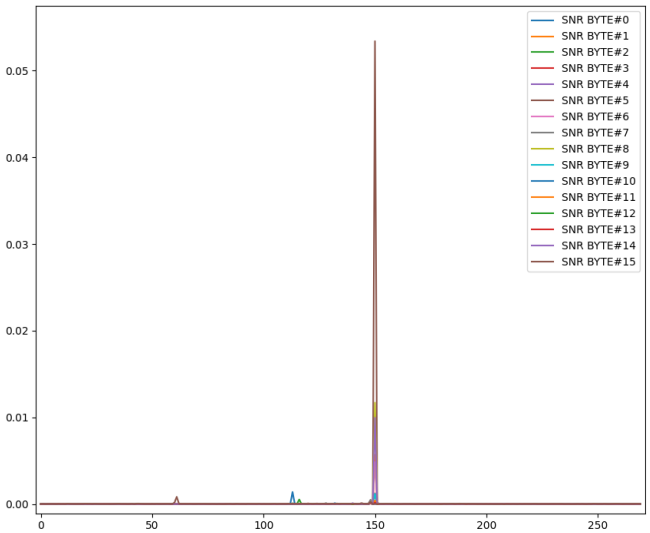

Figure 65 - HD leakage between bytes of plaintext and of first SubBytes output: ASIC-based AES.

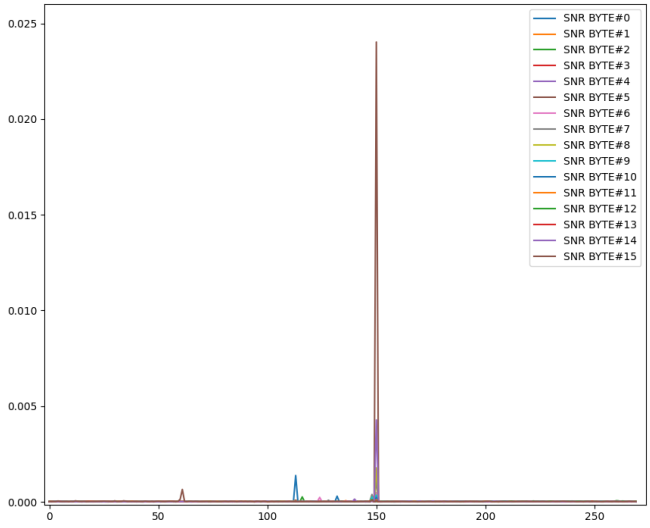

Figure 66 - HD leakage between bytes of first AddRoundKey and SubBytes outputs: ASIC-based AES.

The assumptions of an HD leakage between the bytes of the plaintext and the first SubBytes output (Assumption 2.), and of an HD leakage between the bytes of the first

AddRoundKey and SubBytes operation (Assumption 4.), with the SNR illustrated in Figure 65 and Figure 66, respectively, resulted in the most promising leakage assessment results. Even though both appear to be potentially exploitable leakages, only the HD leakage between plaintext and SubBytes output (Figure 65) allowed to correctly deduce the secret key bytes through the attacks that follow.

4.4.3 Non-profiled attacks

The measurement setup allowed the acquisition of properly synchronized traces, eliminating the need for preprocessing the dataset. Following the same line of thought of 4.3.3, the CPA, LRA and MIA were considered the relevant attacks to experiment against this target.

Upon conducting the leakage assessment, it became apparent that the 150^{th} sample was probably the point where a significant part of actual AES computation took place, and as such the following attacks were directed towards the analysis of this specific sample.

4.4.3.1 Correlation Power Analysis

The CPA attacks was performed on the 150^{th} sample where the leakage associated with the transition from the plaintext bytes to the output of the first SubBytes operation was detected. Accordingly, the HD leakage model between the two intermediate values was assumed.

Under this assumption, the CPA attack, as defined in Section 3.1.2.2, allowed to extract 13 bytes out of the total 16 secret key bytes, from the analysis of a dataset with 500.000 traces. Despite not being able to deduce all key bytes, the attack is considered a success since, in the context of breaking firmware encryption it's feasible to brute-force all 2^{24} remaining combinations to determine the 3 missing bytes. The level of confidence for the extraction of each byte was computed to evaluate the efficacy of the attack in ensuring the attacker that the correct key bytes were indeed inferred.

It's definitely worth pinpointing that three key bytes were successfully extracted with the analysis of less than 1000 traces. The levels of confidence and the minimum number of traces necessary to extract each of the key bytes are described in Table 18 and Table 17, respectively, for all the attacks carried out.

Next, the results obtained from the CPA attack on the 3^{rd} key-byte are presented. For this key-byte position, a minimum of 74.000 traces were required to determine the correct key-byte value. shows the four highest absolute correlation values and corresponding key-byte guess. The key-byte candidate that yielded the highest Pearson's correlation coefficient value was the byte valued 139_{dec}, which is indeed the correct key byte.

Table 14 - CPA attack against 3^{rd} key-byte, on 150th sample: ASIC-based AES-128.

key-byte guess	correlation	sample
139_{dec}	0.0172	150
228_{dec}	-0.0079	150
160_{dec}	-0.0075	150
12_{dec}	0.0073	150

The superior correlation value obtained for the correct key-byte at the 150th sample is visible in Figure 67.

Figure 68 describes the evolution of the correlation value obtained at sample 150, for each key-byte guess, as the amount of analyzed traces increases. The details for obtaining this plot, that allows to interpret how the confidence level for extracting a key-byte changes for an attack depending on the quantity of analyzed traces, are provided in Section 4.3.3.1.

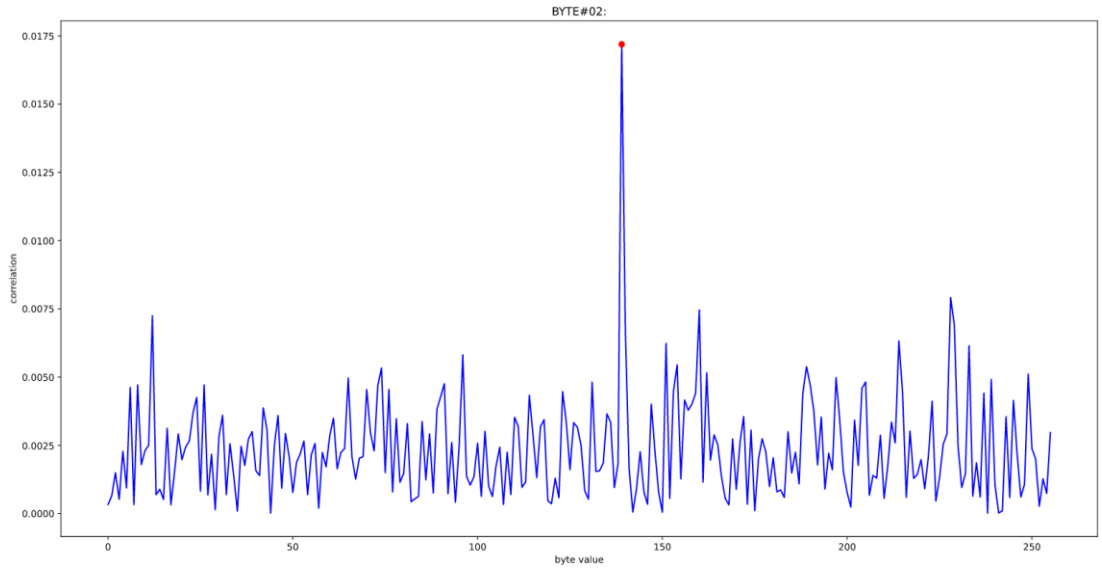

Figure 67 - Correlation value per key-byte guess in sample 150: CPA attack on 3rd key byte (ASIC-based AES).

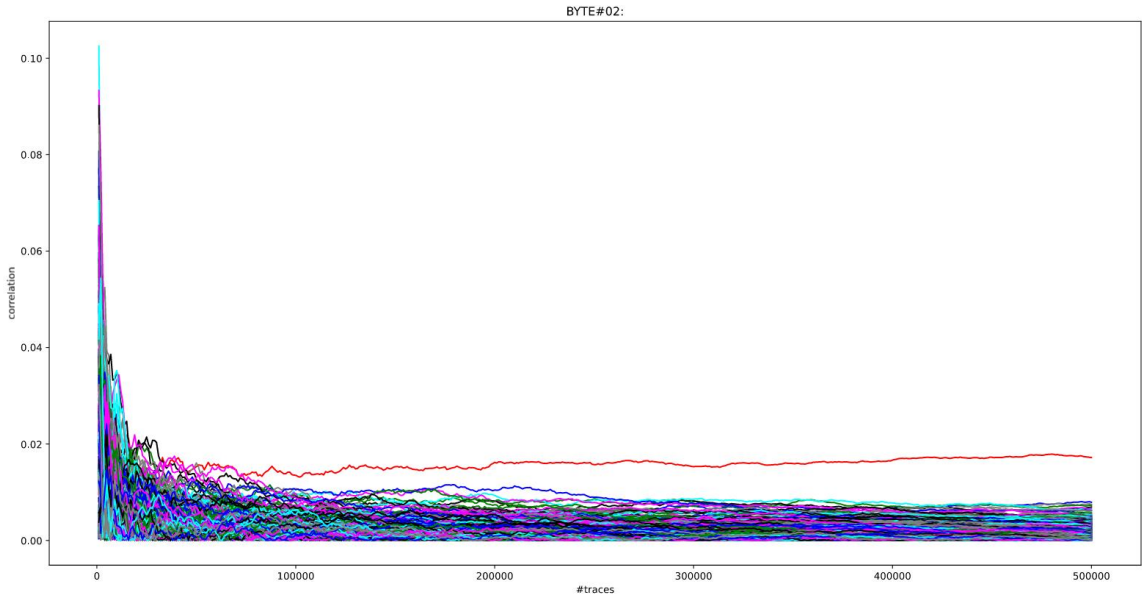

Figure 68 - Evolution of key-byte guess correlation, as the amount of analyzed traces increases: CPA attack on 3rd key-byte (ASIC-based AES).

The fact that the HD leakage between these two intermediate values allowed to extract the key bytes strongly suggests that the hardware-based implementation of AES-128 executed by the cryptoprocessor, at a particular point, overwrites the registers holding the plaintext bytes with the result of the initial SubBytes operation.

Notice how, even though the HD leakage for all byte positions is solely detected at sample 150th, for enough traces, the variation of the EM signal at this sample becomes statistically relevant to such an extent that enables the inference of the key bytes associated with intermediate values that better align with the observed measurements.

4.4.3.2 Linear Regression Analysis

The results obtained from the LRA attack on the same dataset of 500.000 traces surpassed those of CPA in the sense that it enabled the successful extraction of 15 bytes out of the total 16 key bytes with higher confidence levels on average. This is justified by the fact that LRA approximates a leakage model, per key-byte guess, based on the actual observed power consumption, by estimating the contribution of each bit of the intermediate value (in this case, the XOR between the plaintext bytes and the output of the initial SubBytes operation) to the observed instantaneous power consumption of the device. So, for each key-byte guess, the LRA computes the coefficients of the linear equation describing the instantaneous power consumption, in (26), that better fits the observed data. As such, a much more accurate leakage model of the device can be assumed and leveraged in the analysis.

The results from the LRA attack on the 3rd key-byte are presented next. A minimum of 37.000 traces was necessary to infer the correct key-byte for this position. Table 15 displays the four key-byte candidates that yielded the highest R^2 values. The correct key-byte is the first with the highest R^2 value of 0.1962, which is around 2.3x greater than the second highest R^2.

The "confidence" tag on the y-axis of Figure 70 represents the R^2 value. The plot describes how the R^2 value changes as the number of analyzed traces increases. Each individual line corresponds to a key-byte candidate, while the red line, that becomes very distinct from the rest as the number of traces increases, corresponds to the correct key-byte guess.

Table 15 - LRA attack against 3rd key-byte, on sample 150: ASIC-based AES.

key-byte guess	R^2	sample
139_{dec}	0.1962	150
215_{dec}	0.0837	150
214_{dec}	0.0783	150
160_{dec}	0.0745	150

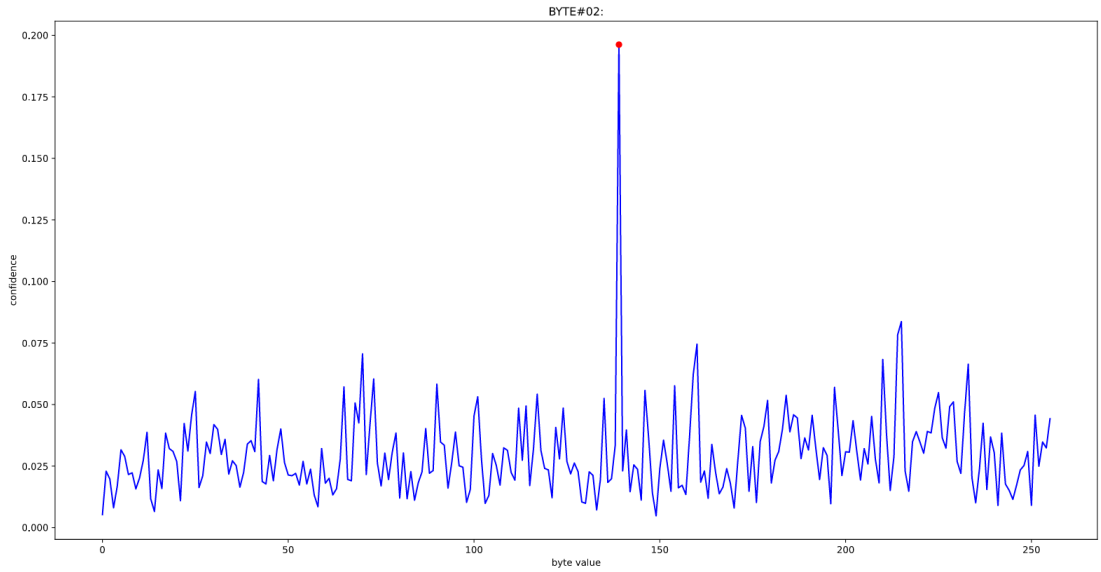

Figure 69 – R^2 value per key-byte guess in sample 150: LRA attack on 3rd key byte (ASIC-based AES).

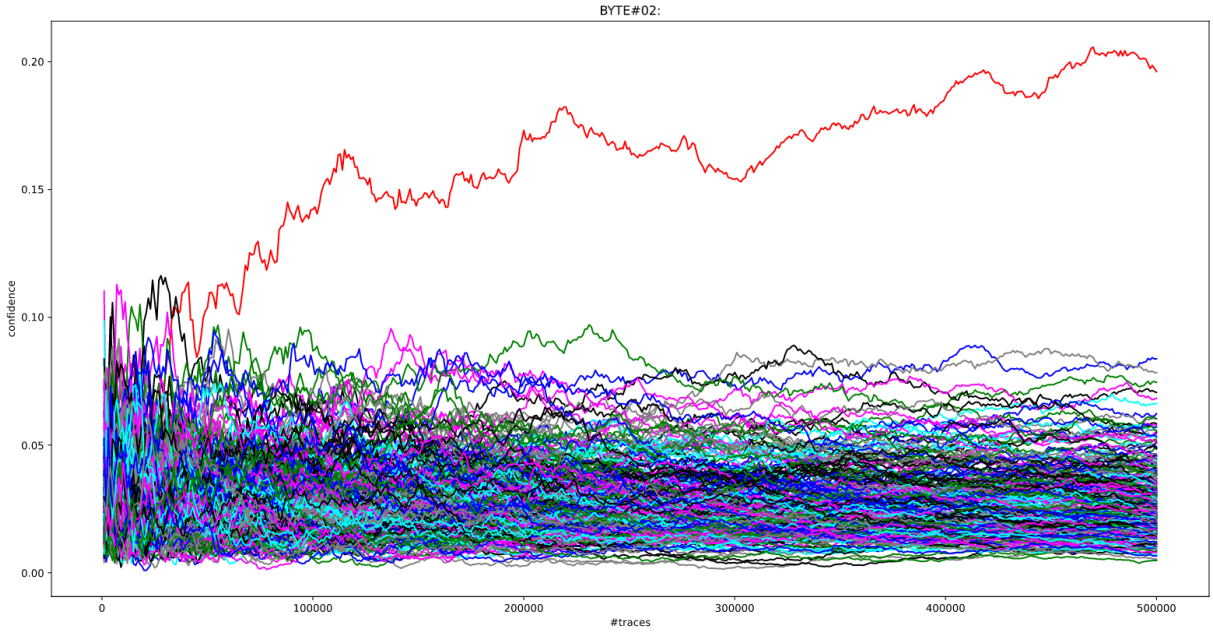

Figure 70 - Evolution of key-byte guess R2 value, as the amount of analyzed traces increases: LRA attack on 3rd key-byte (ASIC-based AES).

The coefficients of (26), obtained from the LRA attack on the first six key-byte positions, and corresponding to the correct key-byte guesses, are described in Table 16. They allow to understand the contribution of each bit, of the intermediate value, to the overall instantaneous power consumption. Coefficient α'_7 corresponds to the LSB bit, the α'_0 to the most-significant-bit, and α'_{-1} to the component of the measured signal that is unrelated to the processing of the data.

Table 16 - LRA coefficients for first six key-byte positions: ASIC-based AES.

		coefficients								
		α'_{-1}	α'_0	α'_1	α'_2	α'_3	α'_4	α'_5	α'_6	α'_7
key-byte position	0	9.88E-1	-5.27E-7	-2.17E-8	2.68E-5	-1.34E-5	-1.74E-6	1.43E-5	8.87E-6	-1.06E-7
	1	9.88E-1	-1.71E-6	3.65E-5	7.63E-6	6.6E-6	1.10E-4	2.33E-4	5.27E-5	1.89E-4
	2	9.88E-1	2.88E-5	1.03E-5	3.92E-5	3.64E-5	2.32E-5	1.53E-5	-8.25E-6	1.07E-5
	3	9.88E-1	2.23E-5	1.18E-5	5.14E-6	1.31E-6	6.69E-6	5.71E-5	1.39E-4	6.01E-5
	4	9.88E-1	1.05E-5	3.61E-6	2.05E-6	4.43E-5	-5.12E-6	-2.49E-5	-1.33E-5	6.07E-6
	5	9.88E-1	-1.3E-5	4.62E-5	1.40E-4	2.87E-4	2.73E-4	2.73E-4	2.81E-4	6.02E-4

4.4.3.3 Mutual Information Analysis

To perform MIA, the HD leakage between the plaintext and the SubBytes output was assumed, similarly to CPA.

A histogram-based approach was implemented for estimating the mutual information value between hypothetical power values and the observed measurements. The analysis was again conducted for the 150th sample since the previous attacks confirmed that its leakage enabled the successful extraction of most secret key bytes. However, with MIA, only 9 out of the total 16 key bytes were correctly deduced from the analysis of the target dataset of 500.000 traces. The poor performance of MIA was already pinpointed in the attack against the software-based implementation of AES-128 due the lower level of confidence provided by the attack as well as the need for an increased number of traces, when compared to the alternative non-profiled attacks, namely the CPA and LRA. In this case, the nine key bytes learned from the attack aren't enough to allow the attacker to easily brute-force the remaining unknown 56 bits of the key.

The potential cause for the failure of the attack might be linked to a poor estimation of the Mutual Information values computed during the analysis, or perhaps the application of the Mutual Information- based distinguisher was unsuitable for this experimental scenario.

Still, the results obtained for the unsuccessful extraction of the 3rd key-byte are presented in Figure 71 and Figure 72. The red dot in Figure 71 and the red line in Figure 72, indicate the correct key-byte guess.

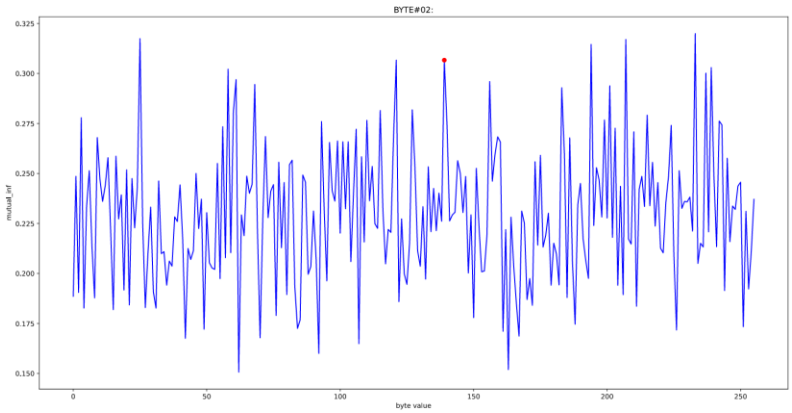

Figure 71 - Mutual Information value per key-byte guess in sample 150: MIA attack on 3rd key byte (ASIC-based AES).

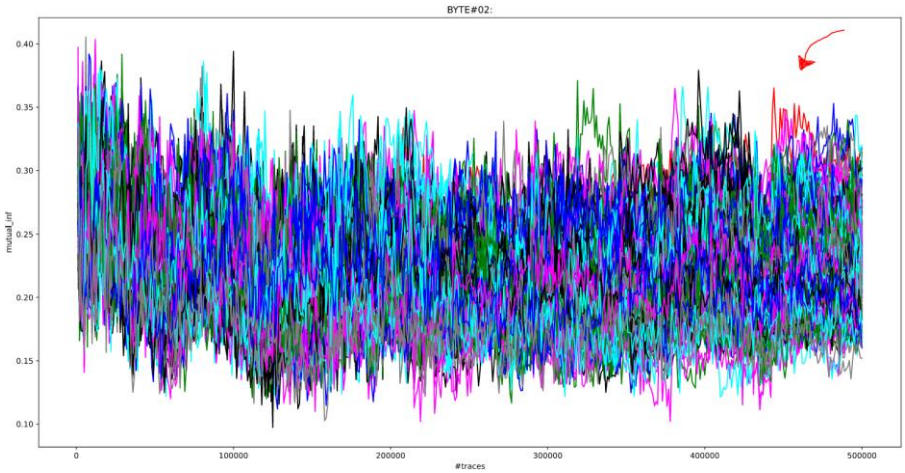

Figure 72- Evolution of key-byte guess Mutual Information value, as the amount of analyzed traces increases: MIA attack on 3rd key-byte (ASIC-based AES).

4.4.4 Profiled attacks

In order to assess whether profile-based attacks facilitated the extraction of the secret key bytes for a smaller set of analyzed traces, comparatively to the model-based attacks, the Template attack was executed. The potential of this type of attack lies in its ability to operate under less restrictive assumptions, thereby relaxing the need of directly assuming a leakage model.

For the profiling phase of the attack, the dataset of 1.500.000 EM traces gathered from the encryption of random known inputs with random known keys, was used.

In the matching phase, 100.000 traces were analyzed from the dataset of 500.000 EM traces that was used to accomplish the previous non-profiled attacks.

Due to the unsatisfactory results obtained from employing the ML classifier in Section 4.3.4.2, only the Template attack was considered in the present section.

4.4.4.1 Template attack

In this attack three possible intermediate values for partitioning the multivariate Gaussian distributions were experimented. These included the (XOR between the first AddRoundKey and SubBytes output), the (XOR between the plaintext and the first SubBytes output), and the (output of the SubBytes operation). As such, three distinct Template attacks were conducted against the ASIC-based AES.

In the profiling phase, the multivariate distribution was calculated for each hypothetical intermediate value based on the observed EM emanation, at sample 150, of the profiling dataset.

Then, in the matching phase, the multivariate Gaussian distribution was calculated for each hypothetical intermediate value based on the observed EM emission at the same leakage point considered in the profiling phase, but now for the target dataset. Ultimately, these were compared with the templates computed in the profiling phase.

As it has been acknowledged in the attack on the software-based AES, constructing the templates during the profiling phase is relatively fast making the acquisition of the profiling dataset the most expensive aspect of the attack.

Unfortunately, all three attempts of the Template attack proved unsuccessful. The most favorable outcome was achieved through the partitioning based on the (XOR between the plaintext and the first SubBytes output) which enabled the extraction of 8 out of the total 16 bytes of the secret key, with the analysis of 100.000 target traces. The results of the Template attack presented in Table 17 and Table 18 refer to the attack based on this partitioning.

In comparison to the LRA attack, the findings from this Template attack indicate that it does not offer significant improvements in terms of minimizing the number of traces required to extract some key bytes.

Since the Template attack performs a multivariate analysis, including more samples in the analysis, for instance samples 149 and 151, could potentially yield more interesting results for other intermediate values.

4.4.5 Conclusion

The results presented previously have demonstrated that the ASIC-based AES-128 encryption implemented on the target device's hardware is vulnerable to PA side-channel attacks. The efficacy of these attacks, particularly CPA and LRA, is evident in their ability to uncover sensitive information processed within the device's cryptoprocessor. Therefore, one should not rely on this hardware-based implementation for sensitive cryptographic applications such as safeguarding encrypted firmware or communication's data, in the absence of proper countermeasures.

Leveraging the hardware-based AES-128 encryption implemented by the dedicated cryptoprocessor of the device is unquestionably tempting for its remarkable speed in performing AES computations. For comparison, the execution time of the cryptoprocessor's implementation is 31.3 times faster than TinyCrypt's software-based AES-128 encryption running on this device. There are other software-based implementations that can be implemented on the device and that are knowingly faster than TinyCrypt's implementation. However, still, none of these implementations come near the efficiency achieved by the device's ASIC-based implementation.

The leakage assessment was a crucial part of the process of finding a setup that allowed to measure the data-dependent power or the EM emanation leaked from the cryptoprocessor, as well as to determine which intermediate values could possibly be observed in the measurements so to conduct the attacks. The effectiveness of each attack, regarding the minimum necessary number of traces to extract each key-byte, can be compared in Table 17. The values are described in a scale of x1000 traces. The confidence levels conveyed by the attacks, for deducing each key-byte, are described in Table 18. Regarding these results, it's worth noting that the non-profiled attacks were carried out on a dataset of 500.000 traces, while the Template attack was performed on 100.000 of those traces. The templates were constructed from 1.500.000 profiling traces.

The success of the attack against the cryptoprocessor's AES-128 implementation is very dependent on the location and positioning of the probe thus, a different EM measurement setup could yield completely different results for the attacks in terms of minimum traces required to extract each key-byte and the corresponding confidence levels. Potentially, by acquiring the trace from a different SoC's surface location the leakage from the processing of an intermediate value associated with a key-byte could be manifestly more evident in the measured EM field and allow for key-byte extracting using less traces. Once again, conducting the acquisition of the EM measurements at a higher sampling rate or in an isolated environment like an anechoic chamber could greatly enhance the results of the attacks described in this section.

In terms of effectiveness for targeting the AES-128 implementation executed by the cryptoprocessor, the LRA attack was deemed the most successful. Under this experimental setting, the analysis of fewer than 60,000 traces resulted in the extraction of at least 12 key bytes. This finding highlights the significant threat posed by LRA in the context of firmware encryption and other cryptographic applications, particularly as further improvements to the experimental conditions may potentially reduce this number even further.

Table 17 - Summary of minimum required traces for each attack (scale of x1000): ASIC-based AES.

		key-byte position															
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	***avg***
attack	CPA	-	1	74	14	-	1	3	13	2	9	112	64	253	89	1	-	**49**
	LRA	208	2	37	6	28	1	4	8	2	4	34	60	245	222	1	-	**58**
	MIA	-	39	-	255	-	3	56	57	14	281	-	493	-	-	44	-	138
	Tem.	-	1	-	5	-	1	3	40	2	3	-	-	-	-	1	-	7

Table 18 - Summary of confidence levels from each attack: ASIC-based AES.

		key-byte position																
		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	***avg***
attack	CPA	~~1.0~~ ~~91~~	2.8 67	2.1 77	2.6 59	~~1.0~~ ~~34~~	4.1 40	3.1 10	3.2 00	3.2 68	2.0 99	2.1 18	1.8 48	1.2 81	1.9 35	2.9 32	~~1.0~~ ~~74~~	**2.3**
	LRA	2.6 12	9.1 26	2.3 44	6.1 87	2.9 59	8.2 53	8.9 21	6.5 69	7.6 25	7.1 36	4.4 14	1.8 71	1.3 22	2.3 32	7.6 55	~~1.0~~ ~~57~~	**5.0**
	MIA	~~1.0~~ ~~25~~	1.5 11	~~1.0~~ ~~08~~	1.3 44	~~1.0~~ ~~86~~	2.8 00	1.3 84	1.1 53	1.5 21	1.2 53	~~1.0~~ ~~89~~	1.0 86	~~1.0~~ ~~03~~	~~1.1~~ ~~03~~	1.2 70	~~1.0~~ ~~12~~	1.3
	Tem.	~~1.3~~ ~~23~~	1.7 90	~~1.6~~ ~~80~~	1.3 54	~~1.0~~ ~~04~~	7.9 55	1.8 56	1.0 84	3.5 08	1.3 53	~~1.2~~ ~~20~~	~~1.4~~ ~~42~~	~~1.0~~ ~~98~~	~~1.0~~ ~~43~~	2.8 07	~~1.2~~ ~~45~~	1.9

4.5 Discussion on protective countermeasures

From the experimental results obtained in the present Chapter it became evident that neither unprotected software-based implementations of AES nor the efficient ASIC-based AES-128 encryption implemented by the device's cryptoaccelerator should be used, in the context of securing encrypted firmware, without proper countermeasures. Therefore, it is highly pertinent to discuss possible protective countermeasures that can be implemented to enable the secure decryption of firmware within the hardware of the target device. The objective of this discussion is to reason about strategies for implementing secure firmware encryption on hardware that inherently lacks SCA protection.

In light of the research on countermeasures explored in Section 3.4, the implementation of countermeasures at the physical and technological levels are not applicable in the current scenario. Even though some physical level countermeasures can introduce an additional layer of complexity to the attacks, they can typically be bypassed with the removal of the associated physical components (e.g., capacitors). Other constraints to the implementation of both physical and technological level countermeasures, in the present context, are the fact that it is not feasible to replace transistor gates nor practical to modify the IC inside the SoC. Therefore, to secure the mechanism of Firmware Encryption with these types of countermeasures, under the current scenario, the device targeted in this dissertation would have to be completely replaced with a different device implementing such countermeasures, which falls outside the intended goal.

One potential viable solution involves using algorithmic level countermeasures. Instead of using algorithms that are known to be vulnerable, the approach would consist of implementing a side-channel resistant implementation. To achieve this, modifications need to be made to the implementation of the cryptographic algorithm. In this instance, the literature suggests that the most efficient approach involves incorporating the combination of masking and hiding techniques. The introduction of these measures can prove effective in dissuading attackers or even rendering certain attacks impractical. This could be due to the added requirement of pre-processing or the substantial number of traces needed to overcome these countermeasures. One must be aware that testing the security of side-channel resistant cryptographic implementations is not a trivial task. It is worth noting that all these measures come with an associated overhead that impacts performance in terms of total execution time and extra power usage. Also, the realization of this type of countermeasures could only be accomplished for software-based implementations running on the device's CPU since modifying the algorithm executed by the cryptoprocessor would require the modification of the underlying ASIC which is unfeasible (recall that ASICs are custom-designed ICs tailored for specific applications that cannot be easily modified or altered after production). Consequently, one would be confined to employing software-based implementations that are, by nature, inherently slower than the ASIC-based implementation provided by the device.

Protocol-based countermeasures are specially interesting in the context of securing firmware encryption. They consist of introducing slight changes to the way the cryptographic application as a whole is achieved, for instance in terms of key management. The solution proposed by Pankaj Rohatgi in [58] provides a solid foundation for reasoning about this type of countermeasures.

Rohatgi's solution focuses on minimizing the amount of information an attacker can learn regarding a cryptographic key, by limiting the number of distinct observations that he can gather for that specific key. It capitalizes on the fact that an attacker can only infer a secret key if he has enough traces associated with the processing of such secret with varying known data. The approach allows for the deployment of a secure cryptographic application that runs on top of an unprotected implementation of AES, while assuring that the keys are secure against key extraction attacks based on power analysis techniques.

In the specific device explored in this document, the countermeasure would enable the use of the efficient, but vulnerable, ASIC-based implementation of AES-128 encryption provided by the device, for secure firmware encryption. However, the effectiveness of this countermeasure heavily relies on the assumption that the attacker cannot make the device encrypt or decrypt arbitrary data using the embedded secret key; otherwise, the countermeasure would fail.

Rohatgi's solution employs simple operations based on hash functions to derive distinct keys. These keys are then used in the encryption and decryption of 16-byte data blocks. This allows to minimize the times a cryptographic key is used, reducing the information an adversary can learn about a key.

Based on the design note in [58] and the insights obtained during the development of this dissertation, the following solution is proposed for discussion purposes only. Please note that it lacks proper validation and should be disregarded for real-world use. The idea is to achieve secure firmware encryption on the unprotected device targeted in the present document:

- The assumption that the device does not encrypt nor decrypt arbitrary data fed by an adversary must be guaranteed by through the implementation of a mechanism, similar to the one of (c)), for validating the integrity and authenticity of the encrypted firmware

image before proceeding to decrypt it. Fault injection attacks may be used to bypass this validation step.

- During manufacturing of a product X, the firmware image is divided into 16-byte blocks. A secret key K is generated by the manufacturer for product X and embedded into the device. Then, to encrypt the firmware image's blocks, the manufacturer selects a block cipher mode of operation that leverages AES-128 encryption algorithm for both encryption and decryption of data, for instance OFB. Note that successful PA attacks against AES in CTR mode have been reported in the literature.
- To encrypt a block i, with $i = 0, \ldots, N$, being N the quantity of firmware image's 16-byte blocks, the manufacturer generates a 128-bit key K_i by using a standard key-derivation algorithm (e.g., *KMAC128*), as recommended by NIST in [59], by doing $K_i = KMAC128(K_{i-1})$, with $K_0 = KMAC128(K)$. Then, each block i is encrypted using AES-128 in OFB mode, to obtain the final encrypted firmware image that is flashed into the device.
- When the device is boot up, the bootloader verifies the authenticity and integrity of the encrypted firmware image. Using its embedded key K, it computes $K_0 = KMAC128(K)$. To decrypt ciphertext block 0, $Y_0 = AES128(IV, K_0)$ is computed using the device's dedicated cryptoprocessor, and Y_0 is XORed to ciphertext block 0 to obtain plaintext block 0. Then, to decrypt ciphertext block i, $Y_i = AES128(Y_{i-1}, K_i)$ is computed and XORed with the ciphertext block i, to obtain plaintext block i. This way, each secret key K_i is only used once in *AES128* and *KMAC128*, therefore the attacker shouldn't be able to learn enough side-channel information to infer K_i.

In the described solution, every time the firmware of the device is updated, the attacker learns additional information about K_i, because he is able to observe the device executing AES-128 using key K_i for the block i of all distinct versions of the encrypted firmware image. Even though this might be negligible, the product manufacturer may consider the possibility of generating K_0 depending not only on the product key K but also on the firmware version. Still, the product manufacturer must be cautious so that this computation of K_0 doesn't leak information on K as the firmware version is updated. To reduce the overhead associated with the proposed countermeasure, the product manufacturer may ponder using a K_i for encrypting n subsequent blocks instead of deriving a distinct key per block.

The focal point of this solution, in contrast to Rohatgi's solution, is the proposal for the adoption of a standardized algorithm for key derivation instead. Both solutions depend on fault injection resistance. Fault injection attacks may be leveraged by an attacker to bypass the firmware signature validation step and consequently enable the attacker to supply the device with arbitrary data which totally renders the countermeasure ineffective.

When considering protective countermeasures, the trade-off between associated overhead and security should be carefully assessed. One notable conclusion drawn from the literature on this subject was that it is not possible to prove the complete security of a system, only the lack of it.

Chapter 5
Conclusion and Future Work

Ensuring the security of electronic devices is of utmost importance in the modern era of technology. Although most devices incorporate standardized secure algorithms, these remain inadvertently susceptible to attacks that exploit the physical properties of these electrical systems. Hence, the exposure to attacks leveraging the analysis of power consumed or EM-field emitted by these devices should always be addressed in the development of systems whose security relies on the execution of cryptographic algorithms.

The background and state-of-art review provided in this document could serve as a point of reference for acquiring the necessary know-how to explore and carry out these attacks in practice, for the purpose of evaluating the vulnerability of devices prior to pondering their use in security-critical applications. The struggles faced and considerations drawn throughout the experimental work elaborated in Chapter 4 can be valuable when targeting devices with relatively similar characteristics.

The outcome of the internship was undeniably positive, with all challenges overcome and concrete results achieved. I participated in many internal presentations to raise awareness on the threat of SCAs while showcasing the results that were obtained for the targeted device and describing experimental procedures. The experience enabled me to broaden my knowledge and skillset significantly, while granting me the chance to dive into the interesting domain of hardware security.

5.1 Future work

Although this work has contributed to the awareness and further understanding of how side-channel leakage in devices can expose them to attack and strategies that can be implemented to mitigate such threats, there are several unexplored paths that hold potential to contribute to the field and complement the present work.

Performing the attacks under improved experimental conditions, with consideration to the conclusions drawn from each targeted implementation (e.g., acquisition with higher sampling rate, isolated and controlled environment for acquiring EM measurements, or others), could provide valuable insights on the minimum number of traces required to extract the secret key with confidence.

Furthermore, implementing and testing the proposed protocolar countermeasure for securing Firmware Encryption and comparing its performance with an algorithmic-based countermeasure that offers a similar level of security could make a significant contribution.

As a complement to the work presented in this dissertation, in the context of securing firmware encryption, it could prove valuable to delve into the realm of Fault Injection attacks and potential strategies for mitigating them. These attacks can be leveraged with malicious intent to bypass integrity and authenticity checks that are in place to prevent attackers from tampering with the firmware and consequently compromise the security of the devices. This

type of research consistently yields benefits that extend beyond the specific objectives (i.e., securing firmware encryption) for which it is conducted.

References

[1] C. Paar, J. Pelzl. Understanding Cryptography: A Textbook for Students and Practitioners. Springer, 2010.

[2] W. Schindler, K. Lemke, C. Paar. A Stochastic Model for Differential Side Channel Cryptanalysis. In J. Rao and B. Sunar, editors, Cryptographic Hardware and Embedded Systems – CHES 2005, volume 3659 of Lecture Notes in Computer Science. Springer, 2005.

[3] Owen Lo, William J. Buchanan & 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, pages 88-107, 2017.

[4] Paul Kocher, Joshua Jaffe, Benjamin Jun. Differential power analysis. Advances in Cryptology — CRYPTO' 99 Lecture Notes in Computer Science, page 388–397, 1999.

[5] Colin O'Flynn, Greg d'Eon. "I, For One, Welcome Our New Power Analysis Overlords: An Introduction to ChipWhisperer-Lint". NewAE Technology Inc, presented at Black Hat USA, 2018.

[6] Stefan Mangard, Elisabeth Oswald, and Thomas Popp. Power analysis attacks: revealing the secrets of smart cards. Springer, 2007.

[7] François Durvaux, Marc Durvaux. SCA-Pitaya: A Practical and Affordable Side-Channel Attack Setup for Power Leakage-Based Evaluations. "Digit. Threat.: Res. Pract., Vol. 1, No. 1, Article 3", 2020.

[8] Kocher, P.C. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In Advances in Cryptology—CRYPTO '96 Proceedings of the 16th Annual International Cryptology Conference, Santa Barbara, CA, USA, 18–22 August 1996; Springer: pp. 104–113, 1996.

[9] Scott Jones, Christophe Tremlet, and Michael Jackson. The fundamentals of secure boot and secure download: how to protect firmware and data within embedded devices. Application Note 6426, Maxim Integrated, 2017.

[10] Mark Randolph, William Diehl. Power Side-Channel Attack Analysis: A Review of 20 Years of Study for the Layman. Dept. of Electrical and Computer Engineering at Virginia Polytechnic Institute and State University, USA. MDPI Journal of Cryptography, 2020.

[11] Paul Kocher, Joshua Jaffe, Benjamin Jun, Pankaj Rohatgi. Introduction to Differential Power Analysis. Springer, 2011.

[12] Owen Lo, William J. Buchanan & Douglas Carson (2017) 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, doi: 10.1080/23742917.2016.1231523

[13] Maaike van Leuken. Comparing Correlation Coefficient and Difference of Means in a Differential Power Analysis Attack. Radbound University, 2019.

[14] E. Brier, C. Clavier, and F. Olivier. Correlation Power Analysis with a Leakage Model. In Proceedings of 6th Workshop on Cryptographic Hardware and Embedded Systems (CHES '04), volume 3156 of Lecture Notes in Computer Science, pages 16–29. Springer Berlin Heidelberg, 2004.

[15] Antoine Chaux. Study and implementation of the Mutual Information Analysis. Master thesis, NXP Semiconductors, University of Bordeaux, 2015.

[16] H. Thiebeauld, G. Gagnerot, A. Wurcker, C. Clavier. SCATTER: A New Dimension in Side-Channel. Université de Limoges & eShard, 2015.

[17] NewAE Technology, 2018. Template Attacks. Available at: https://wiki.newae.com/Template_Attacks.

[18] Marios O. Choudary, Markus G. Kuhn. Efficient Stochastic Methods: Profiled Attacks Beyond 8 Bits. Computer Lab, University of Cambridge, UK.

[19] Julien Doget, Emmanuel Prouff, Matthieu Rivain, Frainçois-Xavier Standaert. Univariate Side Channel Attacks and Leakage Modeling, Extended Version. Cryptology ePrint Archive, Paper 302, 2011.

[20] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, Cécile Dumas. Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database, Long Paper. France.

[21] Tobias Schneider, Amir Moradi. Leakage Assessment Methodology, a clear roadmap for side-channel evaluations. Horst Gortz Institute for IT-Security. Ruhr Universität Bocgum, Germany.

[22] Stan Salvador, Philip Chan. FastDTW: Toward Accurate DTW in Linear Time and Space. Dept. of Comp. Sciences, Florida Institute of Technology, Melbourne.

[23] Hasindu G., Harsha G.. Power Analysis Based Side Channel Attack. Dept. Of Computer Engineering, University of Peradeniya, 2018.

[24] Harshali Zodpe, Arbaz Shaikh. A Survey on Various Cryptanalytic Attacks on the AES Algorithm. International Journal of Next-Generation Computing, 2021, pages 115–123. https://doi.org/10.47164/ijngc.v12i2.202

[25] Silicon Labs. AN0060: Bootloader with AES Encryption. Silicon Laboratories Inc. USA, 2016.

[26] Wikipedia. Block cipher mode of operation. Available at: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation

[27] J. Daemen, V. Rijmen. The Design of Rijndael: AES - The Advanced Encryption Standard. 2002.

[28] FIPS 197. Announcing the Advanced Encryption Standard (AES). National Institute of Standards and Technology (NIST), 2001.

[29] K. Kumar, K. R. Ramkumar, A. Kaur and S. Choudhary. "A Survey on Hardware Implementation of Cryptographic Algorithms Using Field Programmable Gate Array,"

2020 IEEE 9th International Conference on Communication Systems and Network Technologies (CSNT), 2020, pp. 189-194, DOI: 10.1109/CSNT48778.2020.9115742.

[30] K. Gaj, P. Chodowiec. “Cryptographic Engineering FPGA and ASIC Implementations of AES - Chapter 10”. Springer Science+Business Media, LLC, 2009, DOI: 10.1007/978-0-387-71817-0 10.

[31] Open-Source Cryptographic library for AVR microcontroller. Vulnerable RSA implementation at “bigint.c”. Available at: https://github.com/cantora/avr-crypto-lib/blob/master/bigint/bigint.c

[32] WikiChip annotation of the ARM1 die shot by ARM Ltd. Available at: https://en.wikichip.org/wiki/File:arm1_die_shot_%28annotated%29.png

[33] Intel. TinyCrypt AES-128 encryption implementation. Available at: https://github.com/intel/tinycrypt/blob/master/lib/source/aes_encrypt.c

[34] Tim Fisher. What is Firmware? A definition of firmware and how firmware updates work. Feb 14, 2022. Available at: https://www.lifewire.com/what-is-firmware-2625881

[35] Ryan Clancy. Why Firmware Security Matters: Common Vulnerabilities and Best Practices to Stay Safe. Feb 28, 2023. Available at: https://www.eccouncil.org/cybersecurity-exchange/penetration-testing/firmware-security-risks-best-practices/

[36] Aurélien Vasselle, Philippe Maurine, Maxime Cozzi. Breaking Mobile Firmware Encryption through Near-Field Side-Channel Analysis. ASHES 2019 - 3rd Attacks and Solutions in Hardware Security Workshop, Nov 2019, London, United Kingdom. pp.23-32, DOI: 10.1145/3338508.3359571.

[37] Ricardo Jorge do Rosário Maçãs. Evaluating the security of cryptographic systems against electromagnetic attacks. Instituto Superior Técnico de Lisboa. Master’s Thesis supervised by Prof. Dr. Ricardo Chaves, and Prof. Dr. Gonçalo Tavares. November 2017.

[38] Debayan Das, Shreyas Sen. Electromagnetic and Power Side-Channel Analysis: Advanced attacks and Low-Overhead Generic Countermeasures through White-Box Approach. Dept of Electrical and Computer Engineering, Purdue University, USA. Oct 2020.

[39] Lakshminarasimhan Ashwin. Electromagnetic Side-Channel Analysis for Hardware and Software Watermarking. Master thesis, University of Massachusetts Amherst, Dept of Electrical and Computer Engineering. Sept 2011.

[40] Joan Daemen, Vincent Rijmen. The Rihndael Block Cipher. AES Proposal: Rijndael. 2002. Available at: https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf

[41] V. Fischer, M. Drutarovsky. Two methods of Rijndael implementation in reconfigurable hardware. In C. K. Koc¸ and C. Paar, editors, Proc. Cryptographic Hardware and Embedded Systems (CHES’01), LNCS vol. 2162, pp. 81–96. Springer-Verlag, 2001.

[42] Nabihah Ahmad, S.M.Rezaul Hasan. A new ASIC implementation of an advanced encryption standard (AES) crypto-hardware accelerator. Microelectronics Journal, Volume 117, 2021. Available at: https://www.sciencedirect.com/science/article/pii/S0026269221002433

[43] Morris Dworkin. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. SP 800-38A. NIST. Dec 2001. Available at: https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-38a.pdf

[44] Tony R. Kuphaldt. Lessons In Electric Circuits, Volume IV – Digital Circuits, Chapter 3 – Logic Gates. Nov 2007. Available at: https://www.allaboutcircuits.com/textbook/digital/chpt-3/cmos-gate-circuitry/

[45] Agrawal, D., Archambeault, B., Rao, J.R., Rohatgi, P. (2003). The EM Side—Channel(s). In: Kaliski, B.S., Koç, ç.K., Paar, C. (eds) Cryptographic Hardware and Embedded Systems - CHES 2002. CHES 2002. Lecture Notes in Computer Science, vol 2523. Springer, Berlin, Heidelberg. https://doi.org/10.1007/3-540-36400-5_4

[46] Fançois-Xavier Standaert. Introduction to side-channel attacks. In Secure Integrated Circuits and Systems, pages 27–42. Springer, 2010.

[47] Fred de Beer, Marc Witteman, Bartek Gedrojc, Yijun Sheng. Practical Electro-Magnetic Analysis: Non-Invasive Attack Testing Workshop NIAT-2011. Riscure, The Netherlands. Available at: https://csrc.nist.gov/CSRC/media/Events/Non-Invasive-Attack-Testing-Workshop/documents/03_deBeer.pdf

[48] Eric Petters, François-Xavier Standaert, Jean-Jacques Quisquater. Power and electromagnetic analysis: Improved model, consequences and comparison. INTEGRATION, the VLSI journal 40, pag 52-60. ScienceDirect, Elsevier. 2007.

[49] Shan Fu, Zongyue Wang, Fanxing Wei, Guoai Xu, An Wang. Linear Regression Side Channel Attack applied on Constant XOR. International Association for Cryptologic Research. 2017. Available at: https://eprint.iacr.org/2017/1217.pdf

[50] B. Gierlichs, L. Batina, P. Tuyls, B. Preneel. Mutual information analysis: A generic side-channel distinguisher. Cryptographic Hardware and Embedded Systems - CHES 2008, Proceedings of the 10th International Workshop, pp. 426–442, Washington, DC, USA, 10–13 August 2008; Springer: Berlin, Germany, 2008.

[51] Yuanyuan Zhou, Sébastien Duval, François-Xavier Standaert. “Scatter: a Missing Case?”. Constructive Side-Channel Analysis and Secure Design: 11th International Workshop, COSADE 2020, Lugano, Switzerland. pag. 90-103. April 2020. Available at: https://doi.org/10.1007/978-3-030-68773-1_5

[52] Suresh Chari, Josyula R. Rao, Pankaj Rohatgi. Template Attacks. Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA. August 2002. DOI: 10.1007/3-540-36400-5_3

[53] Gilbert Goodwill, Benjamin Jun, Josh Jaffe, Pankaj Rohatgi. A testing methodology for side channel resistance validation. Cryptography Research Inc. Published in NIST.

[54] F. Durvaux and F.-X. Standaert. From Improved Leakage Detection to the Detection of Points of Interests in Leakage Traces, pages 240–262. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. DOI: 10.1007/978-3-662-49890-3 10.

[55] Kris Tiri, Ingrid Verbauwhede. A Dynamic and Differential CMOS Logic Style to Resist Power and Timing Attacks on Security IC’s. UCLA Electrical Engineering Department, Los Angeles. 2004. Available at: https://eprint.iacr.org/2004/066.pdf

[56] Erica Tena-Sánchez, Francisco Eugenio Potestad-Ordóñez, Carlos J. Jiménez-Fernández, Antonio J. Acosta, Ricardo Chaves. Gate-Level Hardware Countermeasure Comparison against Power Analysis Attacks. Appl. Sci. 2022, 12(5), 2390; Fev 2022. Available at: https://doi.org/10.3390/app12052390

[57] Rădulescu A, Choudary MO. Side-Channel Attacks on Masked Bitsliced Implementations of AES. Cryptography. 2022; 6(3):31. Available at: https://doi.org/10.3390/cryptography6030031

[58] Pankaj Rohatgi. Fight side-channel attacks with leakage-resistant protocols. EETimes – Military & Aerospace DesignLine. Sept 2011. Available at: https://www.eetimes.com/fight-side-channel-attacks-with-leakage-resistant-protocols/

[59] Lily Chen. Recommendation for Key Derivation Using Pseudorandom Functions. NIST Special Publication NIST SP 800-108r1. Available at: https://doi.org/10.6028/NIST.SP.800-108r1

[60] Ledger's Advanced Side-Channel Analysis Repository. Accessed in October 2022, available at: https://github.com/Ledger-Donjon/lascar

[61] Qt Side-Channel Analysis tool. Accessed in October 2022, available at: https://github.com/FdLSifu/qscat

[62] ChipWhisperer library. Accessed in November 2022, available at: https://github.com/newaetech/chipwhisperer

[63] SCAred framework. Accessed in November 2022, available at: https://gitlab.com/eshard/scared

[64] Side-Channel Analysis Toolbox. Accessed in November 2022, available at: https://github.com/AISyLab/side-channel-analysis-toolbox

[65] Langer-EMV, RF1 set: Near-Field Probes 30 MHz up to 3 GHz. Accessed on May 2023: https://www.langer-emv.de/en/product/rf-passive-30-mhz-3-ghz/35/rf1-set-near-field-probes-30-mhz-up-to-3-ghz/270