

POLITECNICO DI TORINO

Department of Control and Computer Engineering (DAUIN)

Master's Degree Thesis

Leveraging Deep Learning Techniques for Cross-Family Side-Channel Attacks on 8-bit Microcontrollers



Advisor

Paolo Ernesto Prinetto

Co-Advisor

Samuele Yves Cerini

Candidate

Antonio De Luca

Academic Year 2022/2023

Acknowledgements

I wish to sincerely thank Professor Prinetto for the opportunity to work on this thesis, having me face problems that I never thought I could solve by myself, and for the availability towards me. Special thanks go to his team, that made me feel welcome since the very first day and who I owe many coffees to, more than I can remember.

I would like to express my deepest gratitude to Samuele “Sem” Cerini, who helped me throughout the work and shared passion for the topics (and problems) of this thesis.

I am extremely grateful to my three families: all my family in Lucera to whom I dedicate my whole academic path, my Portuguese family who has adopted me since three years and my friends, that I consider as a further family. You are too many to thank individually, but you know how much you all mean to me.

Finally, to my girlfriend, the one person that made all of this possible, cheering me up in bad times. The only person I feel unutterably close to, even across different countries.

Abstract

The decreasing price of consumer electronics and the rise of the Internet of Things (“IoT”) paradigm are contributing to the massive spread of embedded systems and microcontrollers. These low cost devices, often characterised by limited performance, internet connectivity and low power consumption are now permeating our lives with applications in home appliances, wearable devices and industrial controllers.

Despite being so widely spread, the protection of the data they handle is rarely tackled. Cryptographic algorithms were developed to provide effective protection mechanisms against cyber attackers, although their implementation on physical devices plays an important role in their attack resistance, exposing new vulnerabilities.

The elevated number of embedded devices combined with the importance of sensitive data led to new attack methodologies, known as *Side-Channel Analysis* (“SCA”). SCA consists of a series of techniques that exploit energy leakages (e.g., power, thermal, electromagnetic) to extract secret information about data handled by a device, and decrease the time needed to lead a successful attack by orders of magnitude with respect to brute-force.

In recent years, Deep Learning techniques have been leveraged to achieve improvements in this research field, leading to the rise of *Deep Learning Side-Channel Analysis* (“DLSCA”).

Deep Learning promises to solve some of the problems encountered by classic Side-Channel Analysis techniques, such as the need for human intervention (e.g., features extraction and leakage model selection), and aims at improving the accuracy and the efficiency of the attacks.

The relevance of DLSCA is increasing, as demonstrated by the large quantity of studies carried out since the past mid-decade. Similarly, new challenges for researchers in the field are arising, such as the need to use the knowledge acquired during attacks to build effective defensive mechanisms, the portability of attacks across different devices or “ablation” as a solution to design lighter Deep Learning models.

Among these challenges, the problem of the portability of attacks is tackled only marginally, needing considerable expertise in both Security for Embedded Systems and Deep Learning Techniques.

This work aims at studying the behaviour of Deep Learning models in cross-devices scenarios, exploring their capabilities and limits in new portability contexts, taking advantage of the novelties introduced by DLSCA.

The knowledge gained from devices in a specific group of microcontrollers (known as *profiling devices*, considering *Microchip*’s PIC18XXXK42 family) is exploited to launch an attack on devices from a different group of microcontrollers (referred to as *attack devices*, from *Microchip*’s PIC18XXXK20 family), despite the differences between the two groups.

In the case of Power Side-Channel Analysis, the acquisition of traces from the 8-bit microcontrollers is performed thanks to an open-source toolkit called “ChipWhisperer”,

by *NewAE*. The work relies on the open-source framework “AISY”, developed by the *Delft University of Technology*, and partly on the framework for DLSCA by *eShard*.

The results obtained demonstrate that it is possible to perform cross-family attacks with all the trained models shown in the thesis, although the best performance can only be achieved leveraging an ad-hoc model tuned for the profiling dataset, correctly configured for the attack.

List of Figures

2.1	Diagram of operations performed during encryption in AES-128.	14
2.2	The <code>AddRoundKey</code> operation [1].	15
2.3	The <code>SubBytes</code> operation [2].	15
2.4	The <code>ShiftRows</code> operation [3].	16
2.5	The <code>MixColumns</code> operation [4].	16
2.6	Encryption in ECB mode [5].	19
2.7	Encryption in CBC mode [6].	19
2.8	Encryption in CTR mode [7].	19
3.1	Generic Setup for SCA [8].	22
3.2	Taxonomy of Side-Channel Analysis.	23
3.3	String comparison that leaks timing information.	23
3.4	Effect of input switch in CMOS inverter.	24
3.5	Probe for electromagnetic measurements used in [9].	25
3.6	Heating plate with two temperature sensors for an ATmega162 [10].	25
3.7	Thermal trace that allows to recover secret pin and sign on mobiles [11].	26
3.8	Shunt resistor located upstream of the target device.	29
3.9	Shunt resistor located downstream of the target device.	29
3.10	Implementation of the exponentiation algorithm in RSA [12].	30
3.11	Power consumption trace of the exponentiation algorithm [12].	31
3.12	Power trace collected during AES encryption on XMEGA target.	31
3.13	Simple Power Analysis, annotations on power trace.	32
3.14	Particular of the AES algorithm: the <code>AddRoundKey</code> and <code>SubBytes</code> steps.	32
3.15	Python script of DPA attack on a single bit of the key.	33
3.16	Wrong grouping of traces. The mean does not reveal the contribution of the LSB.	34
3.17	Correct grouping of traces. The mean reveals the contribution of the LSB.	35
3.18	Working principles of a single data bus line.	36
3.19	Current peaks due to bit transfers on bus lines.	37
3.20	Values distribution of Hamming Weight for 4-bit and 8-bit data buses.	38
3.21	Hamming Distance of a register at successive time instants.	38
3.22	Visual representation of correlation between two random variables.	39
3.23	Scatter diagram of Pearson's Correlation Coefficient [13].	40
3.24	Example of trace collected during attack on AES-128 [14].	42
3.25	Reverse CPA with respect to plaintext on collected AES-128 trace [14].	42
3.26	Attack Points in AES [15].	43
4.1	AI, ML and DL.	47
4.2	Neuron in the human brain [16].	48
4.3	Artificial neuron.	49
4.4	The sigmoid activation function [17].	50

4.5	The ReLU activation function [18].	50
4.6	The ELU activation function [18].	51
4.7	AND (A) and XOR (B) truth tables [19].	52
4.8	Example of the MLP architecture.	52
4.9	Example of CNN architecture.	53
4.10	Convolution operation in CNNs.	54
4.11	Rectification in CNNs.	54
4.12	Max and Avg pooling in CNNs.	54
4.13	Example of Autoencoder network.	55
4.14	Effect of Autoencoder on handwritten digit.	56
4.15	Effect of feedback in RNNs.	57
4.16	Scores in classification problem [20].	58
4.17	The Gradient Descent.	59
4.18	Non-convex cost function.	60
4.19	Overfit [21].	62
4.20	Underfit [21].	63
4.21	Supervised Learning for DLSCA [22].	65
4.22	Unsupervised Learning for DLSCA [22].	65
4.23	Confusion Matrix for multi-class classification problem.	67
4.24	Success Rate [12].	68
4.25	Partial Guessing Entropy [12].	69
4.26	ASCAD structure [23].	70
4.27	Example script using the AISY framework [24].	72
4.28	Example script using the SCARED framework [25].	72
4.29	Leakage detection on TinyAES with SCALD [26].	73
5.1	Shortcut Setting for Supervised Learning [22].	74
5.2	Relevance of 8-bit microcontrollers in the North American market [27].	76
5.3	Top view of the HPC Development Board by <i>Microchip</i> [28].	76
5.4	ChipWhisperer-Lite by <i>NewAE</i>	77
5.5	The Capture Board.	78
5.6	Example of code including SimpleSerial functions.	79
5.7	20-pin connector and <i>ChipWhisperer</i> [™] schematic of the GPIO pins [29].	81
5.8	Schematic of the External Clock Generation Module [30].	82
5.9	Interconnections between the <i>ChipWhisperer</i> [™] and PIC18F27K42.	83
5.10	Sketch showing the important interconnections.	84
5.11	AES-128 execution on PIC18F27K42.	85
5.12	Annotations on PIC18F27K42 power trace.	85
5.13	Custom SQI function.	86
5.14	The three scenarios for SQI computation.	87
5.15	Reverse CPA with plaintext bytes 0,1,2,3,4 on PIC18F27K42.	88
5.16	Reverse CPA with key bytes 0,1,2,3,4 on PIC18F27K42.	88
5.17	Expected and actual order in the handling of bytes.	89
5.18	Reverse CPA with key bytes 0,4,8,12 on PIC18F27K42.	89
5.19	Reverse CPA with key bytes 0 and 15 on PIC18F27K42.	90
5.20	Reverse CPA with key bytes 0 and 15 on PIC18LF45K42 and PIC18F46K20.	91
5.21	Definition of a simple MLP in Keras.	93
5.22	Definition of a simple CNN in Keras.	93
5.23	Definition of the first MLP.	94
5.24	MLP tuned for ASCAD-F dataset.	95

6.1	SCA and DLSCA on PIC18F27K42.	96
6.2	basic_MLP with BS=500 and EP=5 on K42_K42 and K42_K20.	98
6.3	Minimum number of traces to reach PGE = 1 with basic_MLP(500,5) on K42_K42.	99
6.4	ascad_MLP with BS=500 and EP=25 on K42_K42 and K42_K20.	100
6.5	Minimum number of traces to reach PGE = 1 with ascad_MLP(500,25) on K42_K42.	101
6.6	gridsearch_MLP with BS=500 and EP=25 on K42_K42 and K42_K20.	102
6.7	Minimum number of traces to reach PGE = 1 with gridsearch_MLP(500,25) on K42_K42.	103
7.1	PGE achieved by all the best configurations of MLPs.	104

List of Tables

4.1	Key used in ASCAD-F.	70
5.1	Key bytes recovered with CPA on PIC18F27K42 in dependence to number of traces.	92
5.2	Key used in both datasets.	92
5.3	Intervals used for the Grid Search.	95
6.1	Number of Epochs and Batch Size investigated in the work.	97
6.2	Average PGE on 16 bytes for <code>basic_MLP</code> on K42_K20.	98
6.3	Average PGE on 16 bytes for <code>ascad_MLP</code> on K42_K20.	100
6.4	Average PGE on 16 bytes for <code>gridsearch_MLP</code> on K42_K20.	102
7.1	Average PGE on 16 bytes for <code>basic_MLP</code> , <code>ascad_MLP</code> and <code>gs_MLP</code> on K42_K20.	105

Contents

1	Introduction	10
2	Advanced Encryption Standard (AES)	12
2.1	Introduction to AES	12
2.2	The Algorithm	13
2.2.1	Encryption Process	13
2.2.2	Decryption Process	16
2.2.3	Key Schedule Algorithm	16
2.3	Cryptanalysis of the AES Algorithm	17
2.4	AES and Hardware Vulnerabilities	17
2.4.1	Hardware and Software Implementations	18
3	Side-Channel Analysis (SCA)	21
3.1	Introduction to SCA	21
3.2	Taxonomy of Side-Channel Analysis Attacks	23
3.2.1	Leakage Sources	23
3.2.2	Access Level	27
3.2.3	Attack Modalities	27
3.2.4	Common Targets	28
3.3	Power Analysis and related Techniques	28
3.3.1	Simple Power Analysis (SPA)	30
3.3.2	Differential Power Analysis (DPA)	32
3.3.3	Correlation Power Analysis (CPA)	35
3.3.4	Attack Points in AES	42
3.4	Countermeasures	44
4	Deep Learning Side-Channel Analysis (DLSCA): State of the Art	46
4.1	Introduction to Deep Learning Side-Channel Analysis	46
4.2	Basic Principles of Deep Learning	48
4.2.1	Artificial Neurons	48
4.2.2	Artificial Neural Networks	51
4.2.3	The Learning Process	57
4.2.4	Adjusting the Networks: Hyperparameters Tuning	63
4.3	Deep Learning applied to Side-Channel Analysis	64
4.3.1	Power Models	66
4.3.2	Metrics	66
4.3.3	Datasets	69
4.3.4	Frameworks	71

5	Development	74
5.1	Motivation and Objectives	74
5.2	Experimental Equipment	75
5.2.1	8-bit MCUs by <i>Microchip</i>	76
5.2.2	The <i>ChipWhisperer</i> [™] Toolchain	77
5.3	Configuration of the Hardware Setup	80
5.3.1	Setting up the Target Board	80
5.3.2	Setting up the Capture Board	81
5.3.3	Board-to-Board Interconnections	82
5.4	Analysis on Traces	85
5.4.1	Simple Power Analysis	85
5.4.2	Where is AES?	86
5.4.3	Correlation Power Analysis	91
5.5	Custom Datasets	92
5.6	Deep Learning Models	93
5.6.1	Neural Networks under Assessment	94
6	Results	96
6.1	Classic SCA and DLSCA on PIC18F27K42	96
6.2	Cross-Family Attacks	97
6.2.1	Basic MLP	97
6.2.2	Ascad MLP	99
6.2.3	GridSearch MLP	101
7	Conclusions	104
7.1	Results Digest and Comments	104
7.2	Future Works	105

Chapter 1

Introduction

Digitalization is a phenomenon that is becoming more and more relevant globally, as demonstrated by reports carried out in recent years [31]. Most data that used to be analog is now being converted to digital, increasing accessibility and concurrently the concern for security. Making a comprehensive approach to security that incorporates all relevant dimensions is essential [32].

In the recent past, techniques such as Side-Channel Analysis helped highlight the weaknesses of electronic devices [12]. Today, in the era of Artificial Intelligence, Deep Learning can unleash the power of Side-Channel Analysis and provide further hints about how cyber criminals lead their attacks [33].

Even though studies carried out in this research field demonstrated to be very promising, many of them lack important details about the experimental setup or used data, making the experiences irreproducible. This is in contradiction with the scientific method proposed by Galileo Galilei and causes confusion in the academic community, since results often claim to outperform the current State of the Art without allowing researchers to verify them [22].

Continuing along the path outlined by researchers, this thesis work emphasizes the reproducibility of experiments using frameworks from the State of the Art [34]. The conducted studies enable the understanding of the level of generalization Deep Learning models can achieve during cross-family attacks. The knowledge gathered can be leveraged to eventually build, in the future, increasingly effective protection mechanisms.

The thesis is structured as follows:

- Chapter 2 summarizes all the necessary information regarding the Advanced Encryption Standard (AES) and the software implementation chosen for the study;
- Chapter 3 gives an overview of what *Side-Channel Analysis* consists of, the main attack approaches and the latest countermeasures developed;
- Chapter 4 describes the main topic of the work, *Deep Learning Side-Channel Analysis*. Its main features are highlighted, as well as its pros and cons with respect to traditional Side-Channel Analysis. Mentions to the most relevant frameworks and datasets in the research field are present;
- Chapter 5 represents the development flow followed during the work. In the first part of the chapter, the experimental setup is presented for the sake of the reproducibility of the experience, as well as the firmware developed for the purpose. The section also focuses on *ChipWhisperer*[™], the tool used during the development.

In the second part of the chapter, the Deep Learning models used to attack the previously generated datasets are discussed;

- Chapter 6 reports the results obtained in chapter 5;
- Chapter 7 presents significant conclusions about the thesis, as well as possible new ideas to explore in the future and improvements to the current work.

Chapter 2

Advanced Encryption Standard (AES)

2.1 Introduction to AES

As early as in 1977, the *National Institute of Standards and Technology* (NIST) chose the Data Encryption Standard (DES) as the cryptographic algorithm to be used for non confidential data.

DES is a symmetric-key algorithm, it uses the same key both for encryption and decryption. It is a block cipher ¹ based on a Feistel Network ², considered problematic due to multiple factors, such as the short size of the key (56 bits only), the complexity of the structure it is based on and the suspects that involved the *National Security Agency* (NSA) about the presumed insertion of a backdoor³.

Since enlarging the key size would make the algorithm safer but at the same time cause longer duration for encryption and decryption [35], in 1997 a call for a new algorithm that could replace DES took place. The winners of the contest were two Belgian cryptographers, Joan Daemen and Vincent Rijmen, that proposed an implementation of what would be later called *Advanced Encryption Standard* (AES) [36].

AES, in a similar way to its predecessor, is a symmetric-key block cipher. It addresses all problems DES is affected by, since:

- The key size is larger than the one DES handles, allowing key sizes of 128, 196 or 256 bits;
- The structure is not based on a Feistel Network, but rather on Shannon's *Confusion and Diffusion* principle. It states that a secure algorithm should create confusion, obfuscating the relationship between the plaintext and the corresponding ciphertext, and diffusion, spreading the plaintext over the whole ciphertext in order to make it harder for attackers to analyze the encrypted data;
- AES was chosen after a public call for a contest took place. This process has a double advantage, since it makes the choice more transparent (thus, reducing the probability of inserting a backdoor in the algorithm) and allows for review of researchers, who can report possible faults in the algorithm.

¹A block cipher is a cryptographic algorithm that operates on a sequence of bytes, called *block*, rather than on a stream of bits.

²The Feistel network operates on each data block by dividing it into two halves, performing operations on each half and then recombining them.

³In cybersecurity, a backdoor is a method of bypassing normal authentication and security controls in a system, allowing unauthorized access to that system.

In 2000 the NIST formally adopted AES as a standard, and consequently companies have incorporated software and hardware implementations of the algorithm in their products.

2.2 The Algorithm

The AES algorithm operates on fixed-sized blocks of 16 bytes of data, arranged into a 4x4 matrix and referred to as *State*. The key size, 128, 192 or 256 bits, determines the number of rounds in the encryption process: 10 rounds for AES-128, 12 rounds for AES-192 and 14 rounds for AES-256. At each round a *round key* is created out of the original key thanks to the *key schedule* algorithm.

The **AddRoundKey**, **SubBytes**, **ShiftRows** and **MixColumns** steps constitute the basic operations used both for encryption and decryption in AES, although their order varies in the two processes.

As stated in Section 2.1, AES follows Shannon's principle of Confusion and Diffusion. Confusion is achieved by the **SubBytes** and **MixColumns** steps, which aim to make the relationship between the plaintext and the ciphertext complex and non-linear⁴. Diffusion is achieved by the **ShiftRows** and **MixColumns** steps, which aim to distribute the effect of each change in the plaintext throughout the entire ciphertext [36].

2.2.1 Encryption Process

Encryption protects information from unauthorized access or manipulation, as it transforms data from a readable form (plaintext) to an unreadable one (ciphertext). The block diagram of encryption in AES-128 is depicted in Figure 2.1.

At first, an **AddRoundKey** operation is performed. Then, the encryption process iterates through all the four basic blocks $N-1$ times, where N is the number of rounds in the AES version taken into account (e.g., 9 out of the 10 rounds in AES-128). The last round is slightly different from the previous ones since it skips the **MixColumns** operation. Skipping it makes the algorithm faster and does not change the security of the encryption process, as it consists in a linear transformation that can be reversed easily by an attacker [36].

⁴In cryptography, a non-linear operation refers to a mathematical function that cannot be expressed as a linear combination of its input variables.

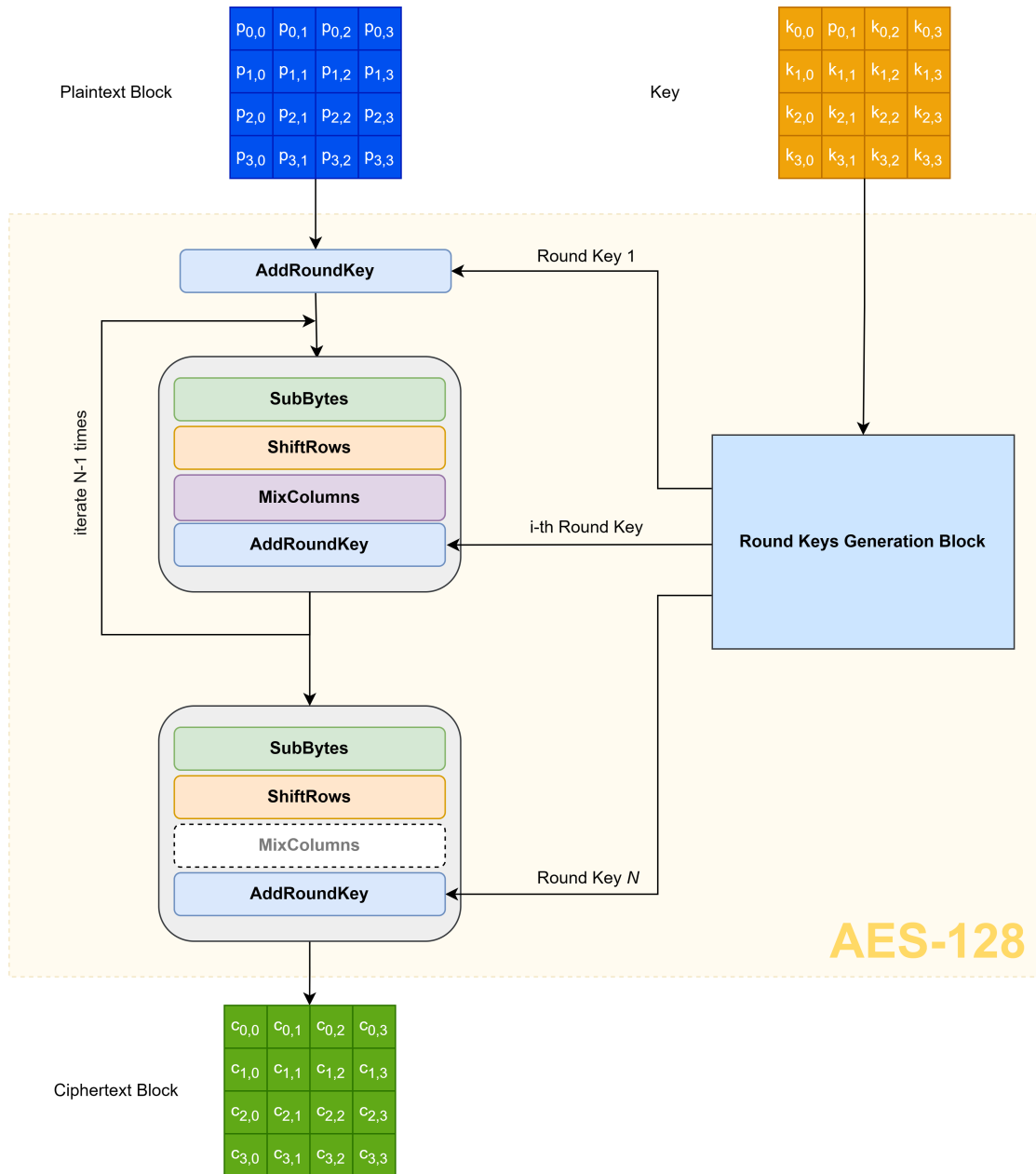


Figure 2.1: Diagram of operations performed during encryption in AES-128.

The AddRoundKey Operation

During this linear operation the *State* is XORed with the *Round Key*, as shown in Figure 2.2.

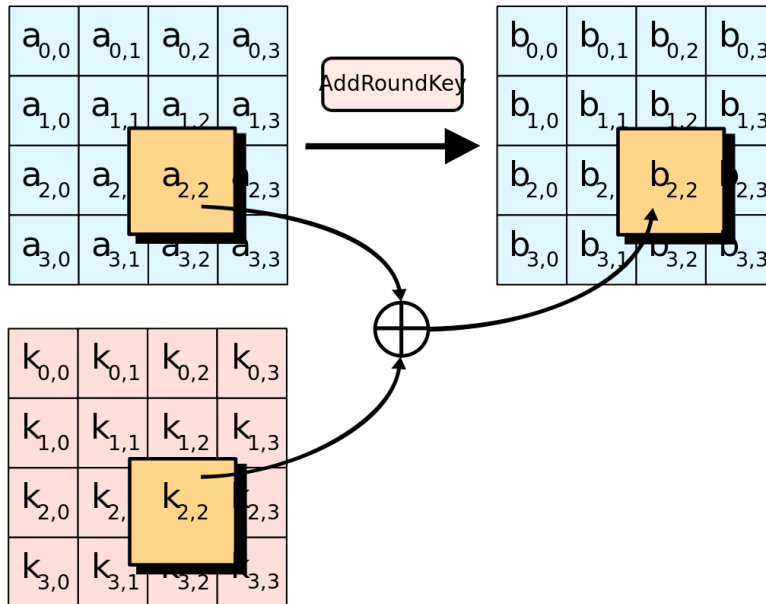


Figure 2.2: The AddRoundKey operation [1].

The SubBytes Operation

This operation performs the substitution of each of the bytes in the *State*, thanks to a Look-Up Table (LUT) known as **S-Box**. The **S-Box** mixes the plaintext in order to make attacks harder. It is designed to be highly non-linear and resistant to cryptanalysis [37]. The operation is illustrated in Figure 2.3.

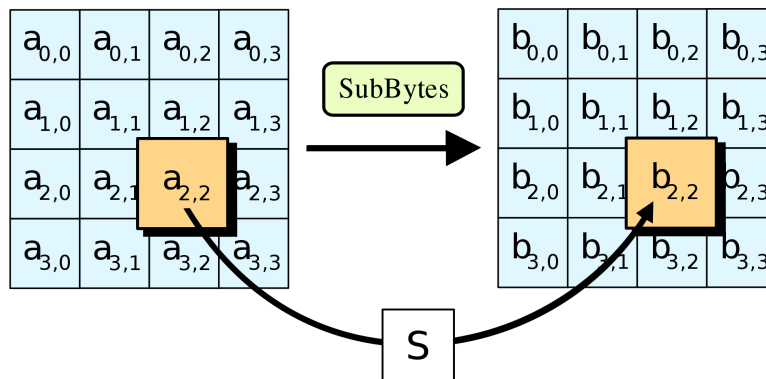


Figure 2.3: The SubBytes operation [2].

The ShiftRows Operation

This linear operation, as the name suggests, shifts the rows of the *State* by a quantity of positions to the left equal to the index of the row itself in the matrix (e.g., row 0 will keep its position in the matrix, row 1 will shift left by one position, and so forth). Figure 2.4 depicts its behavior.

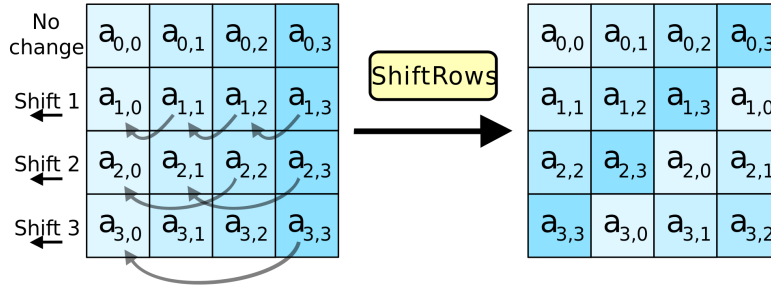


Figure 2.4: The ShiftRows operation [3].

The MixColumns Operation

During this linear operation each column in the *State* matrix is multiplied by a constant polynomial $c(x)$, as illustrated in the Figure 2.5. This step further increases the diffusion of the plaintext throughout the cipher.

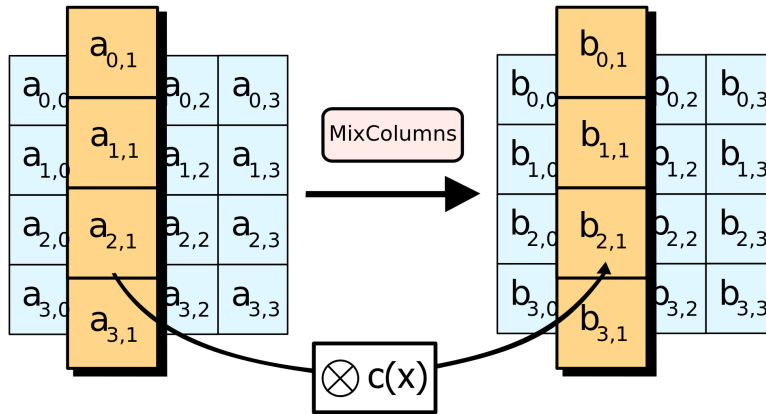


Figure 2.5: The MixColumns operation [4].

2.2.2 Decryption Process

Since AES is a symmetric-key cipher, decryption uses the same key of the encryption. The same four basic blocks described in the previous Section are employed in a reverse order, although three of them (SubBytes, ShiftRows, MixColumns) are slightly different with respect to those employed in the encryption, due to the inverted mathematical operations that take place in this process. On the other hand, since AddRoundKey consists of a XOR operation and is intrinsically invertible, the block is unchanged. Thus, the sequence of operations in AES decryption is:

1. AddRoundKey;
2. InvMixColumns;
3. InvShiftRows;
4. InvSubBytes.

2.2.3 Key Schedule Algorithm

The algorithm produces each round key and is composed of multiple steps:

1. **Copy of the Key.** The original key is copied and used for the first round, so that the encryption process can start immediately. For **AES-128** the first round key is an exact copy of the original key, while in **AES-192** and **AES-256** the first round key is equal to the original key for the first 128 bits;
2. **Key Expansion.** The copy of the key is expanded thanks to mathematical operations and a new set of round keys, each of 16 bytes, is derived (10 keys for **AES-128**, 12 for **AES-196** or 14 for **AES-256**). They are not yet ready to use, as their entropy is still too low;
3. **Key Schedule Core.** This step operates on each word of the copy of the key (i.e., the one generated at the first step of this algorithm). A series of both linear operations, such as XOR of each word by a round constant, and non-linear transformations, such as substitutions through a Look-Up Table, takes place.

2.3 Cryptanalysis of the AES Algorithm

Cryptanalysis is the study of the methods that aim to analyze and eventually break cryptographic ciphers, leveraging their weaknesses in order to obtain decrypted information without knowing the key. The main cryptanalysis methodologies are listed below.

Brute-force attack, which consists in attempts that explore the whole key space. The number of possible combinations that a cyber attacker would have to try before breaking a key in **AES-128** is on average $\frac{1}{2}2^{128}$ times. If a computer could try 2^{40} attempts per day, it would take about 425 sextillion years to guess the key. In order to give figures that show the unfeasibility of a brute-force attack, the sun is expected to run out of hydrogen in “only” 5 billion years. The number of guesses is further increased in **AES-192** and **AES-256** ($\frac{1}{2}2^{192}$ and $\frac{1}{2}2^{256}$, respectively).

Linear cryptanalysis, which aims to find a linear function that approximates the relationship between the plaintext, ciphertext and key of the encryption algorithm. By comparing the predicted key with other known plaintext/ciphertext pairs, the cryptanalyst can refine their approximation and eventually discover the correct key. The process of parameters tuning in the linear function can be both time-consuming and resource-intensive. Moreover, **AES** was designed in order to be resistant against this kind of attacks [36].

Differential cryptanalysis. It is a statistical analysis which is based on the observation of differences in the ciphertext when changes are made in the plaintext. It is proved to be effective with **AES**, although it requires a large amount of data, as well as high computational power [38].

Side-Channel Analysis. This class of attacks exploits information leaked by devices that implement **AES**. Its main drawback stems from the need for physical access, at least for the first phase of the attack. It is the main subject of the next Chapter, where it is extensively examined.

2.4 AES and Hardware Vulnerabilities

Although no serious threat to the mathematical structure of the **AES** algorithm has been found out yet, its implementation on physical devices has a huge impact on its security.

Custom versions of any cryptographic algorithm are never recommended, since the creator may lack expertise and are likely to be less secure than well-established ones.

The public contest announced by the NIST for the creation of **AES** demonstrates that widespread cryptographic algorithms cannot be developed in secret, both for the risk of insertion of backdoors and because of a continuous peer-review on the robustness of the algorithm, as new weaknesses can be found out.

In the wide variety of implementations, there exists a clear distinction between software and hardware ones, due to the advantages they feature.

2.4.1 Hardware and Software Implementations

Hardware implementations are often written in Hardware Description Languages such as VHDL or Verilog. These solutions guarantee:

- Faster performance, since this type of specialized hardware can perform encryption and decryption operations significantly faster than conventional, general-purpose CPUs;
- Reduced power usage, as these special-purpose solutions allow designers to more closely match available hardware to more stringent power consumption constraints.

Many software implementations are available as well, for different programming languages, applications and target devices [39]. Their main advantages are:

- Flexibility, as software **AES** can be easily modified and updated to address security weaknesses or changes in encryption standards;
- Portability, since it can run without modifications on multiple devices, from micro-controllers to general purpose computers;
- Lower cost with respect to hardware **AES**, as they feature a more cost-effective development (e.g., no need for manufacture and assembly of physical components) and can be distributed on a massive scale instantaneously.

TinyAES: Software AES for Embedded Systems

TinyAES [40] is a portable software implementation of **AES** for embedded systems, written in C language and compliant to **AES-128**, **AES-196** and **AES-256**. It can work with multiple modes of operation, such as Electronic Codebook, Cipher Block Chaining and Counter Mode.

Electronic Codebook (ECB). It is the simplest and least secure operation mode for **AES**. It works on data blocks individually, thus identical plaintext blocks will result in identical ciphertext blocks, which can lead to patterns in the encrypted data that can be exploited by attackers. ECB is illustrated in Figure 2.6.

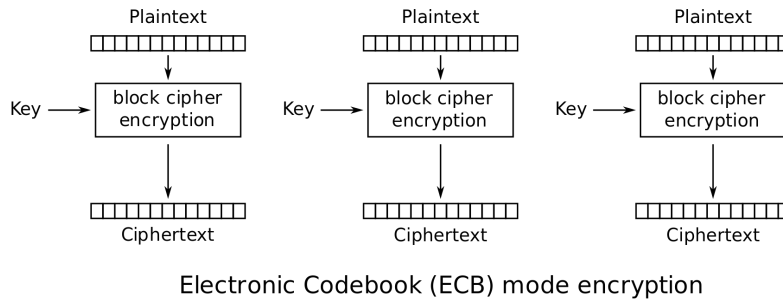


Figure 2.6: Encryption in ECB mode [5].

Cipher Block Chaining (CBC). Each block of data is encrypted with the ciphertext of the previous block, providing a form of feedback, as depicted in Figure 2.7.

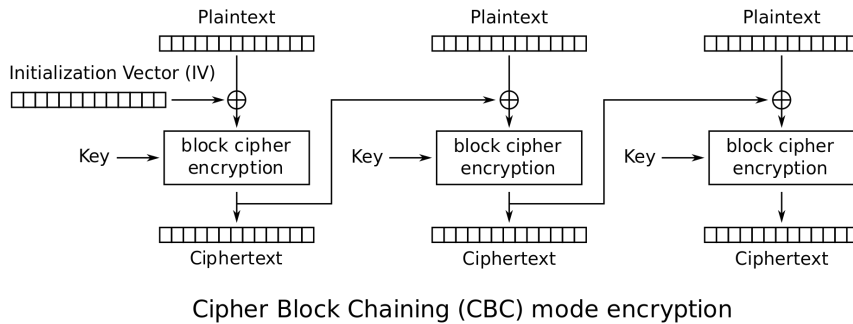


Figure 2.7: Encryption in CBC mode [6].

Counter (CTR). It works on single data blocks, just like ECB mode, but in combination with a counter value to ensure that each block of data is encrypted with a unique key, as in Figure 2.8.

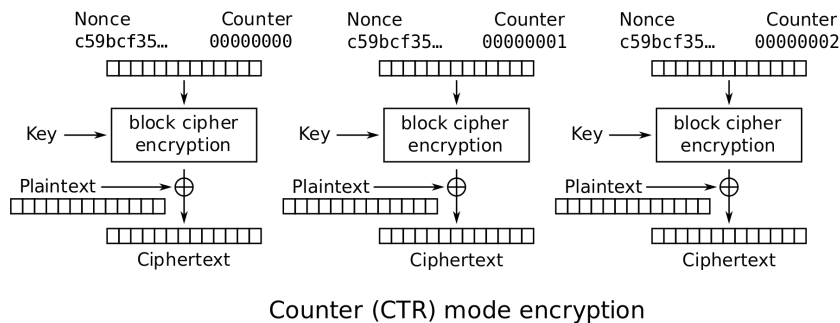


Figure 2.8: Encryption in CTR mode [7].

TinyAES provides a simple Application Programming Interface (API) that exposes functions as simple as `AES_ECB_encrypt()` or `AES_ECB_decrypt()`, if the user intends to operate in ECB mode.

Flexibility is guaranteed since this implementation allows changes to fundamental structures of AES, such as the S-Box.

The size in memory after the code is compiled represents a crucial feature of **TinyAES**, since it was developed for embedded systems. When compiled for ARM MCUs, it uses less than 200 bytes of RAM and 2KBytes of ROM, which is a reasonable amount of memory for most providers, as listed in the websites of major suppliers [41] [42]. Furthermore, **TinyAES** is verified against the example vectors provided by the NIST [43].

Chapter 3

Side-Channel Analysis (SCA)

3.1 Introduction to SCA

The high level of protection provided by cryptographic algorithms currently built into everyday products can be compromised by vulnerabilities at different levels of abstraction. Since protection from attacks is only guaranteed when even the weakest link in the system is secure, a holistic approach is needed capable of addressing every possible weakness.

In contrast to attacks that exploit logical vulnerabilities in cryptographic algorithms, such as cryptanalysis, *Side-Channel Analysis* (SCA) targets the *physical implementation* on which the algorithms are developed. It analyzes unwanted (but also inevitable) information leaked by devices during their normal operations (e.g., heat, power consumption, and so forth) to gain knowledge about the handled data.

The first attempts to leverage side-channel leakage for information disclosure date back to 1914, when crosstalk¹ between telephone wires was used to reveal secret messages in the battle of Flanders. Later on, in the 1960s, the Prime Minister of the United Kingdom ordered surveillance on the French ambassador, leveraging acoustic leakages of telegraphs. Concomitantly, the spy fever broken out during the Cold War forced *NATO* to work on a new standard that could present mandatory countermeasures against side-channel analysis to be deployed in devices, the *TEMPEST* program [44].

The 1990s saw increasing concern for the academic research field and consequently the rise of new hacking strategies able to exploit power and timing information to carry out attacks against digital systems [45]. In 2015, Goller *et al.* described how to extract private keys from smartphones at a distance just observing radiofrequency emissions [46].

In 2018, the “Spectre” and “Meltdown” attacks forced *Intel* and *ARM* to redesign their unreleased CPUs and provide software patches for those already available in the market, resulting into a significant impact on the cost and performance of the devices and on the reputation of the two companies [47].

It appears clear that the strength of Side-Channel Analysis is the capability to adapt to different attack scenarios, exploiting a vast number of leakage sources to hack an even wider variety of devices.

When applied to hardware security, the generic flow of modern attacks is pretty standard, illustrated in Figure 3.1, and can be applied to different targets.

¹In electronics, crosstalk refers to the phenomenon of unwanted transfer of signals between two or more channels in a communication system, mainly due to electromagnetic interference.

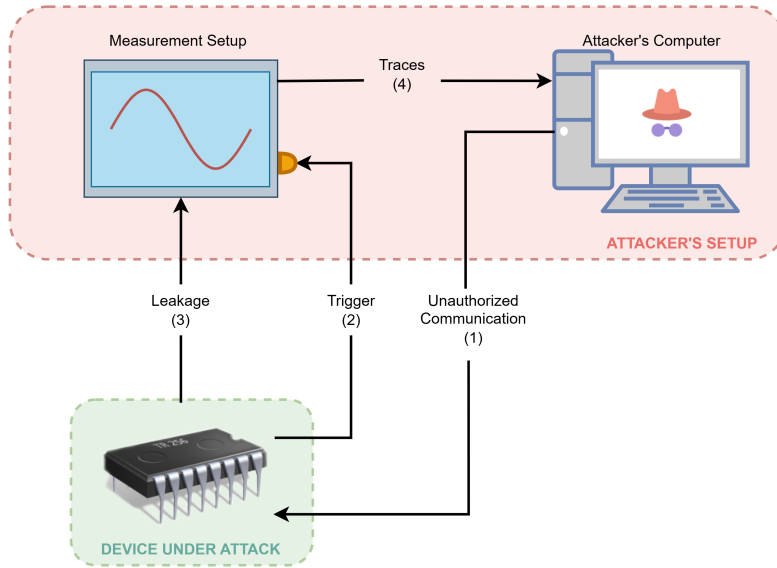


Figure 3.1: Generic Setup for SCA [8].

At first, the attacker builds a measurement setup, through which the chosen source of leakage is made observable. The core instrumentation depends on the chosen leakage source and can consist of a simple electric resistor for attacks on power consumption or more complex heat sensors in the case of thermal analysis.

The attack begins with malicious communication of plaintext to the target (1). During this phase, the target device is forced to perform encryption (or decryption) operations not authorized by the legit owner.

Once the system starts operating, the measurement setup is triggered and starts probing (2). The trigger signal is generally present as the attacker aims at keeping only meaningful temporal cuts of the signal, which often coincide with the start of the cryptographic operation. Consequently, the leakage signal is recorded (3). Traces² are then available for the attacker (4), who will be able to examine them.

²In the context of Side-Channel Analysis, a *trace* is a collection of samples deriving from the measurements performed by the attacker.

3.2 Taxonomy of Side-Channel Analysis Attacks

During the years several works have attempted to provide a complete taxonomy of Side-Channel Analysis, and demonstrated that SCA can be classified along multiple orthogonal axes, as reported in the graphic below.

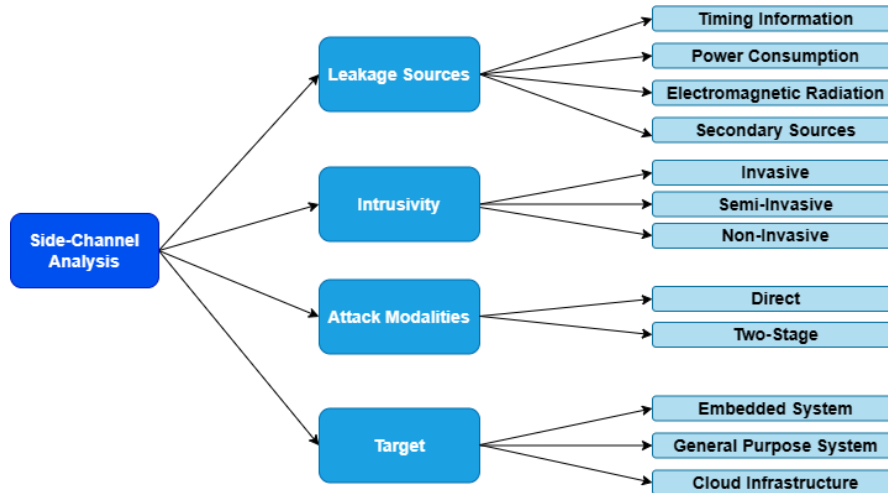


Figure 3.2: Taxonomy of Side-Channel Analysis.

3.2.1 Leakage Sources

Leakage sources represent the unintended signals disclosed into the environment by devices during their normal operations. They often require diverse instruments to be detected and have different origins, from mechanical stress to sound waves. The most frequent ones in literature are listed below.

Timing information. The attacker can gain knowledge about data handled by a device thanks to different execution paths in the code, such as unbalanced conditional branches. Figure 3.3 illustrates a Python function prone to timing attacks.

It performs a character-wise matching and returns earlier from execution if the two elements currently extracted from the two strings are not the same. Such code causes the execution time to be dependent on the nature of the strings themselves, and in the case where they are not identical, it even provides temporal information about *when* they cease being equal.

It is important to notice that on the opposite to what happens with incorrect implementations due to software bugs (e.g., bugs stemming from wrong assumptions, such as underestimating the size of variables), the code fragment is *functionally* correct, as it performs the operations it is meant to.

```
1 # Version 1 - leaks timing information
2 def string_compare(s1,s2):
3     for i in range(len(s1)):
4         if s1[i] != s2[i]:
5             return False
6     return True
```

Figure 3.3: String comparison that leaks timing information.

Power consumption. The origin of this leakage source stems from the basic working principles of digital circuits and it is due to their non-ideality. The typical behavior of a CMOS inverter is depicted in Figure 4.1. The yellow line is the inverter's input, the green one represents its output and the pink line shows the corresponding current consumption.

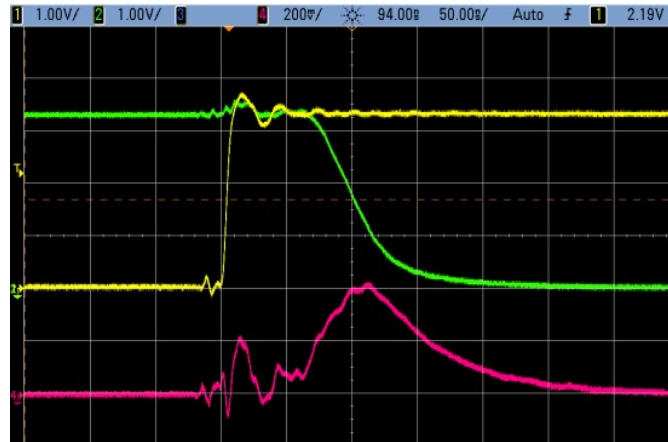


Figure 3.4: Effect of input switch in CMOS inverter. In yellow, the inverter input, in green the inverter output and in pink the deriving current peak [48].

When the inverter's input changes, a simple flip in the value of the output would be expected to take place. However, due to its non-ideality, during the switch both transistors are closed for a short time originating a current that flows from the power supply to the ground, known as *short-circuit* current. Consequently a peak in its current consumption is observable [48].

Electromagnetic radiation. They stem from the energy that is propagated through space by electromagnetic waves, created by the movement of charged particles. They are powerful attacks, as on some chips the equipped radio transmitter can broadcast electromagnetic waves carrying information about handled data, revealing it at distance (e.g., "screaming channels" class of attacks demonstrated that it is possible to infer secret data at 15 meters from the victim device [49]). On the other hand, the needed laboratory instrumentation is generally expensive and the accuracy for probes placement should be high: misplacements that might appear negligible can have a high impact on the measurements. An example of electromagnetic measurements is in Figure 3.5.

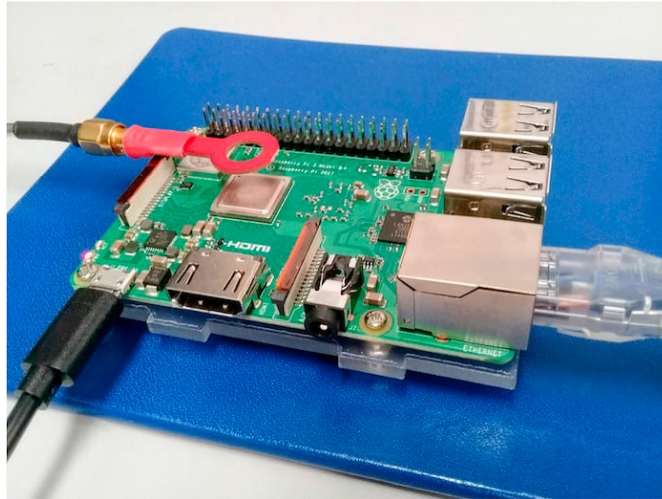


Figure 3.5: Probe for electromagnetic measurements used in [9].

Temperature. Chips do not dissipate a constant amount of heat, but rather a quantity that depends on the operation they are performing. Figure 3.6 displays rear-side and front-side temperature measurement of a microcontroller.

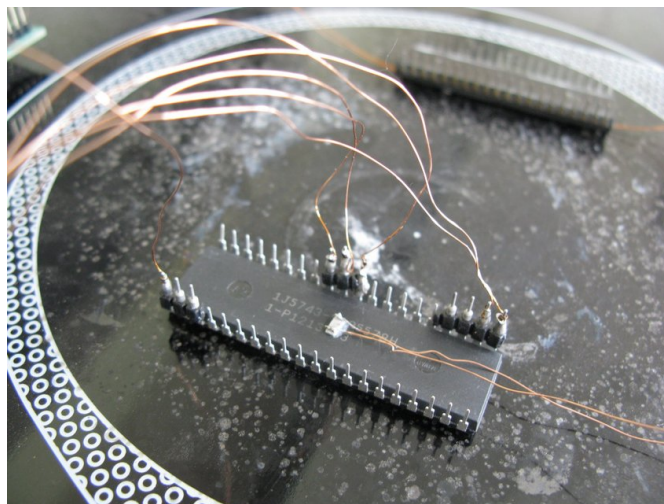


Figure 3.6: Heating plate with two temperature sensors for an ATmega162 [10].

Another remarkable example that demonstrates how this leakage source can be exploited is represented by mobile lock screens. The user is often asked to provide a secret pin or draw a sign with their finger to unlock the device. These operations leave a footprint in the environment, as a heat trace can be used to recover the secret pin or sign, as illustrated in Figure 3.7.

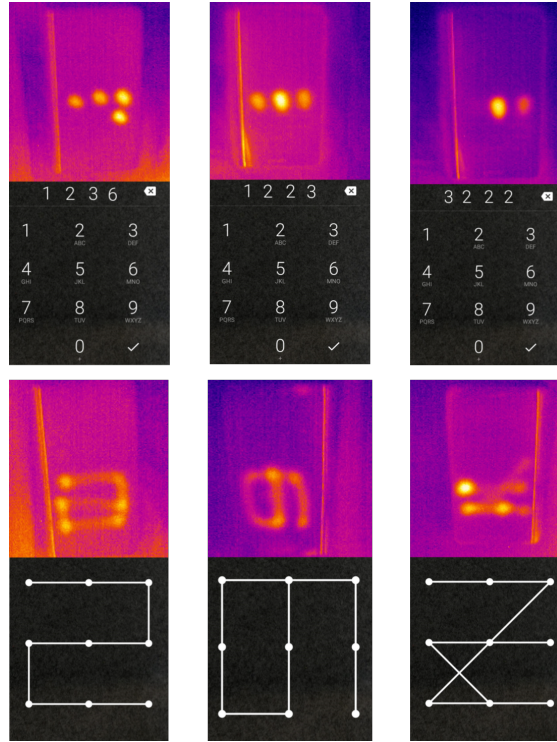


Figure 3.7: Thermal trace that allows to recover secret pin and sign on mobiles [11].

Interestingly, not only the attacker can retrieve the position where fingers stepped, they can also recover the order of the sequence (both for the pin and for the sign), as the thermal trace fades with time, giving hints about the most and least recent positions in the sequence.

Environmental factors, such as humidity, can have a high influence on the quality of measurements [50].

Acoustic noise. Devices emit sound waves when operating, due to the mechanical stress they are subject to. By analyzing the amplitude and frequency content of the acoustic signal, an attacker can infer the timing and nature of the device’s internal operations and disclose information [51].

Optical channels. Due to the working principles of digital circuits, the moving flux of charges causes a thermal increase and a consequent variation in the infrared spectrum, that can be leveraged by attackers to infer the current operation in progress [52]. Furthermore, optical channels can be exploited to “read” the reflection projected by the computer screen on clothes or face of the user thanks to a telescope, a photomultiplier tube and an image-processing software [47].

Although the mentioned leakage sources might appear different from each other, they are often correlated. It is the case of electromagnetic radiations and power consumption, as stated by the *Biot–Savart* law³, or heat and acoustic noise, due to power dissipation and mechanical stress of chips.

The sources mentioned above represent the predominant ones in literature, although many more exist. For instance, the rocking motion of a mobile when the user is typing is

³In physics, the Biot–Savart law is an equation describing the magnetic field generated by a constant electric current.

caught by the system’s accelerometer and can reveal information about the text entered [47].

Additionally, thanks to **social side channels**, it was possible to detect a forthcoming military operation just by noticing an increase in the number of pizzas ordered in a given period at the Pentagon [47].

3.2.2 Access Level

According to Anderson *et al.* [53], the access levels are of three kinds.

Invasive. This class of attacks targets the package and the passivation layer⁴ of chips with advanced techniques, such as decapsulation, depackaging and rebonding [12].

Decapsulation consists in dissolving the packaging material with chemical reactions, usually with acid substances (e.g., sulfuric acid). Once the hole is created the interconnections of interest are made observable.

Depackaging is another process that involves chemical reactions, as the chip is completely immersed into acid until it is free of its package.

Rebonding allows the attacker to restore the functionalities of the device, manually attaching wires that used to connect the package to the chip.

Invasive attacks are generally expensive and the owner of the targeted device is able to detect them, as the physical integrity of the chip is violated. On the other hand, the attacker has maximum freedom to place probing needles and observe a large variety of signals.

Semi-Invasive. Here the physical integrity of the chip is intact, since the package and the passivation layer are not damaged. Laser beams ionize the device, their radiations alter the content of the onboard memory, whose values are now observable.

Non Invasive. The hacker is limited to observe the behavior of the target device and is able to lead attacks based on the physical quantities the chip releases spontaneously into the environment or on the timing of operations. The owner of the targeted device cannot detect an unauthorized interaction, since the physical integrity of the chip is preserved.

3.2.3 Attack Modalities

The chosen attack modality depends on factors such as the availability of the target device for the attacker and their ability to collect a sufficient number of traces.

Profiled Attacks, also known as “two-stages attacks”, consist of two phases. In the first stage, a copy of the target device is used to gain information about the actual target. The attacker has full control of the copy of the device, can study its behavior with the different chunks of plaintext and build a *template* for the attack, which is a statistical generalization of the behavior of the profiling device for the different inputs. The acquired knowledge is then used during the second stage to attack the actual target.

This approach is considered as the most powerful, since an effective template only needs few traces for the attack stage to be successful [54]. On the other hand, it is based on strong assumption, since the hacker should own a profiling device as similar as possible to the attack device. For instance, products running banking applications often feature a

⁴In microelectronics, the passivation layer is a thin layer of insulating material that is applied to the surface of a semiconductor chip to protect the underlying active components and wiring from damage, corrosion, or other environmental factors.

closed implementation, which does not allow replicas easily.

Non-Profiled Attacks, also known as “direct attacks”. They take place in one stage only. The only assumption is that the attacker can record many traces from the device under attack. Powerful techniques such as *Differential Power Analysis* and *Correlation Power Analysis*, widely used both in the literature and in real life attacks, fall into the category.

3.2.4 Common Targets

Since Side-Channel Analysis constitutes attacks on the implementation of security algorithms, many different targets (often referred to as “victims”) have been studied in literature. The main categories they belong to are:

- **Embedded Systems.** Due their pervasiveness and simplicity of their structure with respect to more complex architectures, they were the first taken into account by this class of cryptanalysis. They represent the benchmark for any new approach studied in the research field;
- **General Purpose Computers.** In recent years, successful attacks against mobile phones or personal computers were performed. They present a high level of complexity, due to the underlying hardware and software. In particular, the plethora of hardware modules (e.g., multiple cores, each with numerous threads, hazard detection mechanisms, out-of-order execution blocks, and so forth) that compose modern systems make it hard to understand what portions of a leaked signal represent actual useful information, unless more intrusive attacks are performed. Difficulties also stem from the software, as the executed code often presents aggressive compiler optimizations, or due to the presence of an operating system, which could complicate the hacker’s analysis due to features such as context switching, software interrupts or exceptions, which suddenly interrupt the flow of execution [55] [56];
- **Cloud Infrastructures.** They represent one of the most relevant novelties in the investigation field. Researchers found out that virtualization, that is at the bases of cloud services, can expose further vulnerabilities. The physical co-residency of virtual machines can facilitate hackers’ interference with other virtual machines running on the same physical system, thus spreading the attack when an insufficient logical isolation is provided [57].

3.3 Power Analysis and related Techniques

Great relevance is given to techniques that target power consumption as a source of leakage, as many advances were achieved over the years. Few instruments are needed: a digital oscilloscope (or a capture/sampling device), a resistor and a Personal Computer. The analysis consists in measuring the energy drained by the target device, as it depends on the operations performed and the data handled.

Although its name refers to power, it is more precisely an analysis on current. Since oscilloscopes are only able to measure voltage, supposing that the device works with constant voltage (e.g., most devices work with constant operative voltages of 3.3 or 5V), a **shunt resistor** is placed.

Leveraging Ohm’s law⁵, the value of the current is obtained. The shunt resistor is

⁵Ohm’s law states that current and voltage in linear circuits are linearly dependent through a factor $R = V/I$, called *resistance*.

placed in series to the target device, according to Figure 3.8 or 3.9. Measuring the current across the device is not recommended as the microcontroller behaves as a variable resistor, thus complicating the analysis.

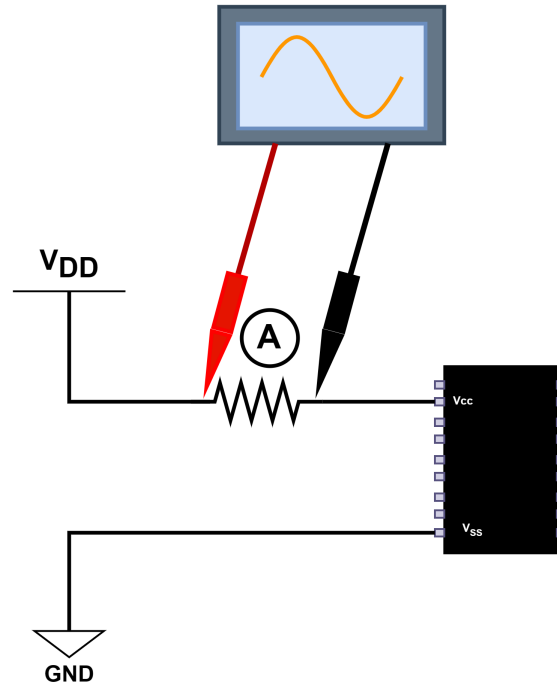


Figure 3.8: Shunt resistor located upstream of the target device.

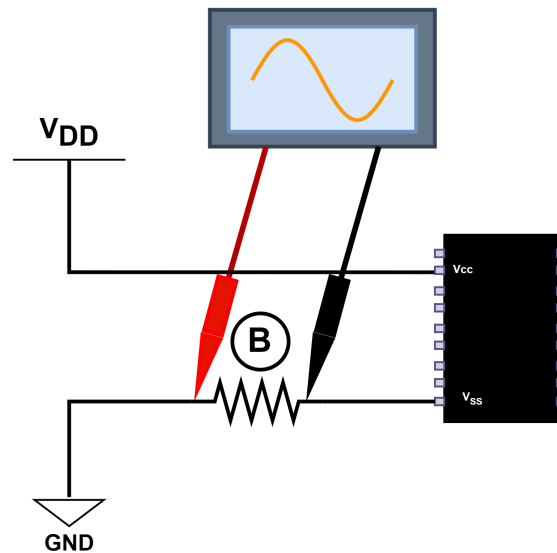


Figure 3.9: Shunt resistor located downstream of the target device.

The value of the resistor is usually around $40\text{-}100\Omega$. Higher values will amplify the amplitude of the power trace, thus facilitating visual inspection. On the other hand, they might drain a considerable quantity of current, stopping the target device (either upstream or downstream) from working.

3.3.1 Simple Power Analysis (SPA)

This technique takes its name from a paper by Kocher *et al.* [58], who first proposed it in 1998. In *Simple Power Analysis*, some power traces are recorded (even though one trace might be sufficient) and visually inspected, looking for significant features, such as peaks in power consumption.

Devices as small as microcontrollers often provide multiple execution units in order to perform instructions with least power or delay (e.g., hardware multiplier for MUL instruction or ALU for ADD instruction), each of which has a characteristic profile in power consumption. SPA works on the simple assumption that different instructions, once executed, leave distinguishable traces into the environment and leak information about the algorithm or even the handled data.

It is often considered as a preliminary step with respect to more advanced techniques, as it provides hints about the executed algorithm as well as the quality of the signal the attacker will be able to analyze.

Using SPA to break the RSA algorithm

Perhaps the most impressive application of Simple Power Analysis regards its effectiveness in breaking the RSA algorithm.

RSA is an asymmetric encryption algorithm⁶. At its core there is the exponentiation algorithm, that can be implemented as in Figure 3.10, where the private key is represented by the variable `secret_data`.

```
1  unsigned int exponentiation_algorithm(unsigned int secret_data, unsigned int m, unsigned int n) {
2      unsigned int P = 1;
3      unsigned int s = m;
4      unsigned int i;
5
6      for(i = 0; i < 10; i++) {
7          if (i > 0)
8              s = (s * s) % n;
9
10         if (secret_data & 0x01)
11             P = (P * s) % n;
12
13         secret_data = secret_data >> 1;
14     }
15     return P;
16 }
```

Figure 3.10: Implementation of the exponentiation algorithm in RSA [12].

The multiplications at line 8 and 11 give the name to the exponentiation algorithm, also known as *square and multiply*.

Taking a look to the listing, it is easy to notice that if the i -th bit of the key is set to '1', the function will execute the `if` statement at line 10, thus revealing information about the nature of the key itself. Applying the same strategy, also bits set to '0' are recognized since the same `if` statement will not be executed. In other words, when the key bit is 1, both square and multiply (SM) will be performed, when '0' only the square operation (S)⁷.

⁶In cryptography, an asymmetric algorithm is a type of encryption system in which two different but mathematically related keys (i.e., *public* and *private* keys) are used for encryption and decryption.

⁷The first iteration of the process is slightly different as the S operation is skipped. However, the considerations hold for the rest of the bits.

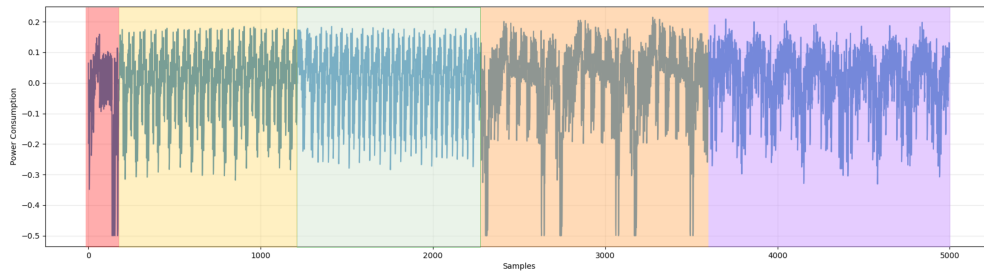


Figure 3.13: Simple Power Analysis, annotations on power trace.

The most distinguishable ones are the big fluctuations of power around sample 3000, the orange portion of trace in the Figure. The eye of an expert attacker will be able to recognize three nearly identical periodic operations. These constitute the three steps of the **ShiftRows** AES operation, during which each of the four rows in the State matrix is shifted by a quantity of positions equal to its row index in the matrix itself. They are only three as one row, the first one, must not shift at all.

Now that the first temporal reference in the algorithm is fixed, the attacker can recognize two groups of 16 periodic patterns each. The first one, in yellow, corresponds to the **AddRoundKey** operations. The second one, in green, is due to the **SubBytes** block. The last group, in purple, corresponds to the **MixColumns** step.

By exclusion, the attacker can also infer that the initial part of the trace, in red, is not strictly related to the algorithm but rather to preliminary procedures executed by the microcontroller before the actual encryption, thus can be discarded.

3.3.2 Differential Power Analysis (DPA)

Hints related to the instructions executed in a microcontroller gained through Simple Power Analysis can provide extremely useful information about the overall algorithm performed by the device, although can rarely provide enough clues to recover the encryption key.

Differential Power Analysis (DPA) was first formulated by Kocher *et al.* in their milestone paper [58]. The working principles of DPA might not be straightforward to follow, thus probably the best way to explain is through an attack example on AES.

As presented in Chapter 2, the AES-128 algorithm arranges the plaintext and key into a 4x4 matrix. The two blocks are then XORed together (**AddRoundKey**) and the resulting byte is eventually substituted into a **S-Box** (**SubBytes**). The operations generate the so called *intermediate value*, as illustrated in Figure 3.14. This quantity has no value for the algorithm itself, but great relevance for cryptanalysis.

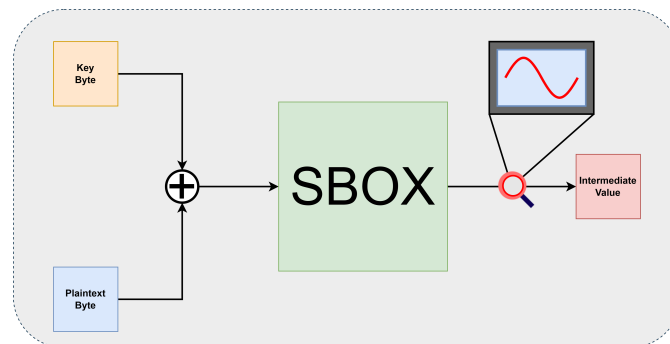


Figure 3.14: Particular of the AES algorithm: the AddRoundKey and SubBytes steps.

The underlying assumption for DPA to work is that the attacker manages to measure the power consumption at the output of the S-Box. Furthermore, the analysis is based on the hypothesis according to which a bit set to '0' or '1' contributes differently to the profile of the total power consumption of an electronic device, as previously stated.

The first phase of the attack consists in traces collection. For each trace, the attacker sends a byte of plaintext (whose value they can fully control) to the target device and records the power consumption of the system during the encryption.

The attack illustrated by the Python script in Figure 3.15 attempts to recover a single key byte, but the assumptions are valid also for the others.

```

6  all_differences = []
7  key_enumeration_list = [hex(i) for i in range(0,256)] # 0x00, 0x01, ..., 0xFF
8
9  for key_guess in key_enumeration_list:
10     # traces whose *GUESSED* LSB is equal to 0
11     traces_with_guessedLSBeq0 = []
12     # traces whose *GUESSED* LSB is equal to 1
13     traces_with_guessedLSBeq1 = []
14
15     for trace in traces:
16         # AddRoundKey between guessed key and plaintext (known)
17         # that generates the trace of the current iteration
18         hypothetical_input_sbox = key_guess ^ plaintext[trace]
19         # SubBytes
20         hypothetical_output_sbox = sbox[hypothetical_input_sbox]
21
22         # if else based on the LSB of the hypothetical output of the SBOX
23         if(hypothetical_output_sbox & 0x01):
24             traces_with_guessedLSBeq0.append(trace)
25         else:
26             traces_with_guessedLSBeq1.append(trace)
27
28     # computation of summary trace for group of LSB0
29     mean_tracesLSB0 = mean(traces_with_guessedLSBeq0)
30     # computation of summary trace for group of LSB1
31     mean_tracesLSB1 = mean(traces_with_guessedLSBeq1)
32     difference_of_means = mean_tracesLSB1 - mean_tracesLSB0
33     all_differences.append(difference_of_means)

```

Figure 3.15: Python script of DPA attack on a single bit of the key.

Initially, all possible values the key byte could assume are enumerated, from 0x00 to 0xFF (line 7). The attack is composed of two nested loops. The outer for cycle (line 9) iterates over the different key byte values described above, whilst the inner one (line 15) goes over the traces previously collected.

For each trace, its associated plaintext is XORed to the current *guess* of key byte, composing the *hypothetical* input of the S-Box (line 18). The hypothetical intermediate value (i.e., the output of the S-Box) is then computed (line 20). It is worth to notice that the hypothetical output of the S-Box is equal to the value the microcontroller generated only if the guess of the key byte is correct.

The attack continues by looking at the least significant bit (LSB) of the hypothetical intermediate value. The trace associated to the plaintext taken into account is appended either to the list where LSBs are *thought to be '0'* or to the one where they are *thought*

to be '1' (line 24 and 26, respectively).

The two groups are only populated with traces whose LSB is respectively '0' or '1' if the guess was correct (i.e., if the guessed key byte corresponds to the actual key byte). If not, then the input of the S-Box is different from the actual byte handled by the microcontroller, and consequently traces are not separated into the two groups by their LSB but rather randomly.

A mean for each of the two groups is then computed (lines 29 and 31), and their subtraction is then performed (line 32). This is a crucial step in DPA, as pointed out by the name itself (*Differential Power Analysis*).

If the grouping was done with the wrong key (Figure 3.16), then the difference will not highlight any effect of the LSB on the power trace (since each group is composed of random traces, not according to the *actual* LSB), and the elements in the difference vector will fluctuate around the zero, with some noise.

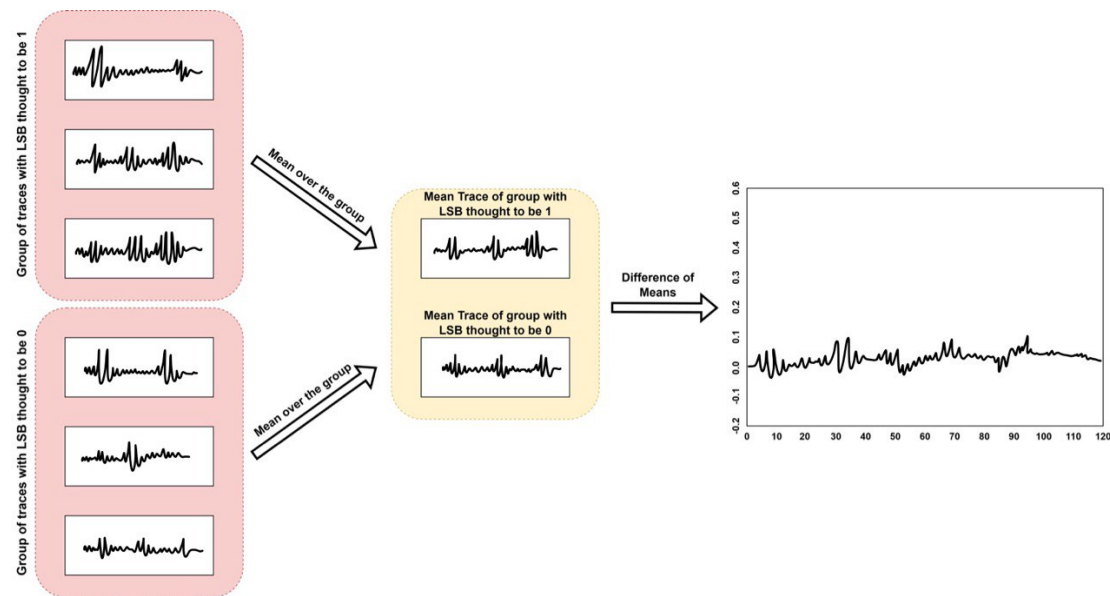


Figure 3.16: Wrong grouping of traces. The mean does not reveal the contribution of the LSB.

On the other hand, if the grouping was done with the right key (Figure 3.17), the traces are actually split according to the LSB.

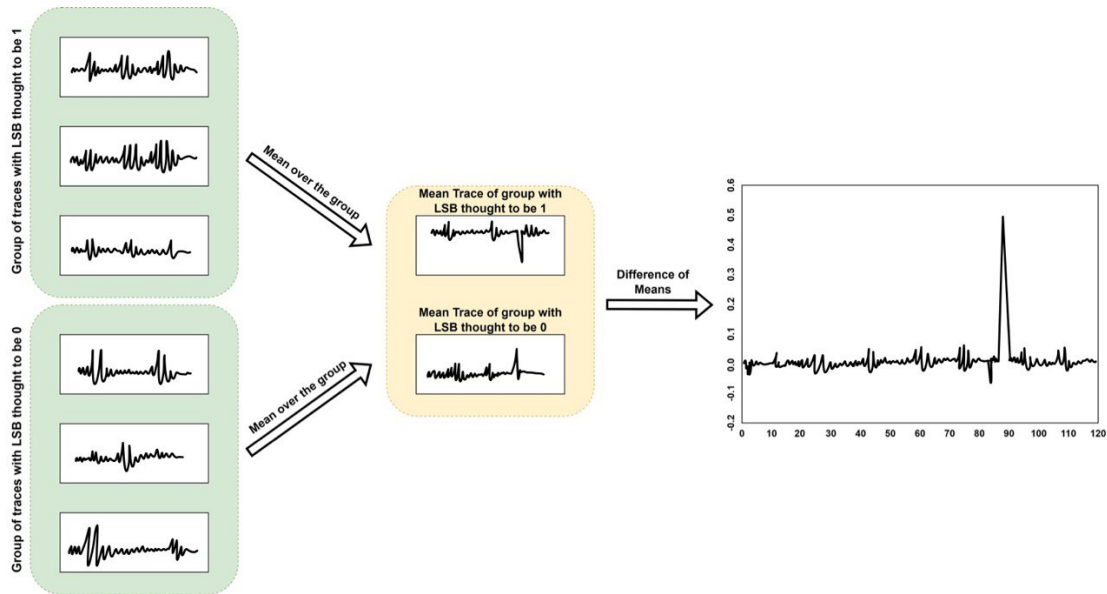


Figure 3.17: Correct grouping of traces. The mean reveals the contribution of the LSB.

The effect of this bit will be visible, as bits set to '0' or '1' consume a slightly different amount of power, which is now highlighted by the subtraction. As a consequence, a spike originates around sample 90.

The example of attack on AES demonstrates the strength of Differential Power Analysis, a technique that analyzes the value of one bit (mono-bit DPA, like the one just described) or a set of several bits (multi-bit DPA, not seen here) to recover the secret key of the encryption [59].

Simple instruments for measuring power consumption allow DPA to attack in few minutes algorithms that would take an enormous amount of time for other cryptanalysis techniques, such as brute-force. On the other hand, many traces are needed to highlight the effect of zeros and ones on the power consumption. However, the higher the number of traces collected the higher the chances of retrieving the secret key.

3.3.3 Correlation Power Analysis (CPA)

This Section introduces power models and Pearson's coefficient as propaedeutic concepts for the most advanced technique in Non-Profiled Side-Channel Analysis, *Correlation Power Analysis* (CPA), explained hereafter.

Power Models

Since attackers can rarely infer secret data from the plain observation of the leakage source, they often rely on **power models**: representations adopted by the hacker to model the leakage source. Among them, in "classic" Power Side-Channel Analysis particular relevance is given to the *Hamming Weight* (HW) and the *Hamming Distance* (HD).

The Hamming Weight (HW) The different components inside an electronic device (e.g., the ALU, a hardware multiplier module, and so forth) are often connected through *bus lines*, parallel interconnections that guarantee higher throughput with respect to serial ones. The *data bus* is a set of lines, each intended to carry a bit of data throughout the

system and serve the various modules. The basic working principles that allow to transfer bits are depicted in Figure 3.18.

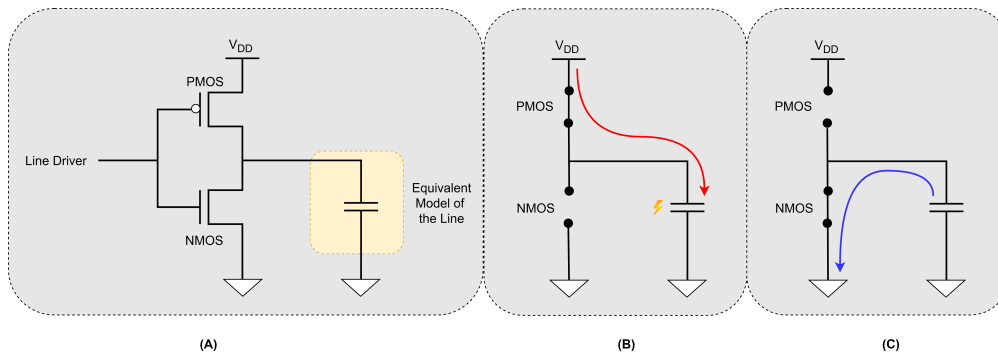


Figure 3.18: Working principles of a single data bus line.

Supposing for simplicity that the line is connected both to a power supply source (V_{DD}) and to the ground terminal, the equivalent model of the line is in Figure 3.18 (A). The data bus behaves similarly to a capacitor, by applying a charge it is possible to bring it to a high voltage, as in Figure 3.18 (B). When doing so, a quantity of current is drawn from V_{DD} (the red arrow in the Figure).

Additionally, the line can be driven to a low voltage level by removing the accumulated charge, as in Figure 3.18 (C). This process originates a current at the ground terminal with reverse direction with respect to the one previously described (the blue arrow in the Figure).

An example of the temporal evolution of successive data transfers on a 2-bit data bus is depicted in Figure 3.19.

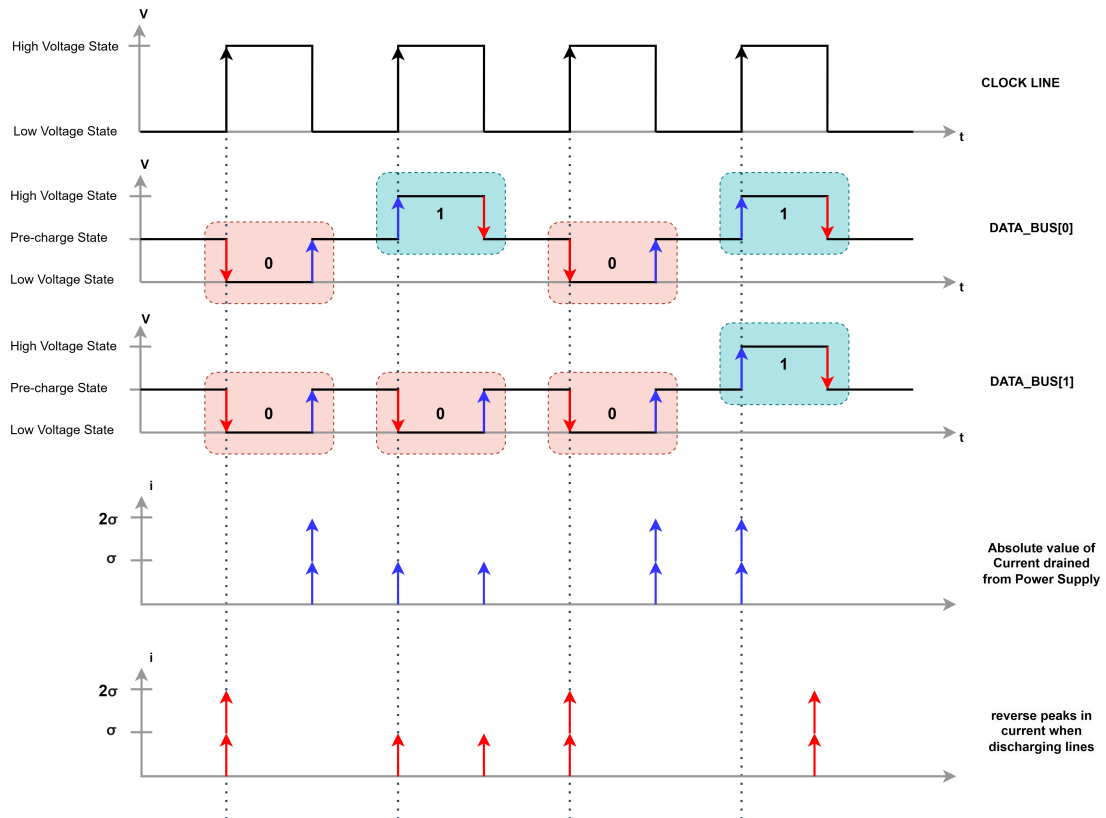


Figure 3.19: Current peaks due to bit transfers on bus lines.

The interconnections are preventively brought to a pre-charge state⁸. Since a transition of a single bus line from low to high means moving some charge towards the line itself, a current peak originates at V_{DD} (each blue arrow in the Figure), in accordance with the phenomenon depicted in Figure 3.18 (B). Multiple low to high transitions in the bus cause multiple peaks.

The attacker can measure the current at the V_{DD} terminal and infer the *number of bits set to '1'* at a given moment in the bus. This quantity is known as the *Hamming Weight* of the data, named after the American mathematician Richard Hamming. It constitutes an effective and straightforward relationship between the number of bits set to '1' in a bus and the observed power consumption.

Depending on the device, the width of the bus might vary, with 8-bit and 16-bit being very frequent. If the binary quantity 01000110 were carried on a 8-bit bus, it means that its Hamming Weight would be equal to $HW_{01000110} = 3$.

16-bit architectures will feature 17 possible classes of HW. On the other hand, 8-bit architectures will provide at most 9 different classes. The distribution of cases across the Hamming Weight classes is exponential in the number of bits, as depicted in Figure 3.20, where a 4-bit and 8-bit buses are compared.

⁸In digital electronics, a pre-charge state refers to the initial condition of the line before it is driven by a signal. It corresponds to a halfway value between the high and low voltage levels and it is generally employed to limit the inrush current during the power up procedure of the system.

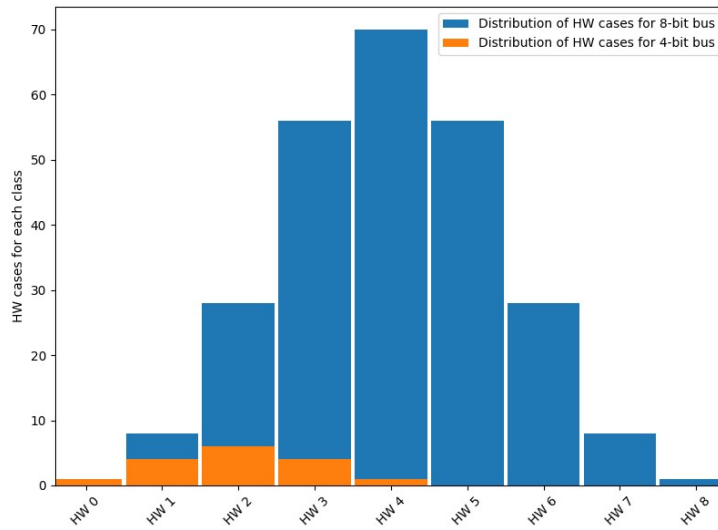


Figure 3.20: Values distribution of Hamming Weight for 4-bit and 8-bit data buses.

The Hamming Distance (HD) The Hamming Weight constitutes a valuable model when the power consumption of bus lines is observable. However, they might be hard to access, forcing the attacker to observe leakage somewhere else.

Electronic digital devices such as microprocessors and microcontrollers always feature *registers*, temporary storage locations that hold data and addresses. Their contents represent a further intrusion point for the hacker, as they might store parts of the key or plaintext handled during the cryptographic operations.

The *Hamming Distance (HD)* model is based on the observation that when the content of a register changes, the associated power consumption depends solely on the number of bits that flipped at a certain instant in time (i.e., from '0' to '1' or from '1' to '0'), as in Figure 3.21.

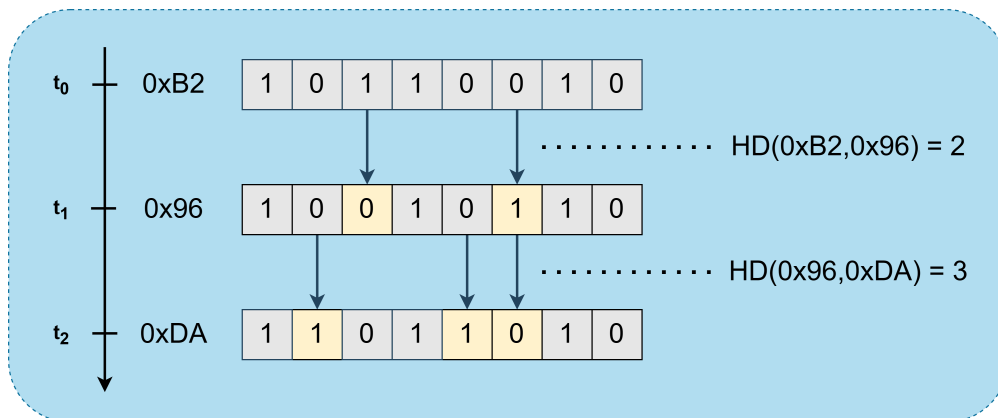


Figure 3.21: Hamming Distance of a register at successive time instants.

Given two strings a and b of the same length n whose characters all belong to an alphabet Σ , the Hamming Distance is formally defined as follows:

$$HD(a, b) = |\{i \in [1, n] : a_i \neq b_i\}| \quad (3.1)$$

In digital systems, each string from the definition above represents the possible content

of a register. The characters are represented by bits, that belong to a binary alphabet and can take the values '0' or '1'.

Furthermore, the relationship between Hamming weight and Hamming distance can be expressed mathematically as in the following equation:

$$HD(a, b) = HW(a \oplus b) \quad (3.2)$$

In real world scenarios, both leakage models can lead to successful attacks. The bus lines often provide points of observation for exploitable quantities. On the other hand, fundamental registers such as those storing flags⁹ often leak precious information about the status of a system and can be very appealing for an attacker.

As pointed out by O'Flynn and van Woudenberg in their book [12], there is no golden rule for deciding when to apply one of the two models rather than the other. An attacker who does not manage to model leakage in a profitable way through the Hamming Weight model should try to resort to the Hamming Distance.

Pearson's Correlation Coefficient

The Hamming weight model works on the assumption that the power consumption of a device and the number of bits set to '1' in its data bus are dependent: the higher number of the latter causes a growth of the former.

In statistics, *correlation* expresses the dependence between two random variables¹⁰. The underlying mathematics at the bases of this quantity is heavy and out of the scope of this thesis. However, the visual representation in Figure 3.22 helps understand its meaning.

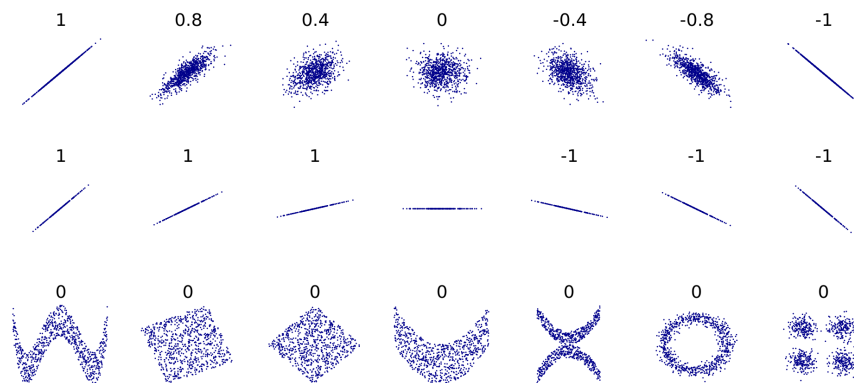


Figure 3.22: Visual representation of correlation between two random variables.

The Figure depicts the correlation between two random variables X and Y, which represents a measure of linear dependence between the two. Three main cases are possible:

- **Positive correlation** (values close to 1). It shows that the two variables move in the same direction (e.g., if one increases, the other does as well). It is the case for height and weight, as tall people generally tend to weight more;

⁹In processors, flags are bits that specify the state of the system (e.g., the Carry Flag indicates if a carry or borrow has been generated by the ALU).

¹⁰A random variable (also known as stochastic variable) is a mathematical formalization of a quantity that depends on random events.

- **Negative correlation** (values close to -1). An increase of a variable implies a decrease of the other. For instance, it is the case for speed and duration of a car journey, as a faster pace implies a shorter trip;
- **Zero correlation** (values close to 0, both positive and negative). No relationship exists between the two variables. An example of zero correlation would be the amount of tea drunk and level of intelligence.

A practical parameter to measure linear correlation is the *Pearson Correlation Coefficient* (Often abbreviated as *PCC* or ρ), whose scatter diagrams are in Figure 3.23.

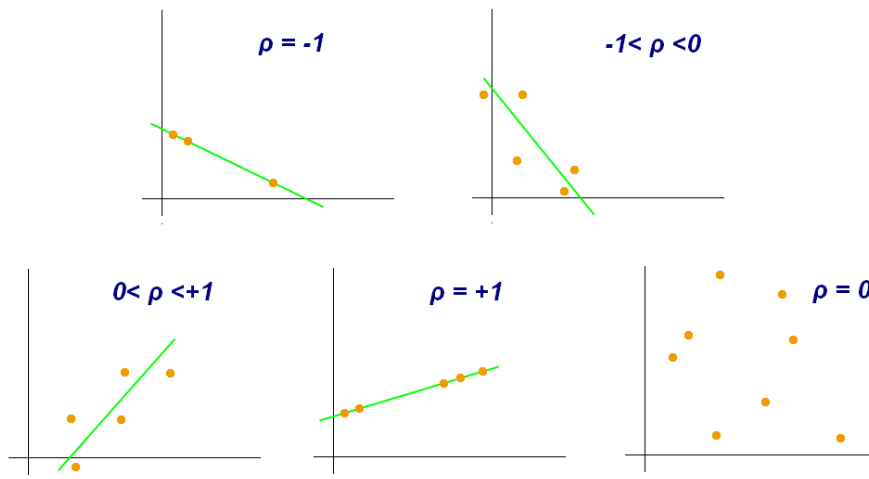


Figure 3.23: Scatter diagram of Pearson’s Correlation Coefficient [13].

Pearson’s coefficient varies in the range $[-1,1]$. Given two random variables X and Y , it is formally expressed as:

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} \quad (3.3)$$

where $\text{cov}(X,Y)$ is the covariance of X and Y , σ_X and σ_Y are the standard deviations of X and Y , respectively.

Direct CPA

Correlation Power Analysis is a powerful technique, first formulated by Brier *et al.* in 2004 [60]. It can be considered as a natural evolution of Differential Power Analysis, as the underlying assumption is similar, concerning the effect that bits set to ‘0’ or ‘1’ have on power consumption.

However, the difference between the two techniques is visible when attacking AES. DPA only exploits *partial* information, one (or at most some) bits of the intermediate value, assuming that the power consumption will vary accordingly. CPA, on the other hand, attempts to predict the *entire* intermediate value, taking advantage of the whole information available. For doing so it leverages statistics, and more precisely correlation.

At first, the hacker sends plaintext to the target device and the power consumption traces are collected, just like in DPA.

For each available trace and for each collected sample the attacker computes, by means of the Hamming Weight model, the *estimated* power consumption due to the hypothetical

intermediate value (i.e., the output of the S-Box due to the combination of plaintext, fully controlled, and the key candidate, the attempted guess).

At this stage, CPA leverages Pearson’s Correlation Coefficient as it is able to outline a linear correlation between the Hamming Weight of the hypothetical intermediate value and the power consumption of the actual intermediate value.

It is important to notice that the selected trace does not necessarily have to feature the highest positive correlation value, but rather the highest *absolute value* of PCC (e.g., between $\rho_1 = 0.45$ and $\rho_2 = -0.57$, the second key candidate will be selected as main guess). Negative values are not discarded as CPA does not look for positive correlation, but rather makes sure the dependence exists.

Interestingly, not only Pearson’s Coefficient can identify the best key byte guess, it also allows to build a ranking of all the other candidates, i.e. the key bytes that generate traces with lower PCC with respect to the main guess. In case the first guess is not the actual key byte, likely it is one of the immediate followers in the ranking, as they show descending correlation values.

The process for recovering a key on AES-128 with CPA seems time-consuming, as for each key byte the whole space of 256 candidates is explored, leading to an overall complexity in the order of:

$$T_{CPA} = N_{key}^{bytes} \cdot N_{byte}^{candidates} t = 16 \cdot 256t = 2^4 \cdot 2^8 t = 2^{12} t \quad (3.4)$$

where t is the time needed for a single attempt on one byte candidate.

Although it might seem a high number, it is worth to notice that while the time needed by brute force varies *exponentially* with the length of the key, equation 3.4 demonstrates that the duration of CPA varies *linearly* with respect to the number of key bytes.

In comparison to DPA, studies demonstrate that Correlation Power Analysis requires less traces to break the key [61]. Additionally, Differential Power Analysis requires an accurate grouping of traces to correctly identify the contribution of a single bit to the power consumption.

On the other hand, CPA leverages a powerful mathematical tool, the Pearson Coefficient, which makes the technique computationally expensive.

Reverse CPA

In Side-Channel Analysis the attacker often has to deal with devices whose internal workings are unknown.

Leakage assessment is a crucial preliminary step for most attacks, as the hacker generally does not know *when* the exploitable information will be released into the environment by the device.

Correlation represents a fundamental tool also for this purpose, as it helps finding a relationship between the collected traces and the plaintext or key that originates them, thus highlighting the period of time during which they leak secret data.

The plot in Figure 3.24 shows one of the traces collected from a XMEGA device during an attack on AES-128.

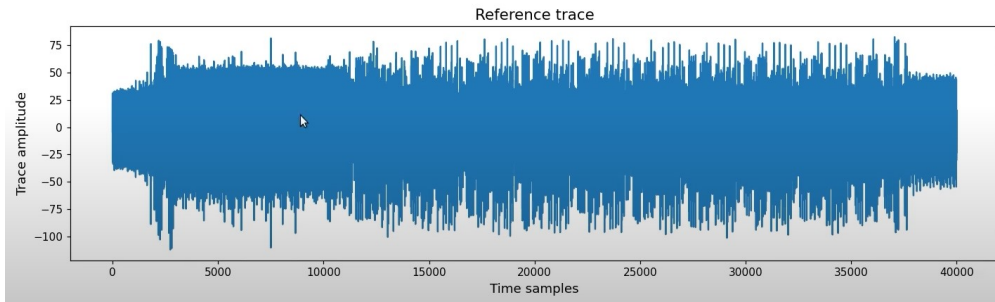


Figure 3.24: Example of trace collected during attack on AES-128 [14].

At the cost of a propaedeutic visual inspection, the attacker can save computational resources since a shorter portion of traces has to be examined during each attempt, as depicted by Figure 3.25.

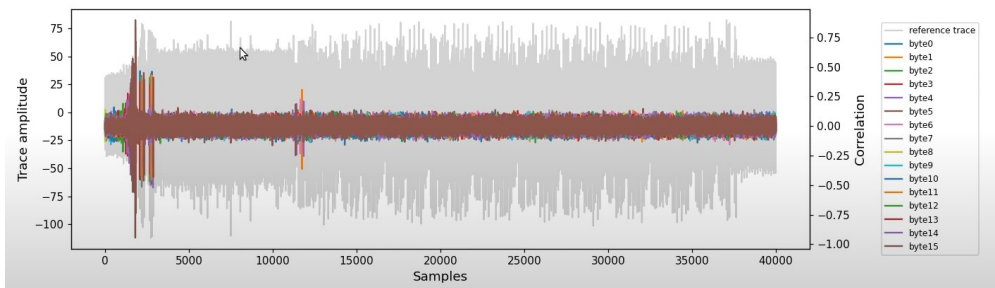


Figure 3.25: Reverse CPA with respect to plaintext on collected AES-128 trace [14].

The plot demonstrates that the plaintext is mainly handled between samples 0 and 13000, since the portion shows peaks of correlation between the trace and the plaintext itself. This means that the attacker should focus on the first half of the trace (i.e., time instants corresponding to samples between 0 and 20000) rather than on the whole available collection of samples.

3.3.4 Attack Points in AES

The previous Sections highlighted that AES although being mathematically robust is not invulnerable, as it allows attacks on its implementation on physical devices. However, another issue that has not been properly discussed so far regards *where* to find its weak points. Researchers call them *Attack Points* (also known as “leakage-sensitive variables”).

The yellow and red dots in Figure 3.26 depict the attack points originated by the first round of AES as well as the beginning of the second one.

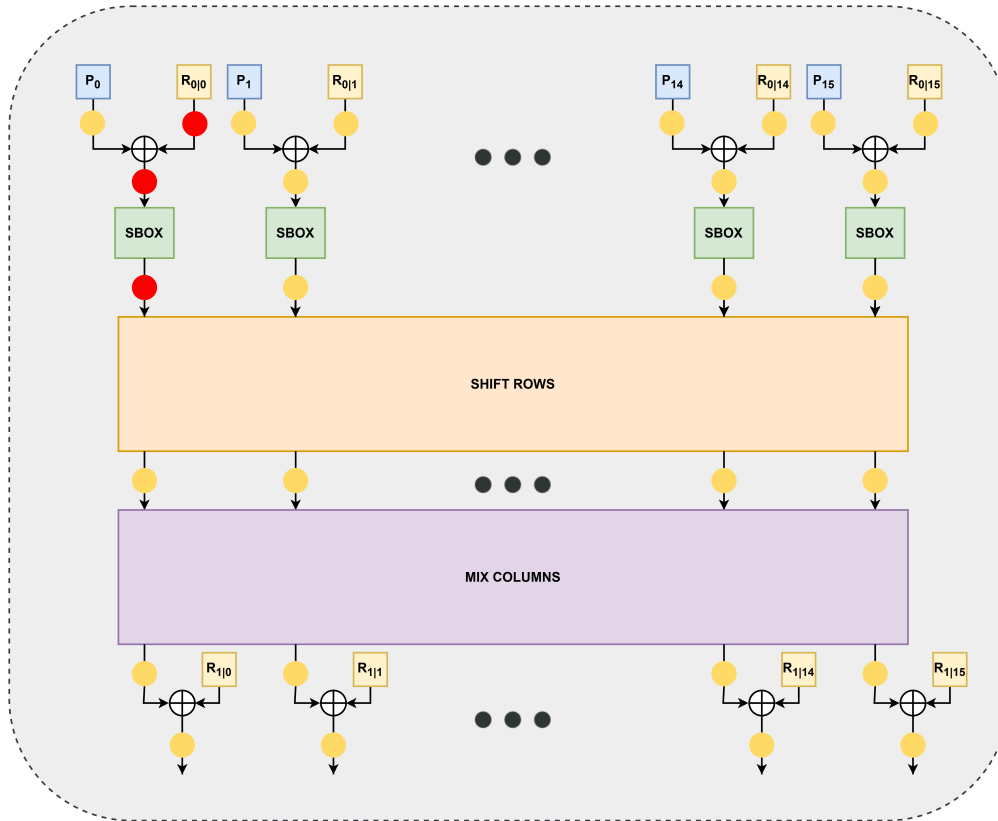


Figure 3.26: Attack Points in AES [15].

The points correspond to operations where computations forced a memory swap (e.g., a change of value in a register) causing a power consumption spike, that is observable. As AES-128 features ten rounds, the available attack points are many more.

Theoretically, you could attack any of them. However, only the first and last rounds are used in literature to recover the key directly, as the ones in the middle are more complex to exploit and need multiple reverse operations for the key recovery [62].

As pointed out by Elie Bursztein, Cybersecurity Director at Google, the most profitable attack points correspond to the red dots in the picture [15]. They are, respectively:

- **Round key retrieval**, as a feed of the key from the memory to the registers takes place. This is the simplest way to recover the key, as it represents a direct attack on it;
- **Output of AddRoundKey**, as the XOR operation might cause flips of bits in registers, whose peaks are visible in the total power consumption. This attack point allows to recover the key after an inverse XOR operation with respect to the plaintext, that is fully controlled by the hacker;
- **Output of the S-Box**, as a new value of the *State* has to be stored in memory. Here the recovery of the key needs a double passage, since both the plaintext has to be XORed out and the S-Box must be inverted.

Attackers' dream would be to always exploit the first kind of attack points. However, researchers proved that it features a low Signal to Noise Ratio (SNR) [62]. This physical quantity is a measure that compares the strength of a desired signal to the level of background noise, as in the following equation:

$$SNR = \frac{P_{signal}}{P_{noise}} \quad (3.5)$$

often measured in decibels (dB).

The low SNR makes the first attack point less appealing with respect to the other two listed above, as its contribution to the power consumption is too blurred and consequently the key recovery more complex.

3.4 Countermeasures

As presented in the previous Sections, many of the vulnerabilities exploited by Side-Channel Analysis stem from the basic working principles of digital circuits, apparently making devices doomed against this kind of attacks. However, big companies or even the diverse state defense departments dedicate considerable resources to preserve their crucial assets (e.g., data and devices), as they constitute strategical components for the various activities. For this reason, several countermeasures were developed over the years.

From a theoretical point of view, leveraging equation 3.5, it is straightforward to understand that the ratio should be decreased as much as possible by cyber defenders, either maximizing the noise power or minimizing the power of the exploitable signal for the attacker.

The main solutions for this purpose can be implemented at different levels.

Hardware Design

This series of countermeasures implies changes at the transistor level of digital circuits.

Many of them aim at maximizing the noise, as in the case of insertion of additional logic. This countermeasure uses complementary signals (e.g., \bar{A} with respect to A) in order to equalize the power consumption of different transitions. It can be applied by doubling the width of the data bus lines, having half of them transport useful data and the rest compensate the transitions of the first ones. This countermeasure is effective as an external observer cannot infer useful information, unless a more intrusive attack is performed with the consequent depackaging of the chip, allowing the hacker observe signals freely. On the other hand, the redundancy implies both a larger die area for the chip and higher power consumption [63].

Hardware countermeasures that decrease the power of exploitable information are also available, as researchers tried to employ Domino Logic rather than well-known CMOS, attempting to reduce the peaks in power consumption [63]. However, the complex design structure constitutes a major disadvantage.

Software Obfuscation

More flexible solutions are provided by software countermeasures, which mainly attempt to puzzle the relationship between the timing of operations and the handled secret data.

Jitter is a natural phenomenon in electronic devices, that deviates periodic events from their clock reference. Artificial jitter can be inserted at multiple stages of the cryptographic algorithm to randomly increase or decrease the duration of operations. It is usually created with instructions that do nothing (e.g., NOP in machine language), and complicates the analysis for the attacker [64].

Another commonly used technique consists in masking sensitive data, using random masks to split cryptographic variables into multiple shares harder to handle with respect to the original one [65].

Changes in the Algorithm and Good Practices

Additional countermeasures for AES were developed and imply changes to its basic blocks. In particular, values in the **S-Box** of the **SubBytes** operation can be edited in order to provide a lower power in the leakage signal, thus reducing the numerator in equation 3.5 [37].

In addition to the indicated countermeasures, researchers proposed guidelines that enhance protection of the systems without changes to the hardware or software. Among these, the introduction of a maximum number of encryptions with the same key, after which it should be considered expired and replaced by a new one.

Chapter 4

Deep Learning Side-Channel Analysis (DLSCA): State of the Art

4.1 Introduction to Deep Learning Side-Channel Analysis

Side-channel Analysis has been a major concern in the security of embedded systems, especially in smart cards and Internet of Things (IoT) devices [66] [9].

Classic SCA, presented in the previous Chapter, relies on statistical analysis of physical characteristics, such as power consumption, heat or electromagnetic radiation, to extract secret information from a cryptographic device. It is effective in many cases, but demonstrates limited capabilities in capturing complex patterns, which can lead to difficulties in recovering encryption keys.

Inferring what operations the target device is performing can be very hard, and expertise in Simple Power Analysis is needed, as the visual inspection might be complex due to the presence of countermeasures or to the noisy nature of the exploitable signal.

A further difficulty for the attacker stems from the adaptability of traditional Side-Channel Analysis techniques, as any device might operate the same cryptographic algorithm with variations with respect to others (e.g., device A might leak a considerable amount of information during the `SubBytes` operation whereas device B might not leak any during that same `AES` operation).

Furthermore, statistical power models built by the attacker can only capture linear dependence between the consumed power and the data, as pointed out in Section 3.3.3.

Artificial Intelligence (AI) has become an ubiquitous technology in modern society, permeating almost every aspect of life, from agriculture to healthcare.

In recent years, Deep Learning (DL), the branch of AI that enables machines to imitate human behavior, has emerged as a powerful tool for a wide range of applications in various fields, including speech recognition, natural language processing and computer vision.

As the elevated number of studies suggests, Deep Learning also constitutes a promising approach to Side-Channel Analysis, since it promises to address the problems traditional SCA is affected by [22].

The first attempts to leverage Artificial Intelligence as an enhancement to classic Side-Channel Analysis techniques date back to the late 2000s, when researchers successfully deployed basic Machine Learning (ML) algorithms, of which Deep Learning techniques represent a subset, for the pre-processing of raw power traces [22]. Such operations were performed in order to highlight the portions of traces that could leak significant amounts of exploitable information, decreasing the need for human intervention.

Although being both approaches of the broader branch of Artificial Intelligence, Machine Learning and Deep Learning are substantially different from each other.

Machine Learning is a technique through which a machine learns by examples from a large amount of data and uses that knowledge to make predictions, identifying patterns and relationships in the data.

Deep Learning, on the other hand, is a more complex type of Machine Learning that uses Neural Networks, structures inspired by the human brain, to learn from large amounts of data at the cost of a high computational power.

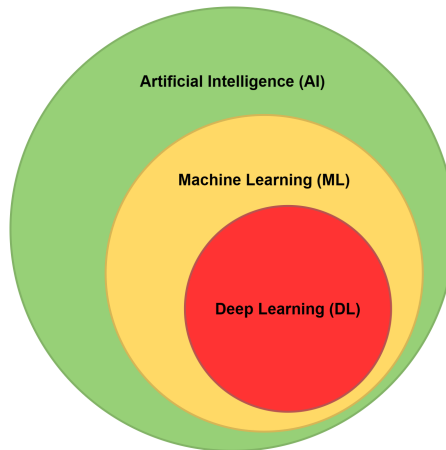


Figure 4.1: AI, ML and DL.

It is hard to outline the end of the Machine Learning era in SCA and the beginning of the Deep Learning one, as the distinction between the two is often blurred. As a temporal reference, researchers date back the usage of the first Deep Learning in Side-Channel Analysis to the past mid-decade.

Nowadays, the number of papers regarding *Deep Learning Side-Channel Analysis* (DLSCA, in short) suggest that Deep Learning outclassed Machine Learning [22]. This overtake is essentially due to two factors that facilitate the attacker during their campaign against the target device, listed below.

Lack of pre-processing for power traces. Deep Learning algorithms autonomously decide what parts of the input data (e.g., the raw power trace collected from the target device) are relevant. Machine Learning algorithms, on the other hand, would require a preliminary stage, called *Features Extraction Phase*, during which the attacker *manually* selects characteristics of interest out of the raw data. In ML, the choice of features impacts highly on the algorithm's performance, as the hacker can explicitly indicate what portions to neglect and which ones to spend computational resources on. Feature extraction is an operation that is heavily based on experience, as specifying features demands knowledge deriving from different attack scenarios.

Performance benefits from Deep Learning algorithms, as numerous studies demonstrate that results outclass methods based on Machine Learning [22]. This is mainly due to algorithms that are more powerful with respect to the ones adopted in ML.

4.2 Basic Principles of Deep Learning

This Section introduces the basic principles of Artificial Intelligence, providing background notions for the correct understanding of Deep Learning Side-Channel Analysis.

4.2.1 Artificial Neurons

Deep Learning attempts to simulate the behavior of the human brain, teaching machines to *think* as humans do. For doing so it mimics the brain down to its basic units, the *neurons*, depicted in Figure 4.2.

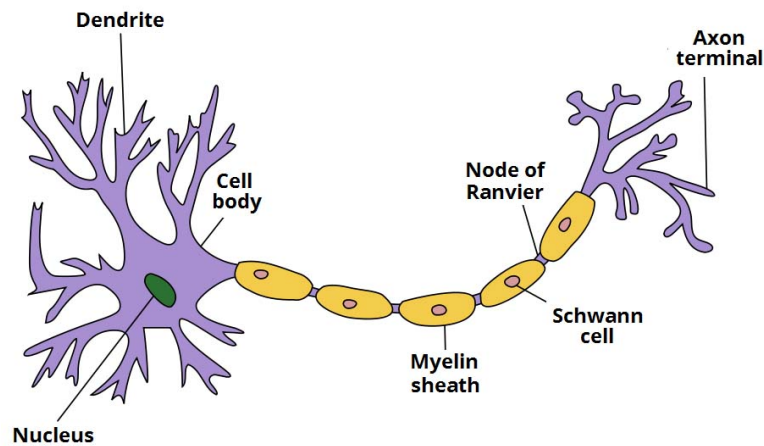


Figure 4.2: Neuron in the human brain [16].

The typical human neuron consists of a cell body, dendrites, and an axon. Dendrites receive signals from other neurons, the nucleus inside the cell body elaborates a reaction to the input stimuli and a consequent electric impulse originates and propagates to the other neurons through the axon.

Similarly, Artificial Intelligence employs its own neurons, called *perceptrons*. Just like their biological counterpart, they accept inputs from the other neurons, can generate a reaction to the stimuli and propagate it to the other perceptrons in the structure. Although the working principles of the human brain are not all figured out yet, mathematical tools are leveraged to mimic their behavior.

The first model of perceptron was formulated by Rosenblatt in 1957 [67], and is illustrated in Figure 4.3.

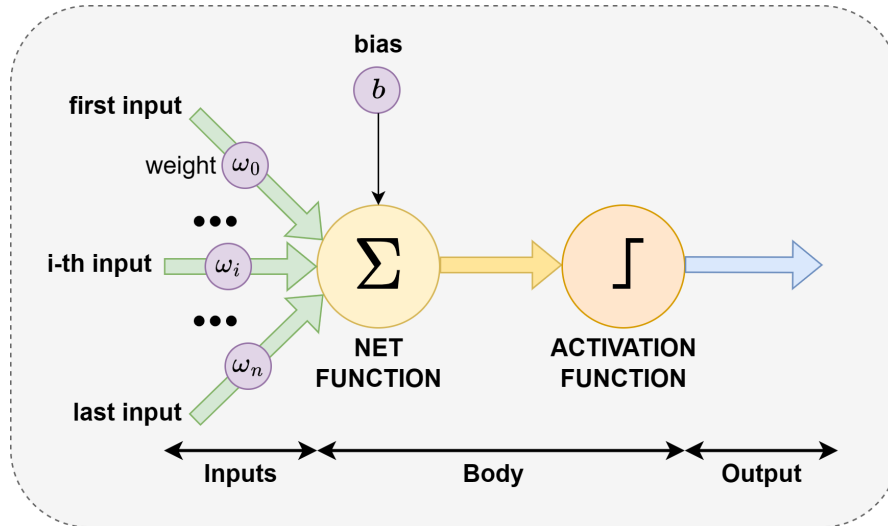


Figure 4.3: Artificial neuron.

The neuron receives the different input signals, each multiplied by a certain factor ω_i . These numerical values, known as *weights*, determine the strength of the connection between the input signal and the neuron. The body is composed of two main parts, the *network function* and the *activation function*.

Network Function

The *network function* computes a weighted sum of the input signals to which a special value, known as *bias*, is then added up. The neural bias allows to make a decision that is not solely determined by the inputs, but rather takes into account the contribution of the specific neuron.

Given n input signals X , each with their own weight ω_i and given a neural bias b , the net function can be formally defined as:

$$F(X, \omega, b) = \sum_{i=0}^n X_i \cdot \omega_i + b \quad (4.1)$$

Activation Function

The sum is then processed by the *activation function*, a non-linear operation. This function determines if, with the data currently processed, the neuron will be activated or not, with a consequent propagation of the carried information to the neurons downstream. Furthermore, it allows the perceptron to learn complex data patterns, not caught by the network function, which is linear.

Many activation functions are present in literature. Some of the most popular ones are:

- The **sigmoid** function, depicted in Figure 4.4.

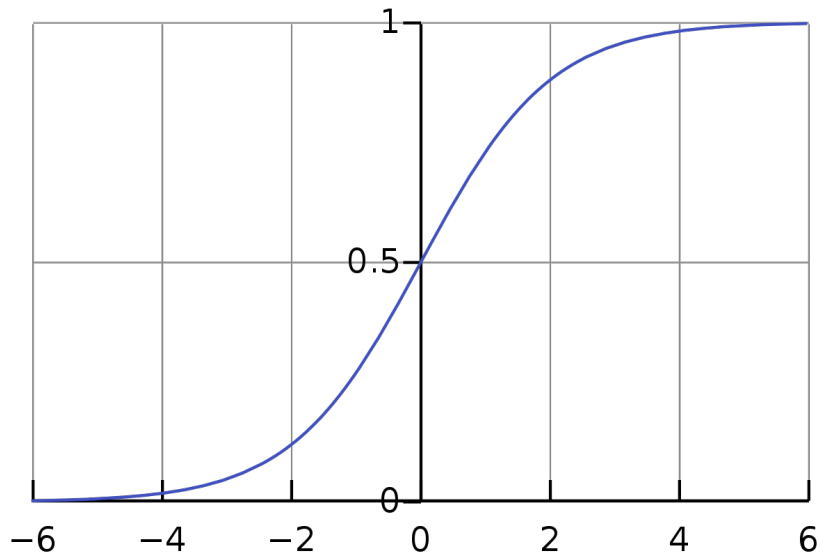


Figure 4.4: The sigmoid activation function [17].

Given a weighted sum x , it is formally defined as:

$$S(x) = \frac{1}{1 + e^{-x}} \quad (4.2)$$

The sigmoid activation function can be interpreted as the probability with which a weighted sum, normalized between 0 and 1, is classified as 1.

- The **ReLU** (*Rectified Linear Unit*) function, illustrated in Figure 4.5.

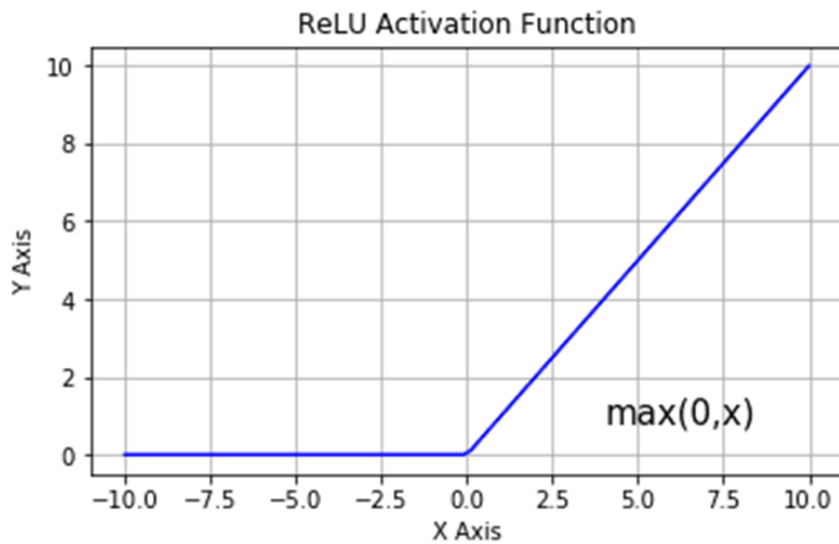


Figure 4.5: The ReLU activation function [18].

Given a weighted sum x , it is expressed as:

$$R(x) = \max(0, x) \quad (4.3)$$

- The **ELU** (*Exponential Linear Unit*) function, as in Figure 4.6.

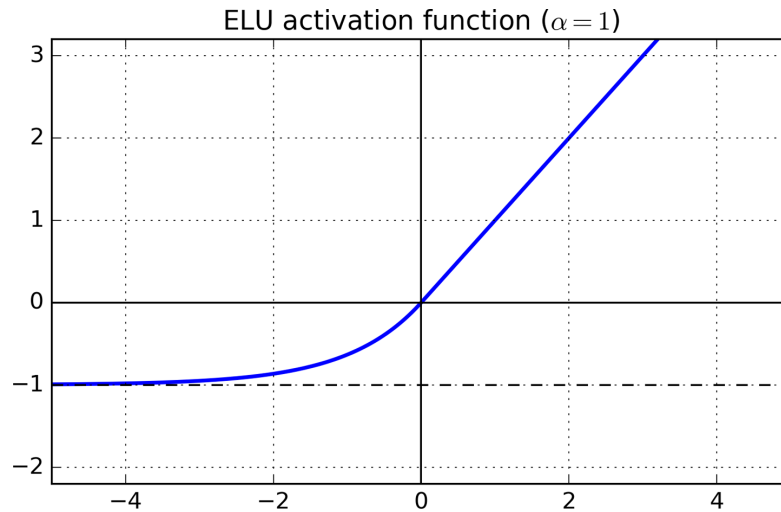


Figure 4.6: The ELU activation function [18].

Similar to the previous one, it is smoother than the ReLU function and allows negative values as output. Given a weighted sum x :

$$E(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha(e^x - 1) & \text{otherwise} \end{cases} \quad (4.4)$$

where α is a constant defined by the user. The main drawback of ELU with respect to ReLU stems from the usage of the exponentiation, as it is slower than computing the maximum value between two numbers.

- The **Softmax** function. It is a mathematical operation that converts a vector of numbers into a vector of probabilities (i.e., the probability density function). The likelihood of each element is normalized with respect to the values in the starting vector. The Softmax provides an output vector that can be mathematically expressed as:

$$D(z)_i = \frac{e^{z_i}}{\sum_{j=0}^N z_j} \quad (4.5)$$

where z is the starting vector of elements and z_i its i -th element.

4.2.2 Artificial Neural Networks

A single perceptron by itself can only solve problems that admit a linear function as a solution, due to the neuron's internal operations (network and activation functions) [19]. However, not all problems admit one. Figure 4.7(A) illustrates the truth table of the AND operator. Here, the true and false outputs can be split thanks to a linear function, thus a single perceptron could be used to separate combinations of the inputs that generate respectively '0' and '1'. Figure 4.7(B) depicts the truth table for the XOR operator. Here it is impossible to separate with a straight line the combinations that lead to the two different outputs.

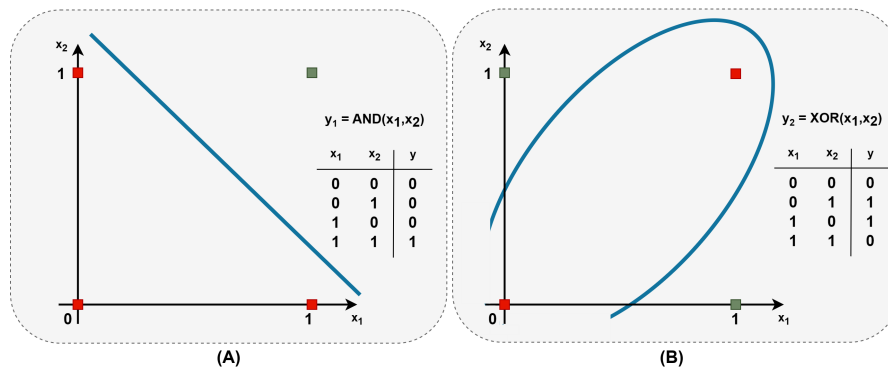


Figure 4.7: AND (A) and XOR (B) truth tables [19].

Although one single neuron fails, multiple perceptrons could solve this problem. Researchers in the Artificial Intelligence field have proposed various architectures that employ numerous perceptrons, known as *Artificial Neural Networks* (ANNs). The most frequent networks in the literature are presented here.

Multilayer Perceptron (MLP)

Perceptrons can be arranged into multiple layers, as in Figure 4.8. The architecture is known as *Multilayer Perceptron*.

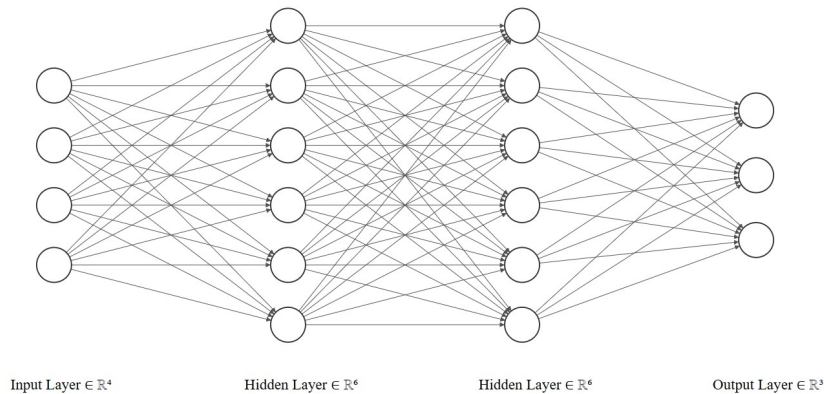


Figure 4.8: Example of the MLP architecture.

Each circle in the figure corresponds to a perceptron and includes both the network and activation functions.

Three different stages can be distinguished:

1. The **Input Layer**, the first group of neurons in vertical in the mesh. The number of perceptrons often coincides with the cardinality of the vector that is fed to the network itself (i.e., with the input data the network has to gain knowledge from and/or classify);
2. The **Hidden Layer(s)**. It performs most of the computation required by the network. If there is more than one, the network is considered a Deep Neural Network (DNN) [22];

3. The **Output Layer**, the last group of neurons in the mesh. The size of the layer depends on the task that the model is designed to perform (e.g., if the model is meant to classify drawings into “dog”, “bird” and “cat”, then there will be three neurons in this layer).

The structure is *fully connected* (i.e., each neuron from layer i is connected to each neuron from layer $i+1$). Additionally, the neural parameters ω and b are trainable, as their refining enables the model to learn patterns and relationships in the input data so that it can make accurate predictions or classifications on new, unseen data.

Currently, MLPs are used in a wide range of applications, such as:

- Classification of the input data into different categories, called *classes*;
- Regression (e.g., prediction of the price of a house depending on its features such as the presence of a garden or the number of rooms). In contrast to classification, the output is a continuous value rather than a class.

Convolutional Neural Network (CNN)

Convolutional Neural Networks (CNNs) are a powerful class of Deep Learning models inspired by the organization of the visual cortex in the human brain. They feature multiple layers, each composed of independent perceptrons. Figure 4.9 depicts a CNN that aims at classifying input images (e.g., a 'X' in the Figure) into three different classes: circle, X mark and triangle. The predicted class is highlighted in green.

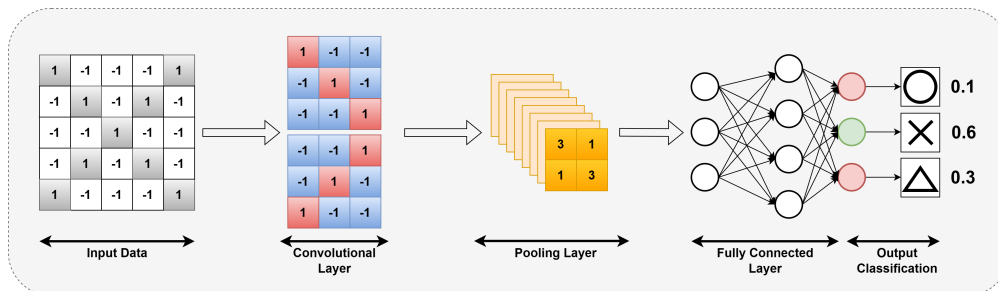


Figure 4.9: Example of CNN architecture.

The input data is arranged as a squared matrix of pixels. The brightness level of the pixel is associated to a numerical value (e.g. from -1 to +1 in the Figure). The CNN correctly classifies the input data, arranged as a matrix of pixels, thanks to different layers:

1. The **Convolutional Layer**. Here a series of filtering matrices (known as *kernels*) is employed. Each filter slides over the input image and, at each position, the dot product between the filter weights and the corresponding pixels of the input image are computed, generating the *Activation Map*. This process highlights possible similarities between the input image and the filter taken into account, as depicted in Figure 4.10.

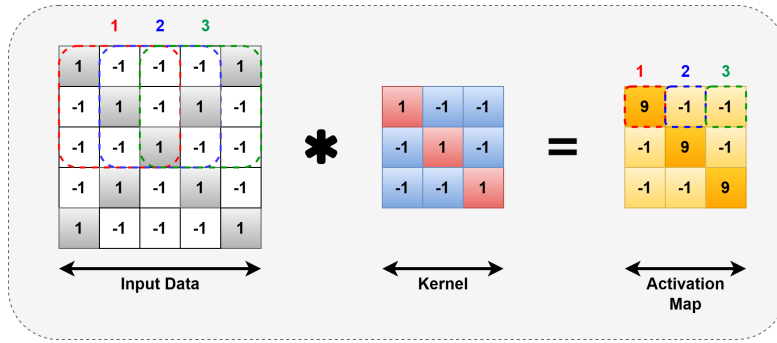


Figure 4.10: Convolution operation in CNNs.

2. The **ReLU layer**. It leverages the ReLU activation function defined in the previous Section to rectify the elements in the map, as in Figure 4.11.

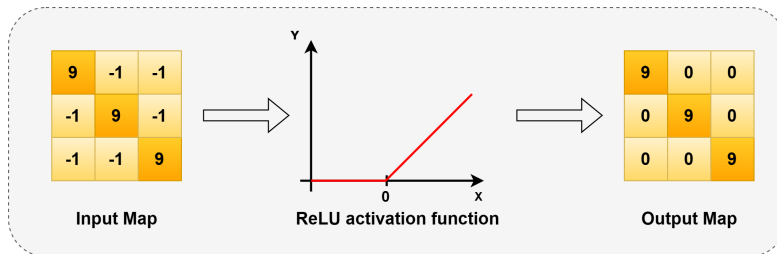


Figure 4.11: Rectification in CNNs.

3. The **Pooling Layer**. This layer computes a summary of the activation map by taking the maximum (*max pooling*) or average (*avg pooling*) value of partial regions in the map, as illustrated in Figure 4.12.

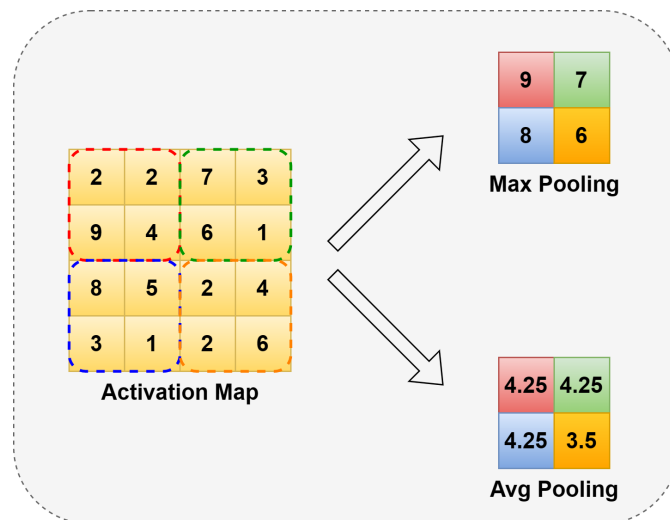


Figure 4.12: Max and Avg pooling in CNNs.

Average pooling smooths out the input data, which can be a drawback when sharp features must be identified. On the other hand, max pooling shows its limits in particular scenarios when the majority of the input image consists in light-colored pixels [68];

4. The **Fully Connected Layer**. It takes the high-level features learned by the convolutional and pooling layers and uses them to classify the input image into one of the possible output classes. Perceptrons belonging to the last layer commonly feature the Softmax activation function, as it intrinsically provides a probability density function among the different output classes.

Multiple rounds of subsequent convolution and pooling layers can be present in a CNN. At each round the kernel shows higher abstraction (e.g., a network that classifies houses will have a first round of kernels that aim at spotting vertical and horizontal edges, while a second round could identify windows or the roof).

Convolutional Neural Networks demonstrate a notable aptitude for detecting and extracting features from images. Some common applications include:

- Classification of input images into different categories (e.g., classification of vehicles into “car” or “motorbike”);
- Object detection, as CNNs can detect and locate objects within an image. This is useful in applications such as self-driving cars and robotics.

Autoencoder Network

Autoencoder Networks are models that can be conceptually divided into three parts, as illustrated in Figure 4.13.

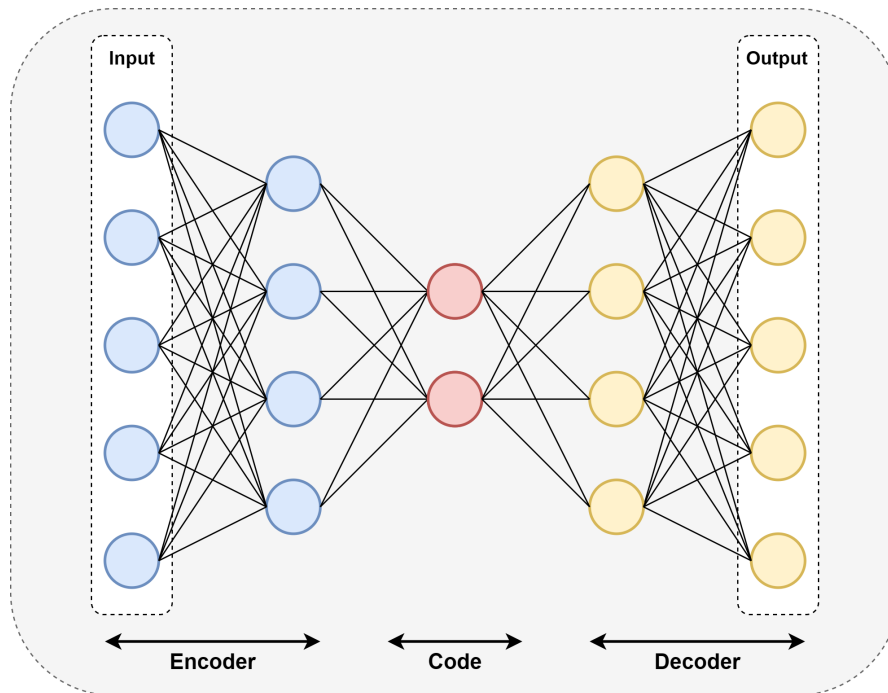


Figure 4.13: Example of Autoencoder network.

The leftmost part, in blue, is called *encoder*. It consists of a fully connected structure that compresses the input data into the *code*, in red. The *decoder* is the rightmost part in the figure, in yellow, and consists of another fully connected structure that tries to reconstruct the original input data out of the compressed information in the code.

The network might seem irrelevant as, when properly built, its output should be as close as possible to the input data. However, it has great relevance since it can detect

which features of the data are relevant, separating the useful signal from the noise. Figure 4.14 depicts the effect of an autoencoder network on a handwritten digit.

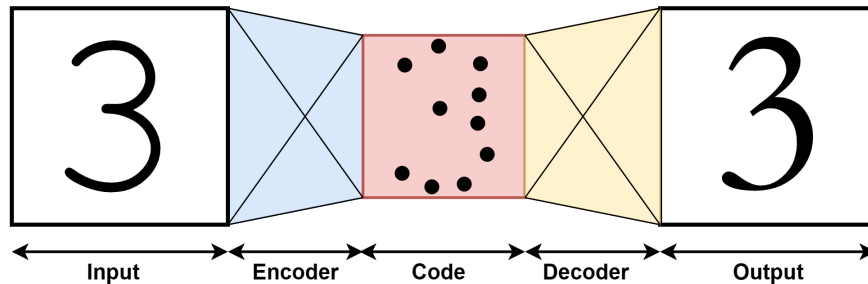


Figure 4.14: Effect of Autoencoder on handwritten digit.

The encoder extracts the meaningful features of the starting image. The partial result is the code portion of the network, that displays the peculiar characteristics of the digit (e.g., dots in the edges and vertices that make number 3 distinguishable from the other figures). The decoder reconstructs the input thanks to the compressed information, building a slightly different version of the handwritten digit that was fed to the network [69].

Autoencoders are commonly used for tasks such as:

- Image denoising. On the opposite to traditional denoising tools, rather than deleting noise from the picture they extrapolate features regarding the signal of interest;
- Generative Models. Once the meaningful features of some specific input signal are extracted, they can be used to build new data that is similar to the starting one, but totally new. Digital manipulations of audio, video, and images known as “Deepfakes” make large use of autoencoders [70].

Recurrent Neural Network (RNN)

MLPs, CNNs and Autoencoders all belong to the category of *feed-forward* Neural Networks, in which the data flows in one direction from the input to the output. Although these models are widely used for classification and regression, they cannot solve problems that require the handling of sequential data.

Let us consider a simple example, the incomplete sentence “*I live in Italy, so I can fluently speak ...*”. Any human, acquiring knowledge from the words in the sentence, can infer that the missing word is “*Italian*”. This guess, natural for people, is impossible for feed-forward Neural Networks as they would require a feature to extract and make predictions on (e.g., the particular length of the sentence to then predict the next word) [71]. A sort of memory mechanism of the input string would be needed to achieve the successful prediction of the following word. This mechanism is at the bases of *Recurrent Neural Networks* (RNNs), models that can operate on a sequence of data rather than on static input.

In RNNs the classic perceptron is employed, with information gathered from upstream neurons, summed up and then fed to the activation function. However, in contrast with feed-forward networks, the output of the perceptron is fed back to the input, creating a loop, as in Figure 4.15 on the left.

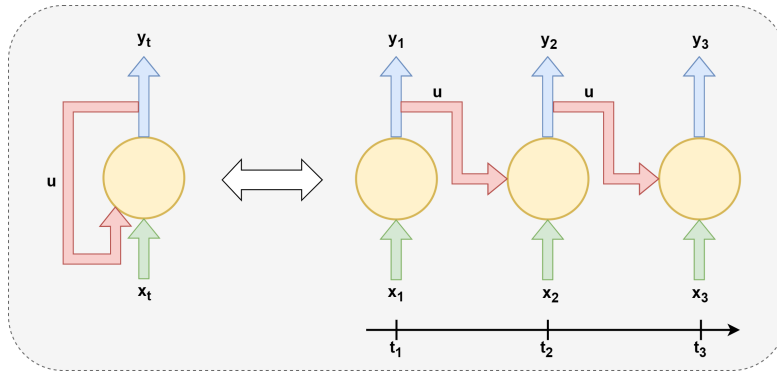


Figure 4.15: Effect of feedback in RNNs.

The time evolution is represented in the rightmost part of the picture. The output of the perceptron at a time t_1 is fed to the input of the very same neuron at time t_2 , and so forth. Thanks to this mechanism, the RNN can gain knowledge out of the string from the example and make a prediction based on the *whole* available information [72].

RNNs have applications in many fields, among which:

- Natural Language Processing (NLP), a field of computer science that focuses on the interaction between computers and human language (e.g., chatGPT by *OpenAI* [73]).
- Financial Forecasting, in order to predict future stock prices based on historical data.

4.2.3 The Learning Process

The goals of the different neural networks were extensively discussed in this Chapter, from data classification to image recognition. Nevertheless, the fundamental mechanism at the bases of their capability of *thinking* was not tackled yet. Machines equipped with artificial intelligence tools learn like humans do, by examples, thanks to the so called *learning process*.

According to the paradigm of **Supervised Learning**, a training set of data is fed to the network. It consists of *labelled* data, i.e. input data along with their output labels. For instance, for image classification in CNNs, the training set might contain thousands of pictures of animals, each with its own class provided (e.g., a photo of a dog along with metadata specifying it is a dog).

From a mathematical point of view, “learning” consists in optimizing the neural network’s parameters (i.e., the weights and biases) by minimizing a cost function defined by the user.

Common Cost Functions

Cost functions represent a method for evaluating how well the specific model fits the given data. If the prediction deviates too much from the expected result, then the cost function will catch the misbehavior.

Many cost functions exist in the literature, and their choice highly depends on the task that the model has to perform, e.g. classification or regression. As pointed out in the previous Section, the two tasks are different, as the former aims at predicting the output from set of finite categorical values (e.g., distinguish handwritten numbers into the ten figures from ‘0’ to ‘9’) while the second deals with predicting a continuous value out of

specific features presented by the input data.

As to regression, the most common cost functions are:

- **Mean Square Error**, also known as *L2 loss*. Given n training examples, each with their specified output class \hat{y}_i , it is mathematically formulated as:

$$MSE = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{n} \quad (4.6)$$

where y_i is the output computed by the neural network taking into account the i -th training example. Due to the nature of this function, mispredictions that are far from the actual values are heavily penalized in comparison to less deviated predictions.

- **Mean Absolute Error**, also known as *L1 loss*. Given n training examples, each with their specified output class \hat{y}_i , it is expressed as:

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n} \quad (4.7)$$

The absolute value in MAE works as the exponentiation function in MSE, since the magnitude of the misprediction is taken into account rather than its sign.

As to classification tasks, the most common cost functions in the literature are:

- **Hinge Loss**. This cost function returns a low value if, among all the possible output classes in a classification problem, the correct one has achieved the highest score. Its mathematical formulation is the following:

$$HingeLoss = \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1) \quad (4.8)$$

where s_{y_i} represents the score achieved by a specific class on the training data i and s_j the score of any other possible output class. A clarifying example of classification with Hinge as cost function is in Figure 4.16, where for each row in the table the score of the class is reported.




	Image #1	Image #2	Image #3
Dog	-0.39	-4.61	1.03
Cat	1.49	3.28	-2.37
Horse	4.21	1.46	-2.27

Figure 4.16: Scores in classification problem [20].

For each triplet, the Hinge Loss function is computed as follows:

$$\begin{aligned} \text{Image 1: } & \max(0, (1.49) - (-0.39) + 1) + \max(0, (4.21) - (-0.39) + 1) = 8.48 \\ \text{Image 2: } & \max(0, (-4.61) - (3.28) + 1) + \max(0, (1.46) - (3.28) + 1) = 0 \\ \text{Image 3: } & \max(0, (1.03) - (-2.27) + 1) + \max(0, (-2.37) - (-2.27) + 1) = 5.2 \end{aligned}$$

As aforementioned, the cost function returns a low value when the score of the correct class is the highest among all possible classes. This is not the case for images 1 and 3, as the model respectively individuates the horse and the dog as most likely animal (instead of dog and horse), thus the function returns a quantity larger than zero. On the other hand, the model is able to correctly recognize a cat in image 2, and consequently the function returns 0.

- **Cross Entropy**, also known as *Negative Log Likelihood*. Given n possible output classes, it is mathematically expressed as:

$$CrossEntropy = - \sum_{i=1}^n y_i \cdot \log(\hat{y}_i) \quad (4.9)$$

where y_i and \hat{y}_i are the expected and predicted classes, respectively. By construction, it heavily penalizes the predictions that are confident but wrong [74].

Gradient Descent

Once the most adequate cost function for the task is chosen, the *gradient descent* is the algorithm used to minimize it [75]. In other words, the gradient descent tries to find the optimal values for the weights ω and biases b in the neural network that make the errors between the predictions and the expected classes as small as possible.

Neglecting the biases and supposing for simplicity that the cost function J is only dependent on the weights ω , a possible graphical representation is in Figure 4.17 (A).

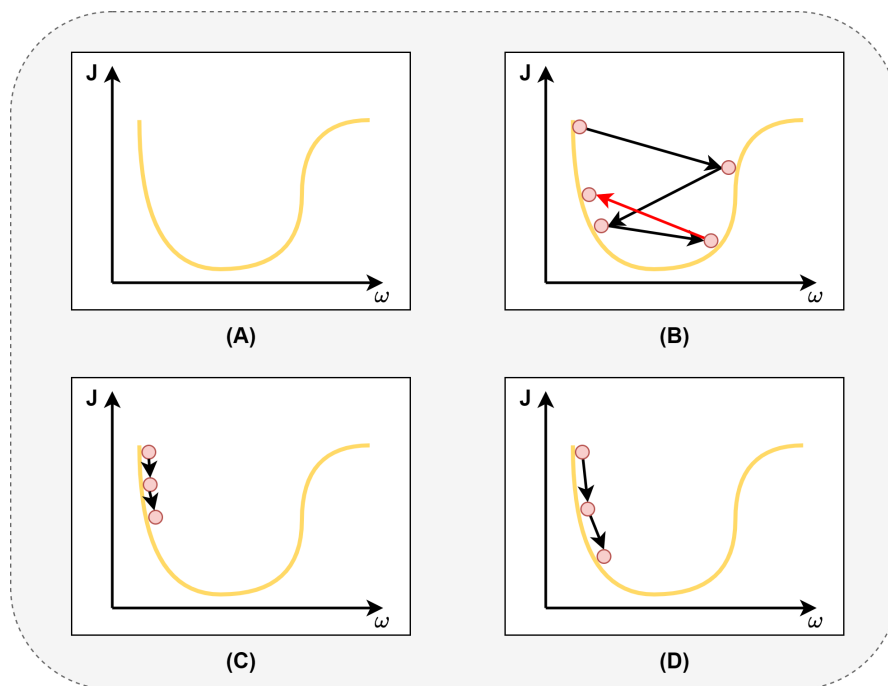


Figure 4.17: The Gradient Descent.

The gradient descent can be visualized as a ball rolling down the hill outlined by the cost function. The algorithm accepts a parameter set by the user, the so called **learning rate** α . This quantity specifies how fast the gradient descent has to move down the slope of J .

A high learning rate, as in Figure 4.17 (B) lets the algorithm move too fast along the slope, possibly never converging to the value of the minimum. A low α , on the other hand, might let the gradient descent move too slowly and converge to the minimum after a long time, wasting computational power 4.17 (C). Finally, Figure 4.17 (D) shows the optimal setting of the learning rate.

The process of optimization of the parameters was introduced by Hinton *et al.* in their paper *Learning representations by back-propagating errors* [76]. It starts operating from the last layer and continues until the first one, reason for which it is referred to as **Back-propagation Algorithm**.

The update of optimization parameters ω and b by means of the gradient descent is mathematically expressed as:

$$\omega_{new} = \omega_{old} - \alpha \cdot \frac{d}{d\omega} J(\omega, b) \quad (4.10)$$

$$b_{new} = b_{old} - \alpha \cdot \frac{d}{db} J(\omega, b) \quad (4.11)$$

where the subscript *old* refers to the parameters before the update and *new* to their value right after. The notations dJ/db and $dJ/d\omega$ refer to the partial derivatives of the cost function J with b s and ω , and the learning rate is represented by α .

The intuitive meaning of the formula is the following: the partial derivative finds the direction towards which the fastest increase of the parameter taken into account (e.g., the weight or the bias) takes place. The sign is then reversed, now heading towards the direction with the highest *decrease*, and following it with pace defined by the learning rate α .

In reality, the bowl model for the cost function from Figure 4.17 (A) only holds in some specific cases, when a linear regression¹ is taken into account. Finding curves like the one in Figure 4.18 is more likely for neural networks.

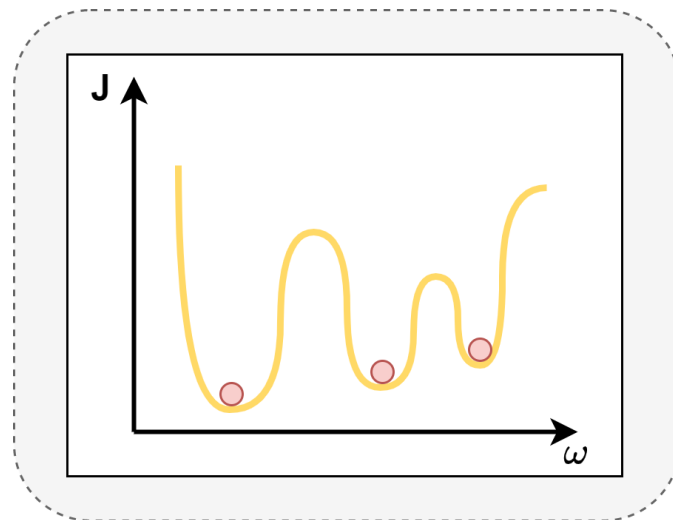


Figure 4.18: Non-convex cost function.

The cost function here is not convex, as it features multiple local minima and a global minimum. The gradient descent might not always converge to the lowest point in the curve, but rather to a local minimum, hence to a sub optimal solution.

¹By “linear” researchers refer to a method that attempts to find a line (i.e., a curve with polynomial degree of one) that fits the data with minimum error as a solution for the regression problem.

Common Optimizers

Most *optimizers* used in Deep Learning are based on gradient descent algorithm and its variants. Here some of the most adopted are reported, although researchers often propose new ones.

The **Stochastic Gradient Descent** (SGD) represents the most basic optimizer in Deep Learning. It computes the gradient of the cost function with respect to the parameters ω and b using a small portion (i.e., a *batch*) of training data and then updates the parameters accordingly. Due to its simplicity it is able to converge faster than other optimizers, hence it is often used on large training datasets [77].

The **RMSprop** optimizer adapts the learning rate α using a moving average of the squared gradient instead of the gradient itself. This helps to prevent the learning rate from becoming too small, with a consequent high computational time [78]. It is particularly effective for tasks that require a learning rate that changes over time, such as Natural Language Processing [79].

The **Adam** optimizer calculates the mean and the variance of the gradient, and then updates the parameters according to a combination of the two computed quantities. It is particularly effective for training neural networks with complex architectures [77].

Overfit and Underfit

As previously stated, the goal of the training is to optimize weights and biases in the neural network, in order to minimize the chosen cost function. Once the training is over, the network should be able to perform its task (e.g., regression, classification and so forth) on yet-unseen data as accurately as on the training data. In other words, a successful model is able to *generalize* to new data.

Once the neural network is free to work on new data, two main problems might arise, known as *overfit* and *underfit*. They are both related to the complexity of the network, i.e. the number of perceptrons it is made of.

The neural network is affected by **overfit** when it performs very well during training, but fails when seeing new data from the same domain. Figure 4.19 depicts an example of overfit.

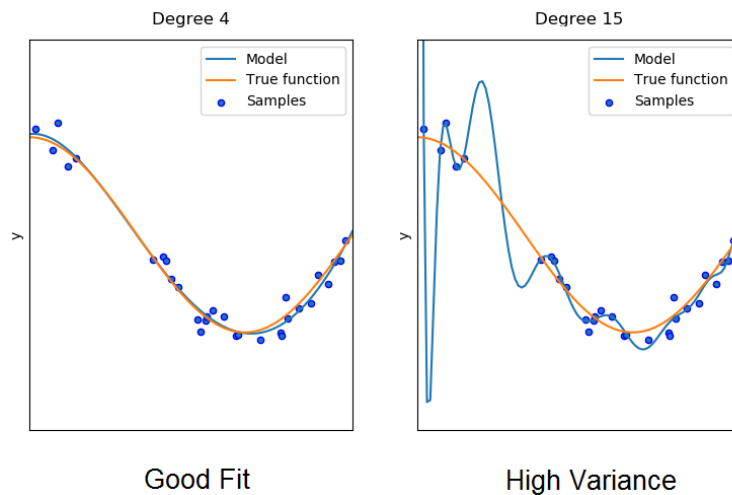


Figure 4.19: Overfit [21].

The neural network on the left has low complexity (e.g., less layers) and fits data well, with a low distance between the dots (i.e., the actual data) and the curve (i.e., the predicted data). If a new dot is added, the distance between the former and the curve will be low. The network on the right features a higher complexity and manages to have an even smaller distance between the dots and the curve.

Between the two models, the second one seems to fit data better than the first one. However, it tries to fit the dots one by one, not recognizing the underlying function that describes the data distribution. This means that when adding a new dot according to the data distribution, this model will show a high error (i.e., a large distance between the new dot and the curve).

Models affected by overfit are characterized by a high variance, depicted as large swings in the Figure, and cannot generalize to the new unseen data. The problem is empirically encountered with a high accuracy in the training phase followed by poor performance when new data is fed to the network.

The main solutions to address the problem are the gradual lightening of the model, attempting to reduce its complexity (e.g., dropping perceptrons in the different layers) or with regularization techniques. Among such techniques, the *early stopping* method is often employed as it consists in halting the training process as soon as performance does not improve, thus preventing a high and undesired complexity of the model.

The opposite problem is called **underfit**, depicted in Figure 4.20.

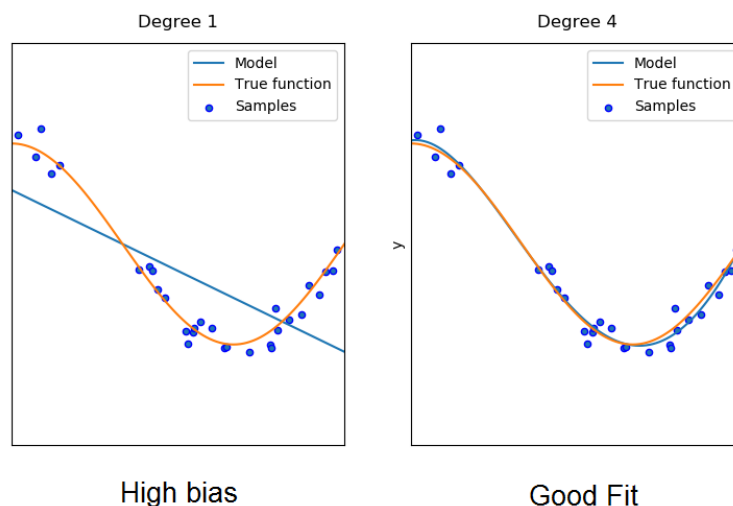


Figure 4.20: Underfit [21].

Here, the model on the left is too simple with respect to the complexity needed to fit the data. Networks affected by this problem feature poor performance already in the training phase, making underfit more easily detectable with respect to overfit. A solution to the problem of underfit might be the gradual increase of the number of perceptrons in the neural network, with a consequent higher complexity provided by the model.

4.2.4 Adjusting the Networks: Hyperparameters Tuning

Weights and biases are not the only parameters to set for building a model that successfully fits the data. When finding the solution to a regression or classification problem, multiple neural networks, each with characteristic settings, can be taken into account.

Hyperparameters are additional knobs to handle when looking for the best model. They consist in configurations of the neural network that are set a priori, before the training begins. The main hyperparameters in Deep Learning include:

- **Learning Rate**, as the convergence of the cost function depends on its value;
- **Number of Layers**, since models that feature higher complexity can incur into the overfit problem, while a low complexity might imply the underfit of the data;
- **Perceptrons per Layer**, as a wider neural network might capture complex patterns at the cost of a high computational time;
- **Activation Functions**, as different functions introduce different levels of non-linearity in the model;
- **Batch Size**. It sets the dimension of the portion of the training set to evaluate before updating the parameters ω and b . High values imply that the model is updated few times during the training, and make the training itself faster but might not catch the complexity needed by the data;
- **Number of Epochs**. It defines how many times the model should perform an entire iteration over the training data;
- **Optimizer**, as models can achieve benefits from the different methods that minimize the cost function.

Once ranges for the hyperparameters of interest are individuated, the so called *search space* is defined, and different search strategies can be adopted.

The **Grid Search** consists of an exhaustive exploration of the search space. Taking into account all possible combinations of hyperparameters, it suffers high dimensional spaces (i.e., spaces defined by the combination of multiple hyperparameters). It benefits from the usage of Graphics Processing Units (GPUs) as the exhaustive search can be parallelized [80].

The **Random Search** does not scan the entire search space, but rather picks points from it. Each point corresponds to a combination of the hyperparameters to be taken into account. With constrained computational resources it can outperform the grid search in large spaces [80].

Genetic Algorithms leverage the principles that regulate the process of natural selection in order to explore the search space of the hyperparameters. At each step, a series of individuals, each resembling a point in the search space, is evaluated with a score. Successive recombinations of the successful parameters generate new individuals to evaluate at the next steps [80].

4.3 Deep Learning applied to Side-Channel Analysis

Deep Learning can follow two different approaches, *Supervised* and *Unsupervised* Learning.

Supervised Learning, as mentioned in Section 4.2.3, is composed of two stages, the training and attack phases. During the first one, a set of labelled data is used to train the network. Here, the weights and biases are optimized in order to minimize a cost function. During the second phase, the trained neural network performs the required task on unseen data.

Unsupervised Learning, on the other hand, only features a single phase. Here the neural network has to extrapolate the features of interest from the data and perform the task, as the labelled data is not employed during the training.

The two approaches present similarities with respect to the attack modalities adopted in Side-Channel Analysis described in Section 3.2.3. Profiled attacks, just like Supervised Learning, feature a preliminary phase during which knowledge about the target device is acquired, and a second stage when the attack is accomplished. Non-profiled attacks, instead, are intrinsically similar to Unsupervised Learning, as the task is performed directly, without a preliminary phase for modelling the target device.

Researchers observed the similarities between the Deep Learning approaches and the attack modalities in classic Side-Channel Analysis and elaborated two main scenarios followed in the literature. The first, displayed in Figure 4.21, represents an attack performed following the Supervised Learning approach.

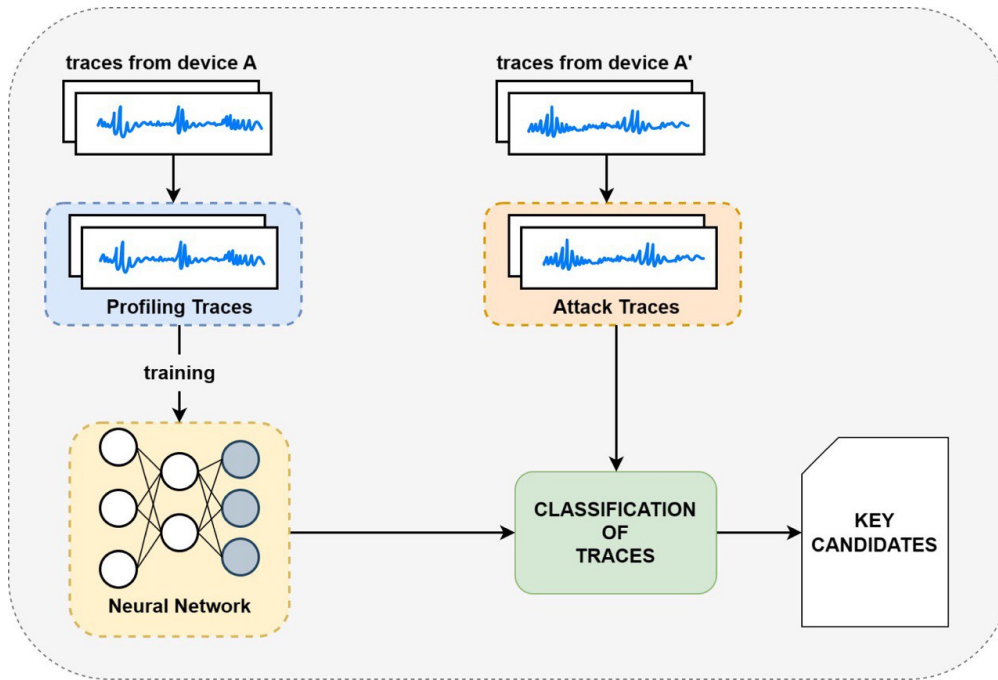


Figure 4.21: Supervised Learning for DLSCA [22].

The analysis involves the actual target of the attack (i.e., device A') and a replica, under full control of the hacker (i.e., device A). Power traces are collected from device A, labelled with the key they refer to, and fed to the designed neural network. This process represents the training phase of the network. Few traces are then collected from the actual target device, which is not controlled by the hacker. The knowledge gained through the training phase on device A is then leveraged for inferring the key used by device A'. For a successful classification the training should be performed with a replica device as similar as possible to the actual target.

Side-Channel Analysis according to the Unsupervised Learning paradigm only involves one device, the actual target of the attack, as depicted in Figure 4.22.

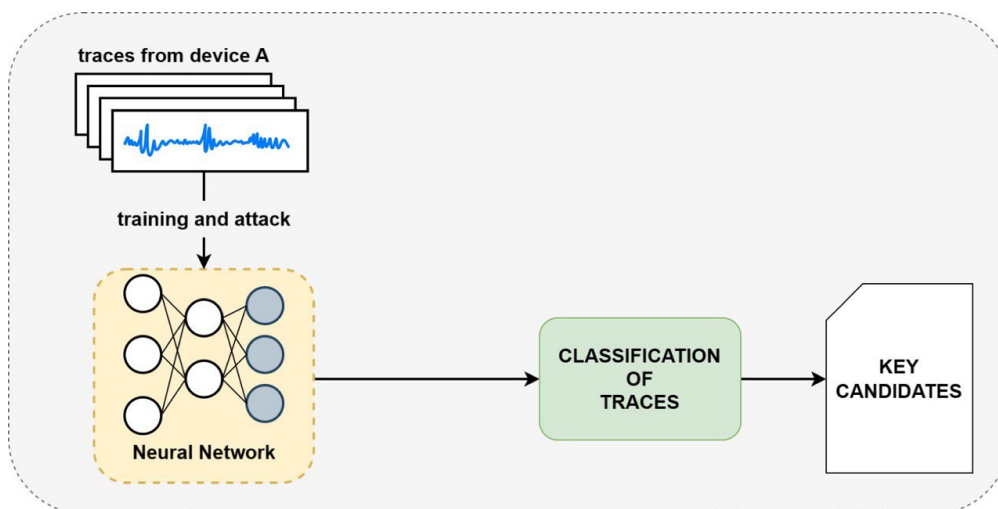


Figure 4.22: Unsupervised Learning for DLSCA [22].

The traces collected by the hacker are used both for training the neural network and

for leading the attack. In contrast with the first approach, the hacker does not need labelled data for the training. The only needed assumption for the analysis is that the attacker can collect a large quantity of power traces from the target. On the other hand, this approach is generally considered more complex and few studies in this directions exist [54].

4.3.1 Power Models

Most works in the literature refer to three main power models adopted in Deep Learning Side-Channel Analysis when attacking AES:

- The **Hamming Weight**. The Intermediate Value (IV) on 8 bits can generate up to 9 different Hamming Weight groups, from HW=0 to HW=8. Inferring the byte value of the IV in classic SCA corresponds to a classification task in DLSCA. Here, the nine groups are considered as possible output classes among which the network has to discriminate. Since the Hamming Weight generates few output classes, the underlying classification problem is considered simple and consequently the power model is widely spread in the literature;
- The **Hamming Distance**. Rather than a guess on the Intermediate Value it evaluates a *pair* of subsequent IVs. In a similar way to the Hamming Weight, it generates 9 output classes, from HD=0 (when the pair of bytes implies no bit flips) to HD=8 (when all bits flip in the byte when passing from IV_1 to IV_2). The classification problem is similar to the one originated by the HW model. However, it is less common in the literature since it works on pairs of Intermediate Values.
- The **Identity** Model. It directly considers the byte value of the IV, thus leading to 256 possible output classes, from 0x00 to 0xFF. The underlying classification problem is retained harder due to the elevated number of outcomes.

4.3.2 Metrics

Evaluating the performance of a model during a classification problem is a crucial aspect of the overall process, and it is achieved through *metrics*. In Deep Learning, two main metrics are employed, the *loss* and the *accuracy*.

The **loss**, also known as cost function, was exhaustively analysed in Section 4.2.3. It corresponds to the uncertainty of a prediction and is based on how much the model's output differs from the expected value.

The **accuracy** is a metric that expresses how often, out of the total guesses, the prediction was correct. In a classification problem with more than two output classes it is mathematically formulated as:

$$Acc = \frac{CP}{CP + WP} \quad (4.12)$$

where CP and WP are the number of correct and wrong predictions, respectively. An example of results deriving from a multi-class classification problem is in Figure 4.23, which depicts the table known as *Confusion Matrix*.

		PREDICTED CLASS			
		A	B	C	D
ACTUAL CLASS	A	6	0	1	2
	B	3	9	1	1
	C	1	0	10	2
	D	1	2	1	12

Figure 4.23: Confusion Matrix for multi-class classification problem.

Here the correct predictions correspond to the elements of the matrix where the predicted class coincides with the actual class, and they are located on the diagonal, thus leading to $CP = 6 + 9 + 10 + 12 = 37$. The total number of mispredictions, in red, is equal to $WP = 15$. The accuracy is therefore:

$$Acc = \frac{CP}{CP + WP} = \frac{37}{37 + 15} = 71.15\% \quad (4.13)$$

Deep Learning Side-Channel Analysis Metrics

The Hamming Weight model is the most common in the literature [81]. However, as pointed out in Section 3.3.3, it generates uneven groups (e.g., one byte has $HW=0$, whereas 70 bytes feature $HW=4$). This phenomenon was studied by Picek *et al.* in their paper *The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluations*, in which they state that classic ML and DL metrics do not represent reliable performance evaluators [81]. A neural network can have a high accuracy simply because it might tend to classify many Intermediate Values into HW class 4, which is far more likely than other classes in the distribution. However, this classification does not bring advantages to the attacker, as there is no real dependence between the Intermediate Value and the HW class predicted by the model. For this reason, researchers developed new metrics that can evaluate performance of models in DLSCA, the *Success Rate* and the *Guessing Entropy*.

The **Success Rate** (SR) is a metric that evaluates the percentage of successful attacks on the key for a particular number of traces. The metric, often expressed as a percentage, ranges between $[0,1]$ and intuitively tends to 1 with a larger number of traces. The plot in Figure 4.24 depicts its behavior.

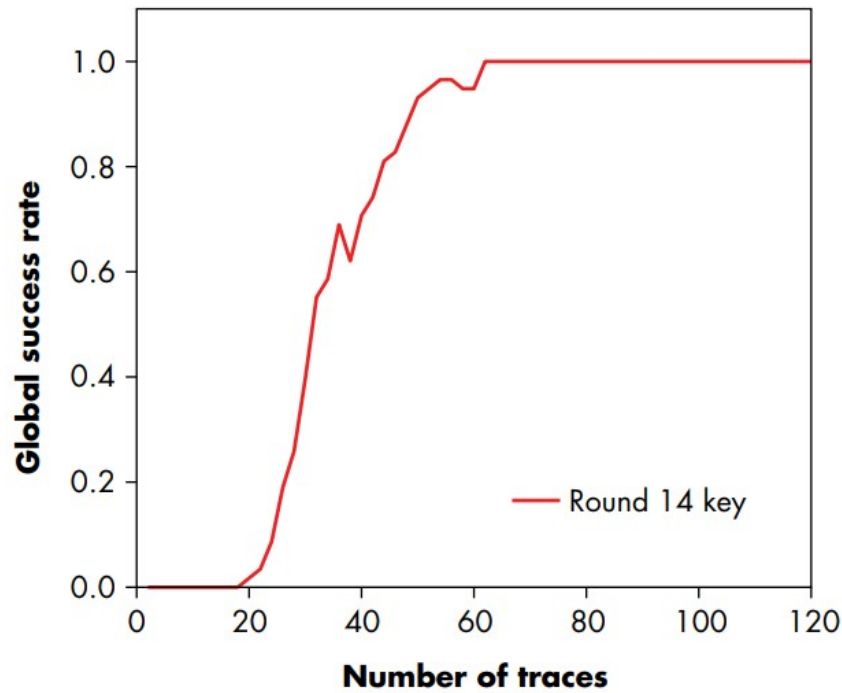


Figure 4.24: Success Rate [12].

In that specific example, with 40 traces, the key is correctly recovered nearly 80% of the time. With 60, the rate is about 95%.

When the metric is used for evaluating the effectiveness of the model on a single key byte, it is referred to as *Partial Success Rate* (PSR).

As stated in Section 3.3.3, a Side-Channel Attack often provides more complete information than simply “key was ABC” or “key was not found”. Each key guess is associated with a confidence figure, into a ranked list. The **Guessing Entropy** (GE) evaluates the number of incorrect guesses in the ranked list, for a particular number of traces. The metric, depicted in Figure 4.25, is mainly used for evaluating the effectiveness of the model on a single byte. When used for this purpose, it is called *Partial Guessing Entropy* (PGE).

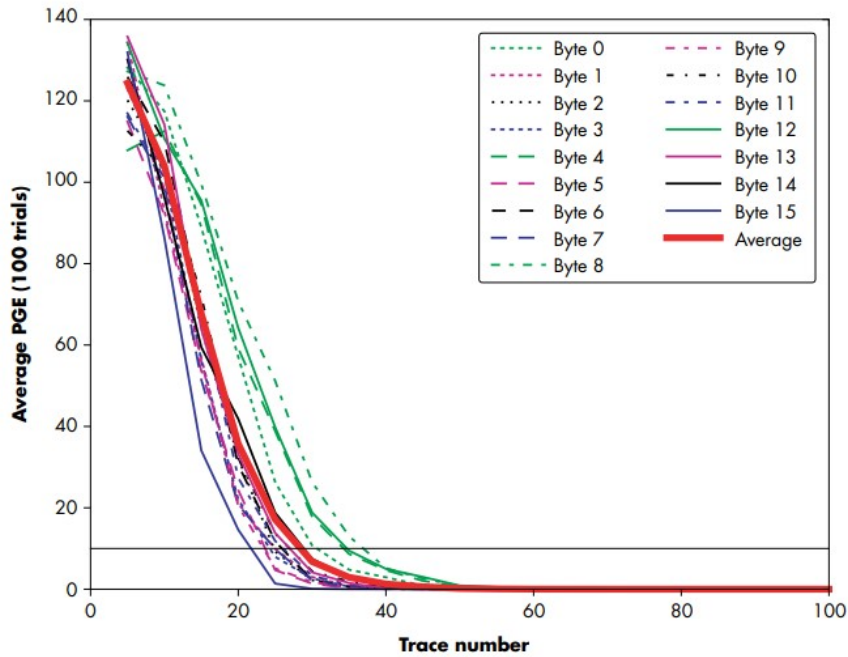


Figure 4.25: Partial Guessing Entropy [12].

PGE ranges between $[0,255]$, where 0 means that the correct key byte was ranked as first in the list².

4.3.3 Datasets

The main drawbacks of Deep Learning stem from the resources required by the training of neural networks, the computational power and the high volume of data.

The former can be solved by employing GPUs rather than general purpose processors, as many tasks during the training can be parallelized. For instance, as stated in the previous Section, Grid Search can be split into multiple threads, each computing the effect of a combination of hyperparameters from the search space.

As to the latter, in recent years researchers have proposed labelled *datasets*, large collection of traces along with their expected class, for training neural networks. Providing traces features a double advantage, as they can be used by people that do not own the instrumentation to perform the collection (tools such as oscilloscopes might be too expensive for hobbyists that approach SCA out of academical contexts) and they serve as benchmark for new experiments. As a matter of facts, datasets constitute a common criterion to evaluate the efficiency of models, since different researchers can assess how hard it is to crack the same key leveraging training on the very same traces.

ASCAD

ASCAD [82], contraction for “ANSSI SCA Database”, is a project developed by the French National Cybersecurity Agency. It attempts to provide collections of data inspired to the *MNIST* database, well known dataset for Artificial Intelligence classification problems.

²Often in the literature the insertion of the correct key byte as first in the ranking is referred to as “unitary guessing entropy”, thus making PGE range between $[1,256]$.

The first release is composed of the two databases `ATM_AES_v1_fixed_key` and `ATM_AES_v1_variable_key`, also known as ASCAD-F and ASCAD-R. They consist of traces collected from the execution of AES on an `ATMega8515`, which does not feature any SCA countermeasure.

ASCAD-F provides 50,000 traces for the training of the neural network and 10,000 for evaluating its performance. The key is kept fixed to the value in Table 4.1.

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Byte Value	0x4D	0xFB	0xE0	0xF2	0x72	0x21	0xFE	0x10	0xA7	0x8D	0x4A	0xDC	0x8E	0x49	0x04	0x69

Table 4.1: Key used in ASCAD-F.

ASCAD-R consists of 300,000 power traces, divided into 200,000 for the training phase and 100,000 for the attack phase. The key is kept fixed for one third of the collection and randomized for the rest. The higher cardinality of this collection with respect to ASCAD-F is due to the difficulty for the model to learn features that derive from traces with different keys, as they introduce a more complex pattern.

The second release provides two databases obtained thanks to power measurements during the execution of AES on `STM32F303RCT7`. The device is a 32-bit microcontroller lightly protected against Side-Channel Analysis as software masking and jittering countermeasures are present. ASCADv2 provides the measurement of 800,000 encryptions, as the neural network needs hundred thousands of traces to model features originated by a microcontroller with high data bus width and equipped with countermeasures.

Each of the datasets in the two versions shows the same structure, inspired by the MNIST database and depicted in Figure 4.26 [23].

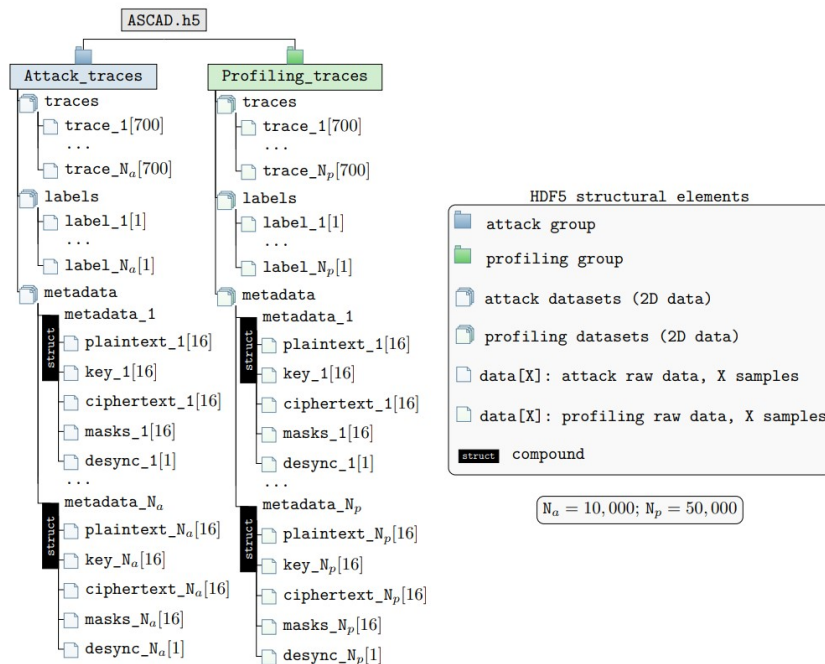


Figure 4.26: ASCAD structure [23].

In the file two groups are present, `Profiling_traces` and `Attack_traces`. Each

group is composed of three fields: `traces`, `labels` and `metadata`.

Each trace is the collection of 700 samples measured at the output of the first round of AES. This cut is performed to provide light traces which can be processed easily even with general purpose computers.

Labels represent the expected output class of the classification problem. In ASCADv1 the Hamming Weight is employed and therefore the label consists in the HW of the output of the `S-Box` at the first round of AES.

The Metadata field aggregates the plaintext, key and corresponding ciphertext. Possible countermeasures are also taken into account, as the mask and the desynchronization by a constant amount are explicated.

The database is stored in `.h5` format, which is a standard for Machine Learning datasets.

DPA Contests

The four DPA contests are competitions called by the *Telecom ParisTech* and *AIST*. The organizers provide traces that allow participants to recover the key, thus comparing in an objective manner their attack algorithms. DPAv1 and DPAv3 are now closed and do not let researchers submit models. DPAv2 and DPAv4 are still open and regard hardware and software implementations of AES-128, respectively. The second edition of the contest provides 1,000,000 unprotected traces on a `SASEBO-GII` board. The latest contest features 32,000 profiling traces with masking countermeasures on a `SAKURA-G` board.

Further details about the four contests can be found at <https://www.dpacontest.org/home/>.

4.3.4 Frameworks

As soon as studies demonstrating the effectiveness of Deep Learning in Side-Channel Analysis were carried out, many researchers started exploring new solutions that could help recover keys with less traces and fewer assumptions. As a result, DLSCA is a research field in rapid growth, set in between the domains of Artificial Intelligence and Embedded Cybersecurity. In many cases researchers expertise one domain but lack competencies in the other, due to the rare interactions between the two.

Starting from 2019, researchers have attempted to propose frameworks that could ease up the training and deployment of neural networks for Side-Channel Analysis.

The AISY Framework by *Delft TU*

Developed by the AisyLab group of the *Delft University of Technology*, it represents one of the most complete frameworks in the DLSCA panorama. The AISY Framework [24] was released in 2021, it is regularly updated and provides a plethora of features.

It is possible to create Python attack scripts, as illustrated by the listing in Figure 4.27.

```

import aisy_sca
from app import *
from custom.custom_models.neural_networks import *

aisy = aisy_sca.Aisy()
aisy.set_resources_root_folder(resources_root_folder)
aisy.set_database_root_folder(databases_root_folder)
aisy.set_datasets_root_folder(datasets_root_folder)
aisy.set_database_name("database_ascad.sqlite")
aisy.set_dataset(datasets_dict["ascad-variable.h5"])
aisy.set_aes_leakage_model(leakage_model="HW", byte=2)
aisy.set_batch_size(400)
aisy.set_epochs(20)
aisy.set_neural_network(mlp)
aisy.run()

```

Figure 4.27: Example script using the AISY framework [24].

In the image, the script performs an attack on the third byte of the key, selecting the Hamming Weight as power model. By default the attack point corresponds to the output of the S-Box at the first round of AES, although it can be easily changed by the user thanks to the `aisy.set_aes_leakage_model()` function. The chosen neural network (mlp), defined by the user, is trained with traces from the *ASCAD-R* dataset. The training features 20 iterations of the whole training dataset (`epochs = 20`). During each of them, the weights and biases of the network are updated after every 400 new traces (`batch_size = 400`).

The researchers from *Delft TU* promise to update metrics and leakage models regularly, as DLSCA is an emerging research field and new disruptive discoveries could affect significantly its State of the Art. Furthermore, a high level of customization is provided, as the user can define their own metrics, leakage models and cost functions.

The overall framework is complete, as the user can explicitate every parameter (e.g., attack point, evaluation metric, and so forth) before performing the attack. Its only impactful drawback stems from its maturity, as multiple bugs were found out during the development of this work. Nevertheless, the researchers from *Delft TU* often made quick fixes on demand following our bug notices.

The full documentation is available at https://aisylab.github.io/AISY_docs/.

SCAred by *eShard*

It is a Python library developed by the French company *eShard*, which allows to easily launch Side-Channel Analysis attacks. An example is depicted in Figure 4.28.

```

import scared
delta = scared.aes.selection_functions.encrypt.OutSboxFirstRound()
cpa_attack = scared.CPAAttack(
    selection_function=delta,
    model=scared.HammingWeight(),
    discriminant=scared.maxabs
)
ths = scared.traces.read_ths_from_ets_file('dpa_v2.ets')
container = scared.Container(ths, frame=slice(2340, 2395), preprocesses=[])
cpa_attack.run(container)

```

Figure 4.28: Example script using the SCAred framework [25].

The code specifies that the attack point is the Intermediate Value of the first round

of AES and that the Hamming Weight model is deployed for the analysis. Profiling traces are obtained thanks to the collections provided by DPAv2 contest, cut between samples 2340 and 2395.

SCAred is a very flexible solution as it allows users to define customized power models and attack points. Examples of attacks running in the Jupyter Environment are available at <https://mybinder.org/v2/g1/eshard%2Fscared-notebooks/master>. The full documentation is available at <https://eshard.gitlab.io/scared/index.html>.

SCALD by Google

SCALD, acronym for “Side Channel Attacks Leak Detector”, is a framework developed in Python by Google.

Differently from the other frameworks presented in this Section, it does not help the user to ease up the deployment of attacks. On the opposite, it is particularly useful for defenders as it highlights the code fragments in software implementations of AES that leak exploitable information, as in Figure 4.29.

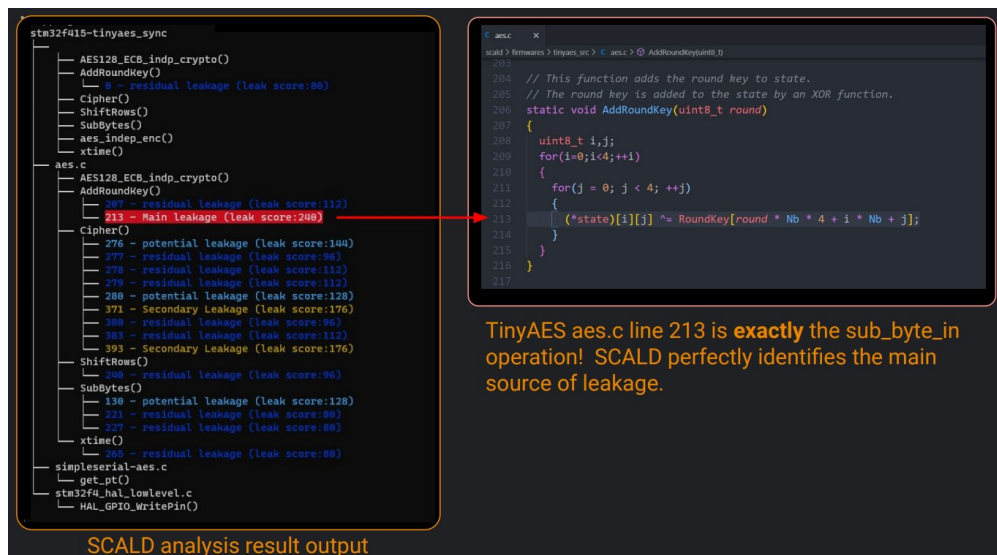


Figure 4.29: Leakage detection on TinyAES with SCALD [26].

The framework successfully identifies the line of code performing the AddRoundKey operation as main source of leakage, by assigning it a high leak score and highlighting it in red.

Information about SCALD is fragmentary, although the authors provide further details at <https://elie.net/scald>.

Chapter 5

Development

5.1 Motivation and Objectives

Deep Learning Side-Channel Analysis has gained significant relevance in recent years, drawing the attention of researchers from the domains of Embedded Systems Security and Artificial Intelligence. As a consequence, numerous research groups often propose new techniques that achieve attacks with fewer assumptions or improved performance. Nevertheless, the increasing attention in the research field was not accompanied by the *standardization* of experiments: many papers claim to outclass the State of the Art but lack fundamental details about the used traces, as well as the training employed for the different neural networks. Additionally, as Stjepan Picek points out during the seminar *Deep Learning for Side-Channel Analysis* [22], researchers often adopt shortcuts in academical contexts, which could lead to overestimating the obtained results. One of these misconceptions is represented by the usual setting deployed during the Supervised Learning experiments, depicted in Figure 5.1, which appears to be different from the ideal one illustrated in Figure 4.21.

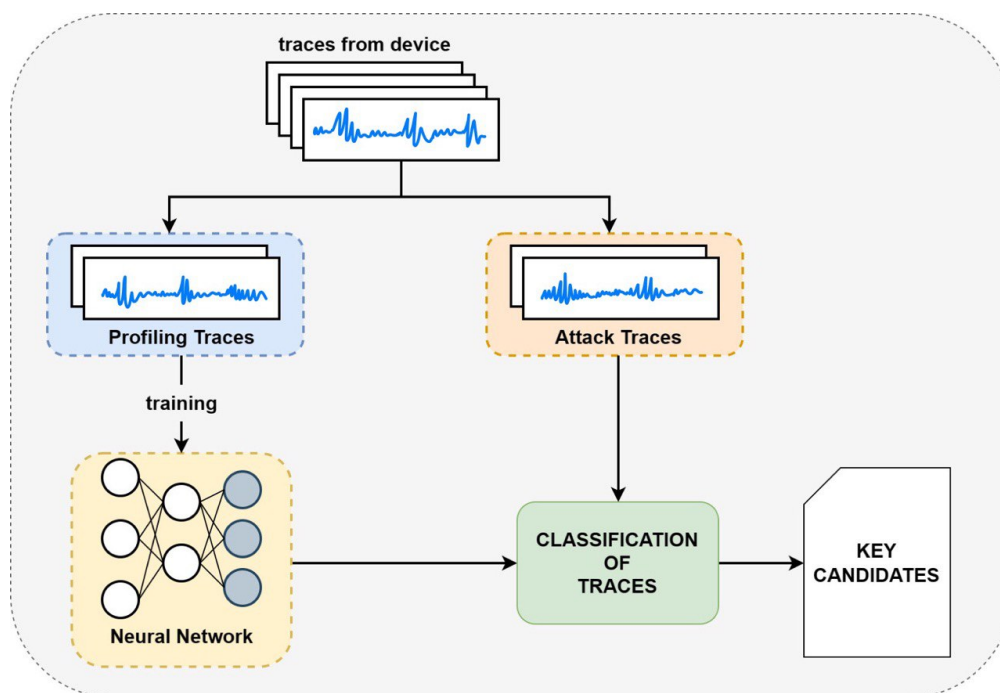


Figure 5.1: Shortcut Setting for Supervised Learning [22].

While in the ideal setting the profiling and attack traces are collected from two different devices, here the collection campaign is unique. Out of the single collection two groups of traces are created, for training and attack evaluation.

This shortcut, although seeming harmless, highly impacts the quality of the experiments: the model learns the features deriving from the key and the handled data but does not deal with the higher complexity that stems from the variations in the manufacturing process of the different devices. Conversely, when both devices are distinct, like in the case depicted in Figure 4.21, the data collected includes variations due to production inaccuracies which inevitably worsen the results of the experiment.

Very few researchers tackle this problem, known as **portability of the attacks** [22]. Perhaps the most impressive work in this direction is the experiment by Bhasin *et al.* described in their paper *Mind the Portability: A Warriors Guide through Realistic Profiled Side-channel Analysis* [83]. Here, attacks on the single device (i.e., as in the shortcut setting) and on different copies of the same device (i.e., as in the ideal one) are performed, demonstrating the higher complexity of the latter.

This thesis proposes a further step in this direction, evaluating the effects of attacks targeting devices that do not belong to the same family of those used for the training of the neural network. Cross-family devices show a higher level of dissimilarity with respect to the microcontrollers used by Bhasin *et al.*, not sufficiently studied in the literature. Three different models of microcontrollers from family PIC18FXXK42 by *Microchip* are used to train three neural networks in order to attack a device from family PIC18FXXK20.

The work impersonates three attackers with different degrees of commitment:

- Low effort. A simple MLP composed of two hidden layers is used to crack the key without prior knowledge of the power traces;
- Medium effort. The hacker analyzes the power traces and knowing the degree of complexity needed by the problem decides to employ a neural network deployed in a similar attack scenario;
- High effort. A neural network, product of a fine tuning on the available traces, is chosen to lead the attack.

5.2 Experimental Equipment

16-bits and 32-bits microcontrollers are becoming more and more popular in the market, as companies managed to provide them with capabilities to run resource-hungry applications, such as Digital Signal Processing (DSP) [84]. However, not all applications need cutting-edge technologies. In fact, most do not. 8-bit microcontrollers are still relevant in the market due to their low cost and satisfactory computational power, and projections show they will keep pace in the microcontroller industry in the upcoming years, as shown in Figure 5.2.

North America microcontroller market size, by product, 2016 - 2027 (USD Billion)

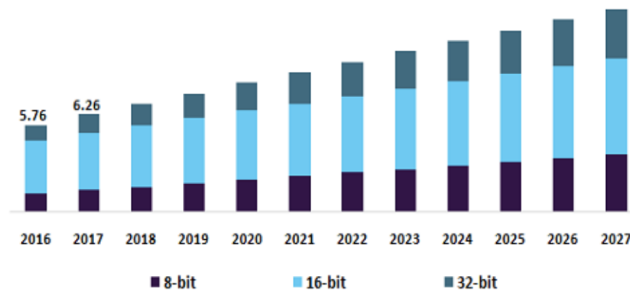


Figure 5.2: Relevance of 8-bit microcontrollers in the North American market [27].

5.2.1 8-bit MCUs by *Microchip*

Choosing the target microcontrollers was the preliminary step of the experience. At first, several companies were taken into account, such as *ARM*, *ST*, *Nordic* and *Microchip*. All of them offer a plethora of 8-bit and 16-bit devices, although, as pointed out in Section 3.3.3, the ultimate decision came down to 8-bit devices because of the lower complexity of their power traces.

A necessary feature is the memory size, as all microcontrollers should at least contain the *Tiny-AES* software implementation. Devices with at least 2KB of RAM and 256B of ROM were chosen, as some margin is provided with respect to the minimum requirements.

A further important criterion for the choice of the adopted microcontrollers stems from the possibility to have a single device to program all of them, rather different programming tools. *Microchip* sells a Development Board that allows chips from their 8-bit PIC line to be plugged in and programmed, a versatile solution to handle multiple microcontrollers.

The board, whose top side is illustrated in Figure 5.3, is called “Curiosity High Pin Count Development Board”. It includes an integrated programmer and debugger and requires no additional hardware to get started.

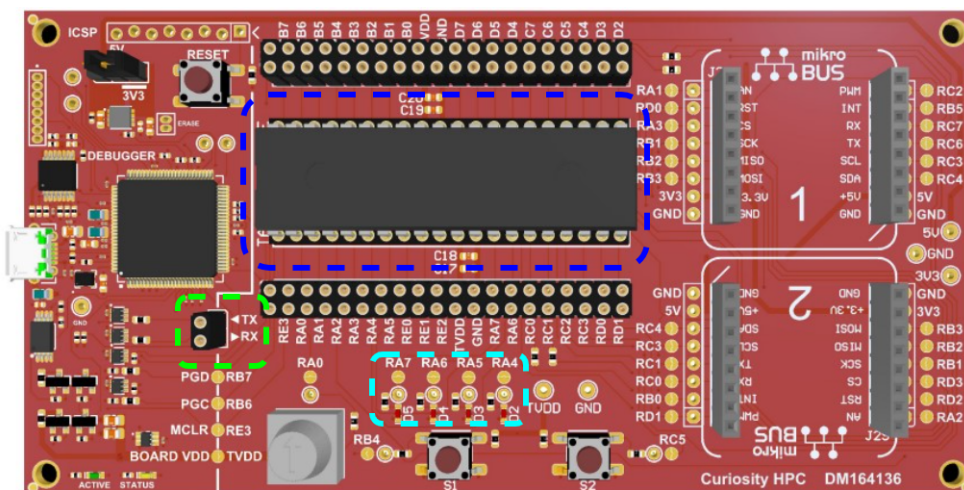


Figure 5.3: Top view of the HPC Development Board by *Microchip* [28].

It features a slot for the allocation of the microcontroller (*socket*), highlighted in blue in the aforementioned image. Only DIP and PDIP chip packages, with 28 and 40 pins

respectively, can be plugged in. Four LEDs are provided for easy debugging, shown in light blue in Figure 5.3. Furthermore, a serial interface, shown in green, allows data exchange between the microcontroller and the host computer connected to the HPC board, both in reception and transmission.

As to microcontrollers, the final selection taking into account the constraints mentioned above is the following:

- PIC18F27K42, PIC18F47K42, PIC18LF45K42, PIC18LF46K42, PIC18F26K42 and PIC18F45K42 from the PIC18 K42 family;
- PIC18F46K40 and PIC18F27K40 from the PIC18 K40 family;
- PIC18F46K20 from the PIC18 K20 family.

5.2.2 The *ChipWhisperer*TM Toolchain

The collection of power traces requires few tools, an oscilloscope and a resistor. The former, although present in most laboratories of electronics due to its numerous applications, is very expensive (a quick research shows prices starting from 1000\$) and out of reach for hobbyists or people that approach Embedded Cybersecurity out of academic contexts.

NewAE proposes cheap starter kits known as *ChipWhisperer*TM that allow to replace the aforementioned tools, as well as a software framework that helps the attacker perform the analysis.

The *ChipWhisperer*TM Hardware

Multiple starter kits are available in their website [29]. *ChipWhisperer-Nano* is the lowest-cost option (around 50\$) although shows limitations on the types of attacks that can be performed, as the target microcontroller is fixed and cannot be replaced.

A more flexible solution is represented by *ChipWhisperer-Lite* (around 250\$), a single device that integrates high speed power measurement, as well as programming capabilities for most chips designed by leading companies in the market.

This product, depicted in Figure 5.4, can be conceptually divided into two parts.

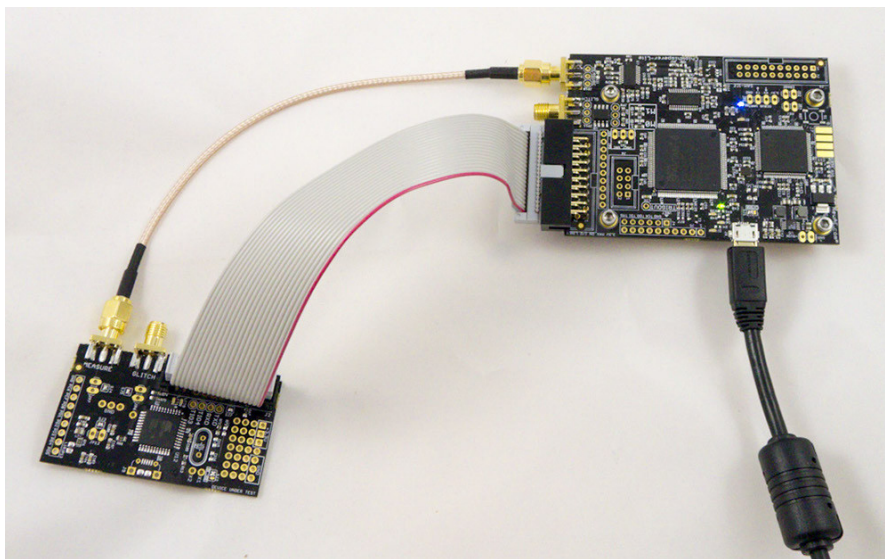


Figure 5.4: *ChipWhisperer-Lite* by *NewAE*.

A smaller PCB embeds the target device, object of the power analysis (the available integrated targets on the website are ATXmega128D4-AU, STM32F303RCT6 and STM32F030F4P6). On the other hand, the largest PCB embeds the capture logic, which is able to program the target device (via the flat cable in grey) and measure its power consumption during the execution of encryption operations (through the coaxial cable). Although the two PCBs are sold connected together, they can be detached from each other. This allows the Side-channel analyst to connect and attack different off-the-shelf target microcontrollers.

A closer look to the capture board of *ChipWhisperer-Lite* is in Figure 5.5.

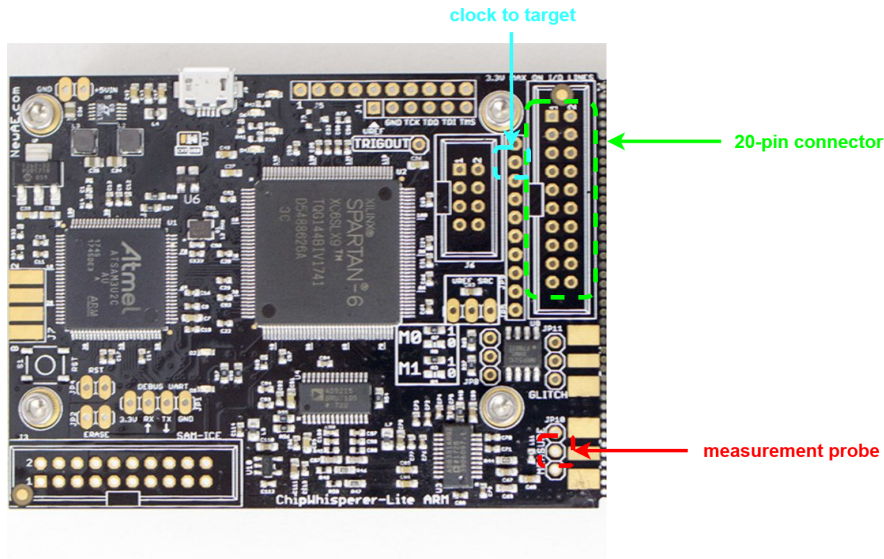


Figure 5.5: The Capture Board.

The capture board features a clock generation system that is able to drive, through the pin highlighted in light blue color, a clock signal to the target with frequency between $5MHz$ and $200MHz$.

The pin highlighted in red is the input to the measurement module. It can sample power traces coming from the target device up to $105MS/s$ ¹.

Finally, the 20-pin connector highlighted in green handles most of the communication from and to the target device. It allows the UART communication to send the plaintext and receive the ciphertext. It also embeds the trigger signal used as a temporal reference for the start of the measurements.

The *ChipWhisperer*TM Software

NewAE also provides an open-source Python library for controlling the capture hardware and the communications with the target, accessible through a *Jupyter Notebook* environment. The Python API exposes the functions that are used to regulate the behavior of the clock generator, the trigger system and the measurement module. Moreover, it allows to set the plaintext and the key, as well as obtain the corresponding ciphertext along with the acquired power traces.

Additional Python libraries such as `numpy` and `matplotlib` constitute useful tools for the analysis of traces, as the former handles large numerical vectors and the latter is able to draw plots to outline traces and results.

¹By construction, the maximum frequency that can be measured by the capture board corresponds to one fourth of its maximum sample rate.

Further details can be found at <https://chipwhisperer.readthedocs.io/en/latest/index.html#api>,

The SimpleSerial Protocol

Designed by *NewAE*, it regulates the interactions between the capture board and the target device, which are always initiated by the former. It is delivered in two versions, v1.1 and v2.1. In both cases the communication is established through packets.

SimpleSerialv1.1 is the simplest of the two. Each packet is composed as follows:

```
[cmd, data_0, ..., data_n, terminator]
```

- The `cmd` field states the operation the packet refers to. The capture board can indicate 'p' to send a block of plaintext or 'k' to send the key. The target board sends the command 'r' to indicate that the current packet contains a response to an operation previously requested;
- The `data` field resembles the sequence of ASCII byte values of the payload. The number of allowed data bytes is defined by the user via firmware;
- The `terminator` of the packet consists in the '\n' character.

SimpleSerialv2.1 is more complex, as the package is constituted by the following fields:

```
[cmd, scmd, dlen, data_0, ..., data_n, CRC]
```

The novelties with respect to the first version consist in:

- The `scmd` field, which specifies additional options with respect to the command that change the performed operation accordingly (e.g., the user can specify that `scmd = 0x00` corresponds to AES while `scmd = 0x01` performs DES);
- The `dlen` field, that explicitates the length of the data sequence;
- The `CRC` field, a checksum on the packet that assures its correctness.

With both versions the user must define the callback function that is associated to each command, as in Figure 5.6.

```
1  #include "simpleserial.h"
2
3  uint8_t encrypt_plaintext(uint8_t cmd, uint8_t scmd, uint8_t dlen, uint8_t* data)
4  {
5      // perform the Encryption Process
6      return 0;
7  }
8
9  int main(){
10
11     // initialize system
12
13     simpleserial_addcmd('p', 16, encrypt_plaintext);
14
15     while(1)
16     |     simpleserial_get();
17
18 }
```

Figure 5.6: Example of code including SimpleSerial functions.

The example shows the callback function that encrypts the plaintext, from line 3 to 7 (which in this work will be filled with the encryption function provided by the C library `Tiny-AES`). The callback is tied to a command (`'p'` in the example) thanks to the `simpleserial_addcmd` function, provided by the protocol. The maximum length of data the callback can handle is specified as second parameter. Furthermore, the protocol provides the function `simpleserial_get()` (at line 16), that performs a continuous check on new packets.

These functions are part of the firmware that will be flashed to the target device. The attacker can initiate the communication thanks to Python functions defined in the *ChipWhisperer*TM API, which act at a higher layer of abstraction, e.g. `send_key(key_bytes)`.

During the development of the thesis the `SimpleSerialv1.1` protocol was used as it is more lightweight and easier to handle due to the absence of the `CRC` field.

5.3 Configuration of the Hardware Setup

With integrated targets the user can flash example code by *NewAE* directly from the *ChipWhisperer*TM. The firmware specifies what operations the microcontroller has to execute, i.e. what algorithms have to be analyzed, from encryption schemes to password checks [85].

5.3.1 Setting up the Target Board

When the capture board is used to analyze other microcontrollers, the firmware flashing process is customized to the selected chip. In the case of `PIC18` devices, the programming phase must take part through the *Curiosity HPC Development Board* previously described.

Peripherals and Files

Designed by *Microchip*, *MPLAB X* represents the Integrated Development Environment for `PIC` and `AVR` firmware programming. It allows to specify the modules of interest thanks to a graphical interface and generates C code that constitutes a starting point for the developer. During the development of the thesis, the following resources were employed:

- the **System module**. `PIC` microcontrollers feature multiple sources to generate the clock signal for the various components. The default is the clock signal deriving from its internal quartz oscillator. Nevertheless, the capture board by *ChipWhisperer*TM needs to drive the target's clock signal in order to synchronously² sample the power traces. For this reason, the “external clock source” mode was enabled. Furthermore, the frequency value of the external clock source is specified, equal to `16MHz`;
- the **UART module**. It is needed to handle the serial communication between the capture and target boards. The former starts the data flow by sending the plaintext³, the latter replies at the end of the `AES` algorithm with the ciphertext. The module is used in “asynchronous full duplex” mode, as two channels are present (`Tx` and `Rx`) and can be used simultaneously. The baud rate on each channel is `9600bps`;

²*ChipWhisperer*TM performs synchronous sampling with respect to the clock of the target device, as its probe module is able to tie the measurements to the rising edge of the victim's clock signal.

³The key is stored inside the device, no key transfer ever takes place, as it could expose unexpected vulnerabilities and jeopardize the experiment.

- the **GPIO module**. The rising edge of one pin in the peripheral serves as trigger to the capture board for the start of the power measurements. The trigger is set at the beginning of the AES encryption phase.

Additionally, the C project includes files from the two libraries `TinyAES` and `SimpleSerial`. The former represents the software implementation of the AES algorithm, configured as AES-128 in ECB mode, the latter describes the interaction between the capture and target boards.

5.3.2 Setting up the Capture Board

The capture board, due to its versatile nature, can be configured according to the different target devices that the user intends to analyze. The main operations to be performed regard the GPIO Peripheral and the External Clock Generation Module. According to the API, the functionalities of the pins highlighted in Figure 5.5 can be modified by the user in Python.

The GPIO pins belong to the part of the capture board highlighted in green in Figure 5.5, whose schematic is in Figure 5.7, on the right.

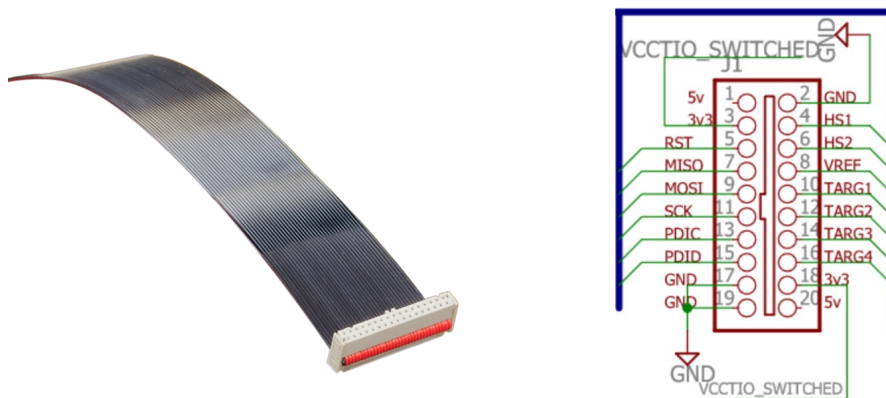


Figure 5.7: 20-pin connector and *ChipWhisperer*[™] schematic of the GPIO pins [29].

In order to allow the UART communication between the target and the capture boards, the pins `TARG1` and `TARG2` must be configured as `"serial_rx"` and `"serial_tx"`, respectively.

From the perspective of the capture board, the pin `TARG4` will be used as trigger input to start the power measurements. The user must set its initial value to high impedance by writing `"high_z"` and specify that the selected trigger mode is `"rising_edge"`.

The schematic of the External Clock Generation Module is depicted in Figure 5.8.

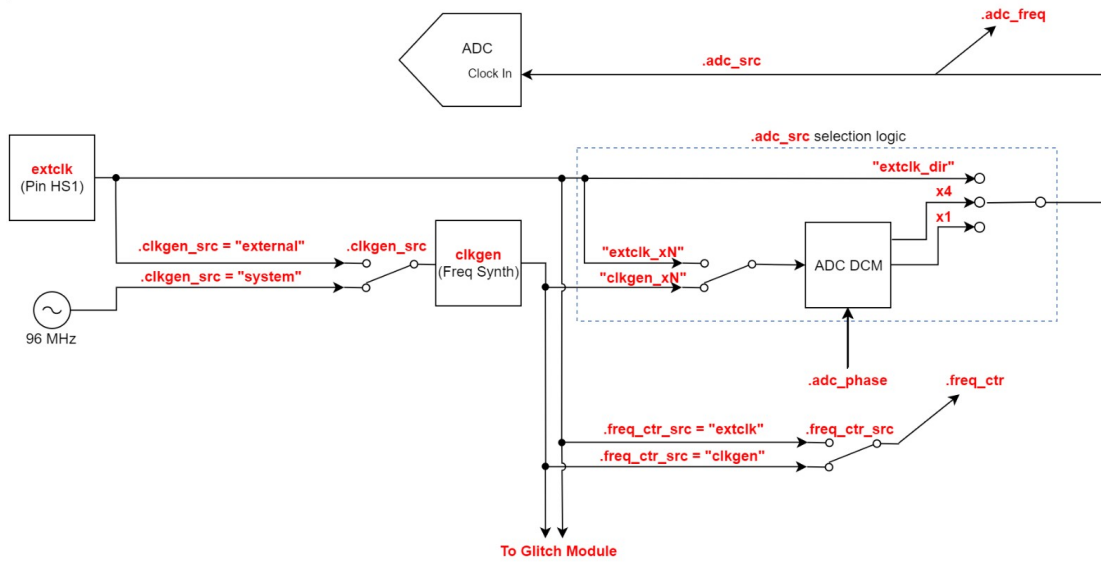


Figure 5.8: Schematic of the External Clock Generation Module [30].

The correct configuration should select `"system"` as clock source, set a frequency equal to `16000000` (i.e., 16MHz) and finally configure `"clkgen_x4"` as ADC source in order to have a sampling rate four times higher than the target's clock frequency, as pointed out in Section 5.2.2. Once the module is correctly set, the pin HS2 from Figure 5.7 can be selected as Clock Output by assigning to it the value `"clkgen"`.

5.3.3 Board-to-Board Interconnections

After the individual setup of both boards is complete and their functionalities are correctly defined, the physical connections can be established. The ideal measurement setup can be built in two ways, illustrated in Chapter 3.3 in Figures 3.8 and 3.9.

In contrast to the hardware setup deployed with native target devices (i.e., the flat connector and the coaxial cable), jumper wires are used to connect the capture board to the PIC18 microcontroller, as depicted in Figure 5.9 and outlined in Figure 5.10.

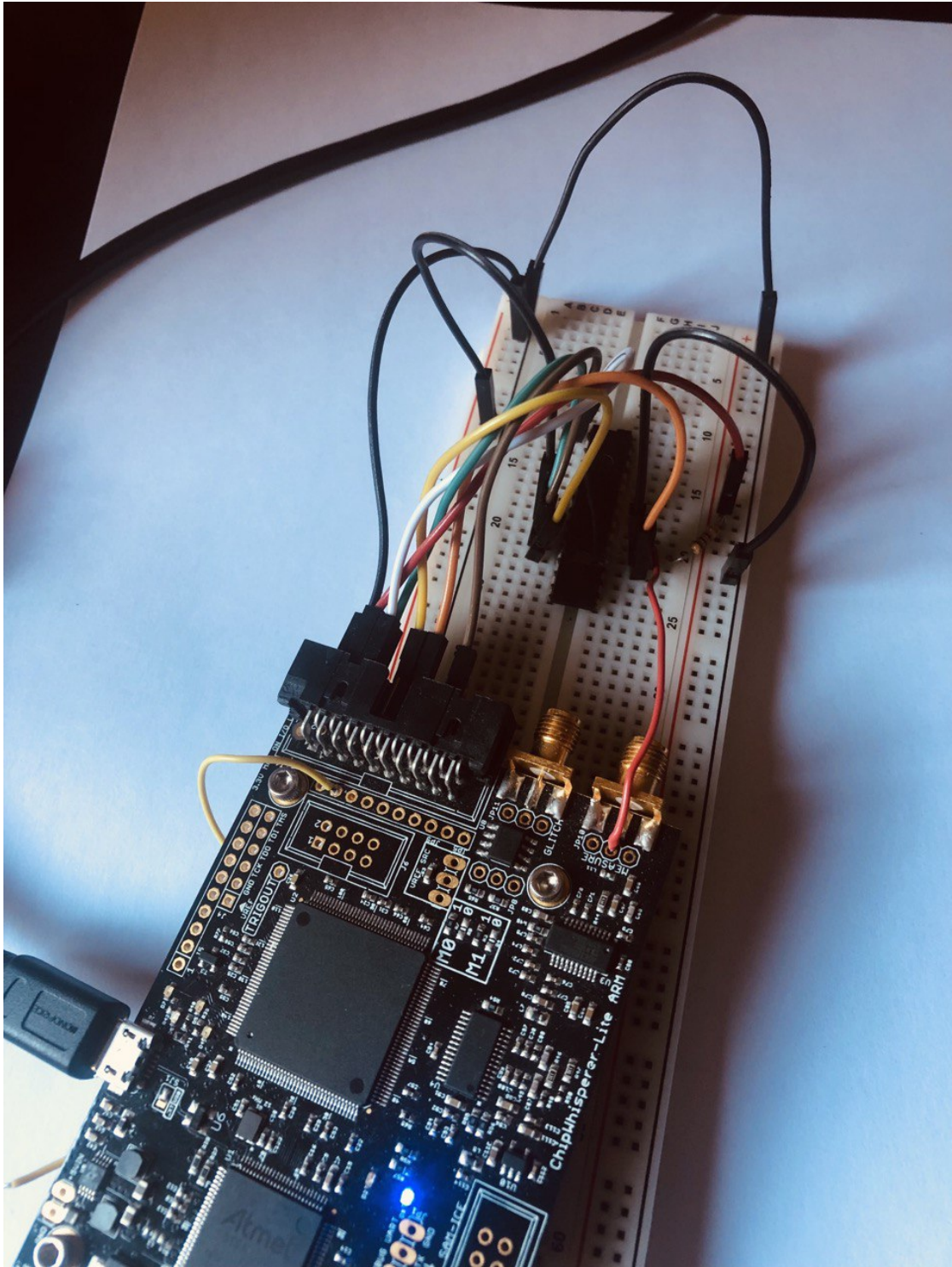


Figure 5.9: Interconnections between the *ChipWhisperer*[™] and PIC18F27K42.

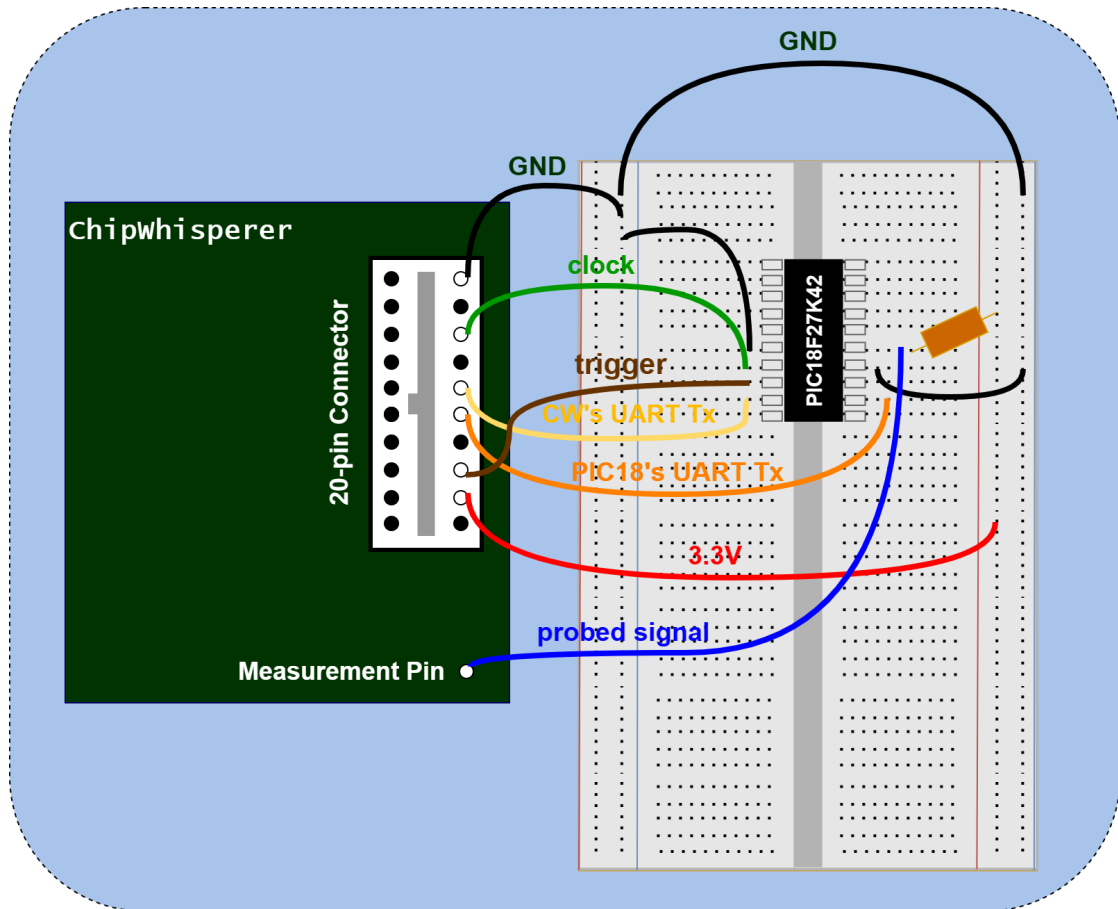


Figure 5.10: Sketch showing the important interconnections.

The following interconnections can be found:

- The green line, connected to the HS2 pin of the *ChipWhisperer*[™], carries the square wave of the clock signal, which is fed to the microcontroller;
- The brown line connects the TARG4 pin of the *ChipWhisperer*[™] to the GPIO pin of the microcontroller selected as trigger for the start of the measurements;
- The yellow line connects the UART_Tx pin of the microcontroller to the TARG1 pin (i.e., the UART_Rx) of the *ChipWhisperer*[™];
- The orange line connects the TARG2 pin (i.e., the UART_Tx) of the *ChipWhisperer*[™] to the UART_Rx pin of the microcontroller;
- The black lines connect the pin at constant low voltage (0V) of the *ChipWhisperer*[™] to both the ground pins of the microcontroller, on the left and on the right;
- The red line connects the pin at constant high voltage (3,3V) of the *ChipWhisperer*[™] to the voltage supply pin of PIC18 through a resistor of 47Ω. The resistor provides a way to measure the voltage drop across the microcontroller rather than the absorbed current, as pointed out in Section 3.3;
- The blue line (red in the previous photo) connects the voltage supply pin of the microcontroller to the measurement pin of the *ChipWhisperer*[™], which corresponds

to the input of its internal ADC⁴.

5.4 Analysis on Traces

Once the connections are set, the trace collection phase can start thanks to the Python functions defined in the *ChipWhisperer*TM Capture API [30].

Most of the forthcoming evaluations in this Chapter derive from the analysis of traces collected on PIC18F27K42, although relevant mentions to the other devices are pointed out as well.

5.4.1 Simple Power Analysis

As pointed out in Section 3.3.1, the preliminary analysis often consists of a visual inspection. The collection of a power trace from PIC18F27K42 by means of the *ChipWhisperer*TM provided the wave depicted in Figure 5.11.

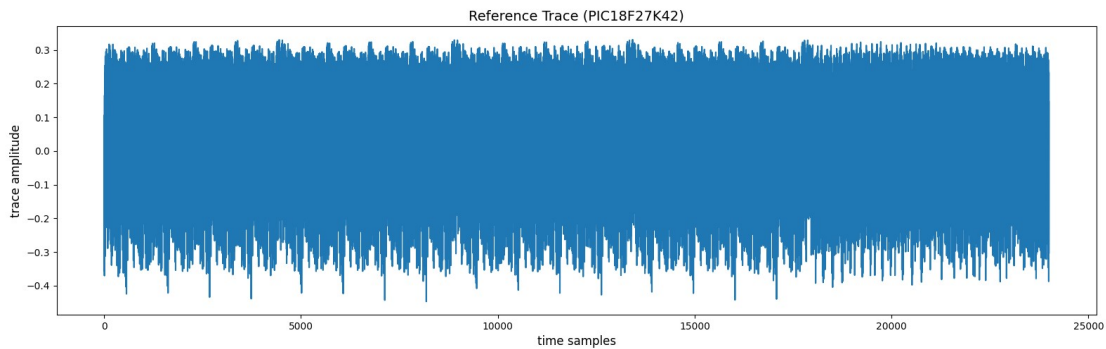


Figure 5.11: AES-128 execution on PIC18F27K42.

Unfortunately the visual inspection proved to be harder than expected. The trace does not clearly show the peculiarities of AES that are evident on other devices (e.g., power spikes in Figure 3.13 that allow to identify the different operations in AES-128). The Simple Power Analysis run on the trace distinguishes the portions in Figure 5.12.

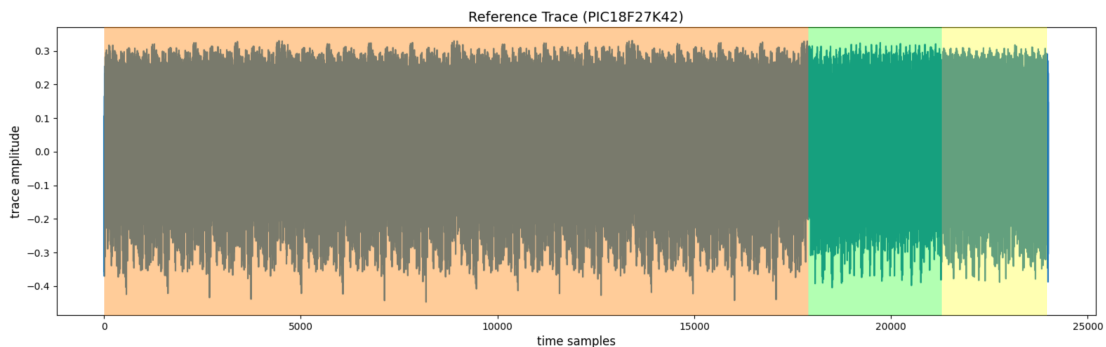


Figure 5.12: Annotations on PIC18F27K42 power trace.

⁴In electronics, an Analog to Digital Converter (ADC) is a component that performs the transformation of a continuous signal into a discrete one.

In the orange part of the picture (from sample 0 to 18,000) it is possible to distinguish 16 negative spikes that likely refer to an operation of the encryption algorithm (being 16 the number of bytes of both the secret key and the plaintext). Two more zones can be distinguished from sample 18,000, the green and yellow ones, although it is hard to tell which operations they refer to.

5.4.2 Where is AES?

The observable leakage does not allow the attacker to easily detect the peculiar current spikes originated from the execution of the AES-128 algorithm. In order to improve the observable signal and understand if there is a problem with the setup, two operations are performed: the computation of a custom metric, called Signal Quality Index (SQI) and the Reverse Correlation Power Analysis.

SQI computation

The Signal Quality Index is a custom-developed metric used to evaluate the strength of a desired signal when different hardware setups are employed. The final goal is to obtain a quantitative result indicating the setup able to provide power traces with higher quality, allowing to maximize the exploitable leakage.

Since the series of targeted microcontrollers features two ground pins, there are three possible scenarios to explore: only the left ground connected, only the right ground connected or both. As connecting different ground pins may activate (or not) unwanted sections of the internal microcontroller circuitry, testing which combination leads to less noisy power traces can greatly facilitate the subsequent power analysis steps performed by the neural networks. The deployed SQI function is displayed in Figure 5.13. For each scenario four signals, a correct and three wrong ones, were employed to study the behavior of the custom metric⁵.

```
def SQI(correct, wrong1, wrong2, wrong3):  
  
    wrong_sum = wrong1 + wrong2 + wrong3  
    wrong_avg = wrong_sum / 3  
  
    trace_diff = np.subtract(correct, wrong_avg)  
  
    sum = np.sum(np.abs(trace_diff))  
  
    return sum
```

Figure 5.13: Custom SQI function.

For each scenario, a summary trace stemming from three wrong signals is subtracted from the correct trace. The SQI is equal to the sum of the absolute values of the difference. The higher the value, the more clearly the correct trace distinguishes itself from the others. In other words, the best experimental setup tested will be the one returning the highest value of the SQI metric. The three scenarios are depicted in Figure 5.14.

⁵The four signals derive from the `Basic_Password_Check` program by *NewAE*, which provides a signal reputed correct - the one related to the correct character in the password to be checked - and an arbitrary number of traces considered incorrect, all produced by sending incorrect password characters

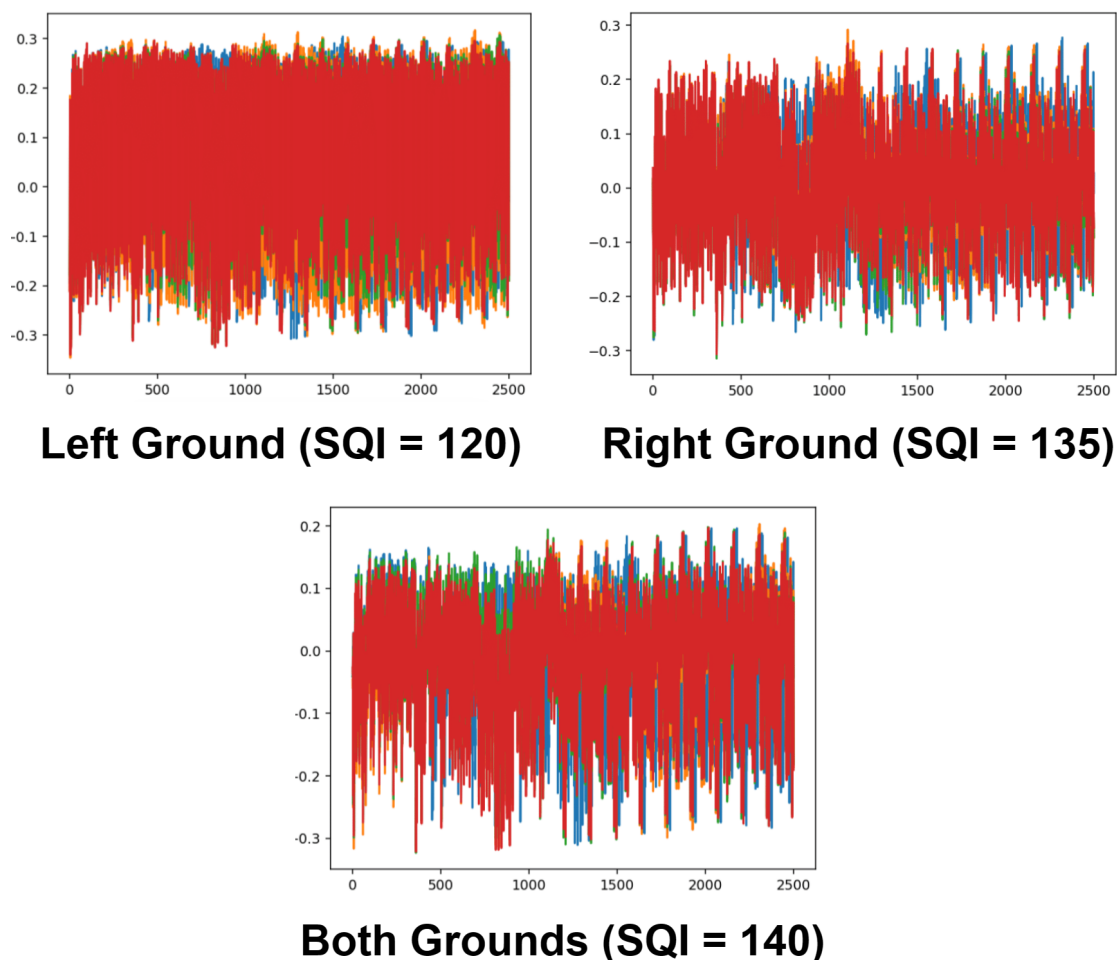


Figure 5.14: The three scenarios for SQI computation.

A visual observation of the waves used to compute the SQI confirms the values obtained. In the first plot it is hard to distinguish the correct trace, in blue, from the three incorrect ones (in red, orange and green). The situation is improved in the second plot, where some blue zones are clearly more visible. However, the visual inspection on the third plot demonstrates that the configuration with both ground pins connected achieves the highest SQI, slightly better with respect to the second one.

The result of the analysis does not represent a surprise, as connecting both pins is suggested by the datasheet of the microcontroller since the left one is mainly used by the peripherals and the right one by the core [86]. This means that the initial chosen setting, in Figure 5.10, is the one characterized with the highest SQI. The outcome of this analysis confirms that the traces previously inspected with Simple Power Analysis are meaningful, even though visually inferring the operations taking place in the AES algorithm proved to be harder than expected.

Reverse CPA

An additional check that can assure the meaningfulness of the traces is the *Reverse Correlation Power Analysis* introduced in Section 3.3.3. By means of this technique, the portions of the power trace that show high correlation with the key or plaintext are highlighted. Thanks to the framework developed by *eShard* it is possible to lead Reverse

CPA with respect to both the plaintext and the key. The Reverse CPA with respect to the plaintext bytes 0, 1, 2, 3, 4 is displayed in Figure 5.15.

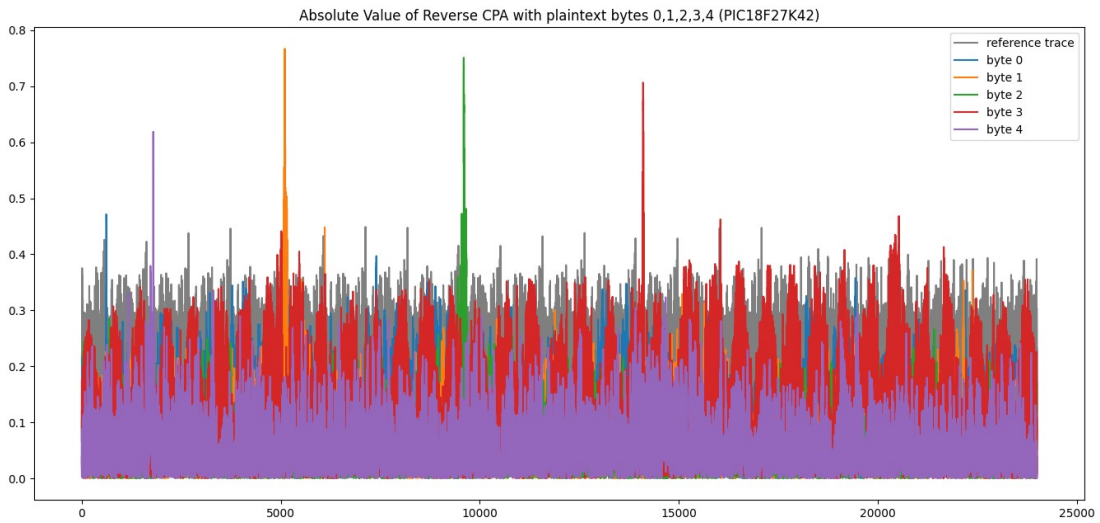


Figure 5.15: Reverse CPA with plaintext bytes 0,1,2,3,4 on PIC18F27K42.

The plot shows that there exists correlation between the power trace and the plaintext bytes, thus providing a proof that the waves are meaningful. Nevertheless, a peculiar behavior is observable: the correlation spikes do not follow the order of the bytes (i.e., 0, 1, 2, 3, 4), but rather 0, 4, 1, 2, 3.

The Reverse CPA with respect to the key bytes 0, 1, 2, 3, 4 is displayed in Figure 5.16.

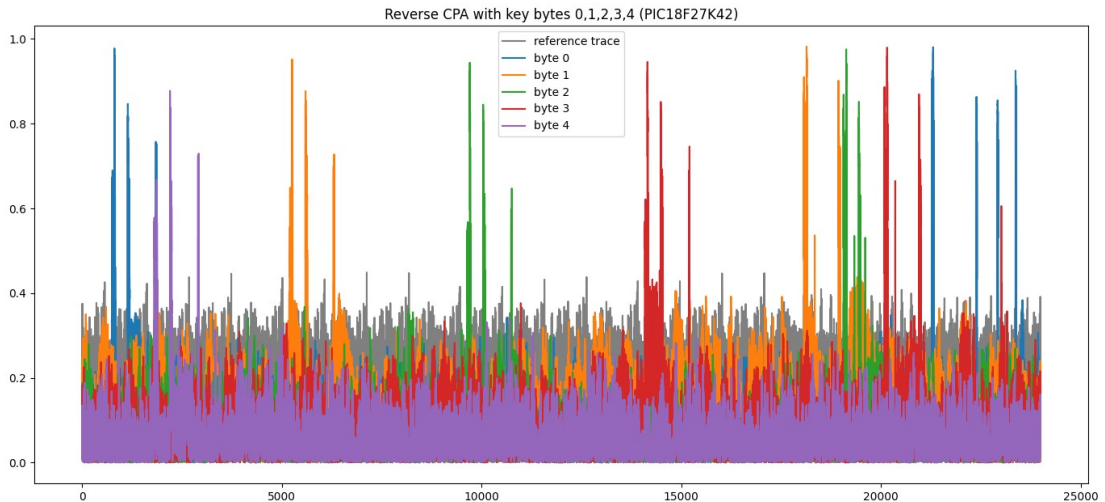


Figure 5.16: Reverse CPA with key bytes 0,1,2,3,4 on PIC18F27K42.

Here, just like in Figure 5.15, the ascending byte order is not respected. However, the drawing of the State matrix in which the key and plaintext bytes are arranged, in Figure 5.17, makes a possible order pattern appear.

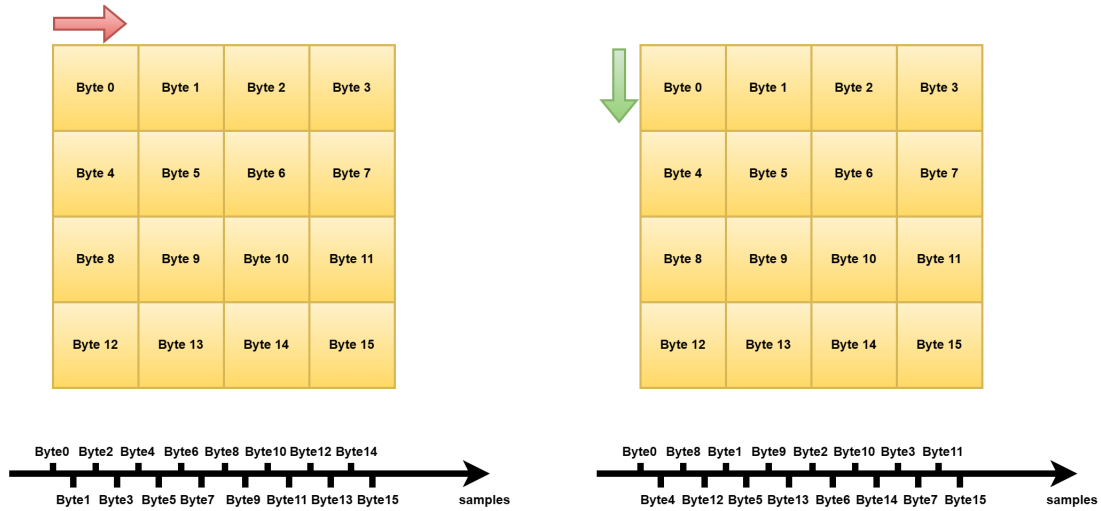


Figure 5.17: Expected and actual order in the handling of bytes.

The hypothesis is that the matrix is not handled by rows, but rather by columns. A new analysis is then run on the collected traces. Now instead of tackling bytes in the row-major order, the column-major sequence is taken into account (i.e., 0, 4, 8, 12 and so forth). Figure 5.18 illustrates that the spikes between samples 0 and 6,000 follow the new expected order.

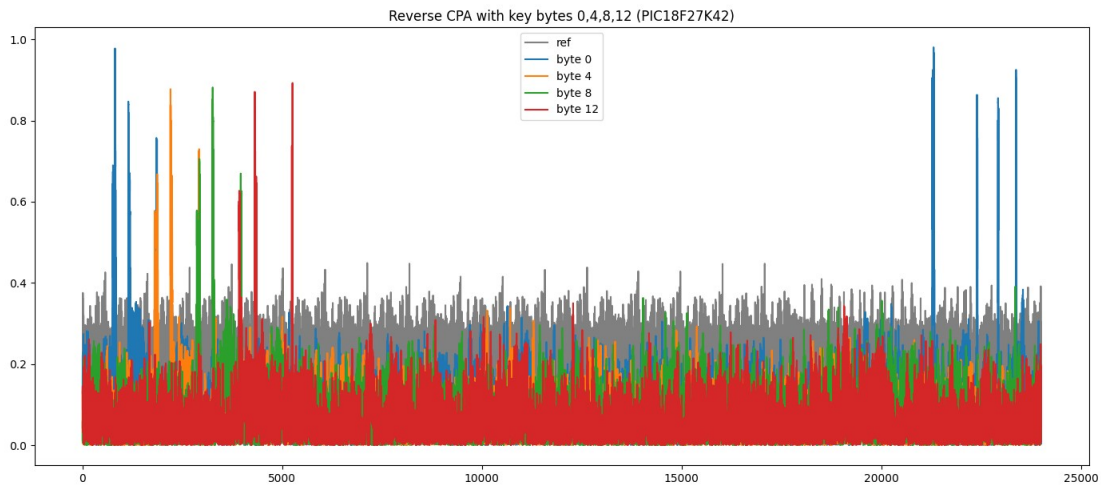


Figure 5.18: Reverse CPA with key bytes 0,4,8,12 on PIC18F27K42.

The behavior is now finally captured. The last visual inspection performed on PIC18F27K42 is in Figure 5.19.

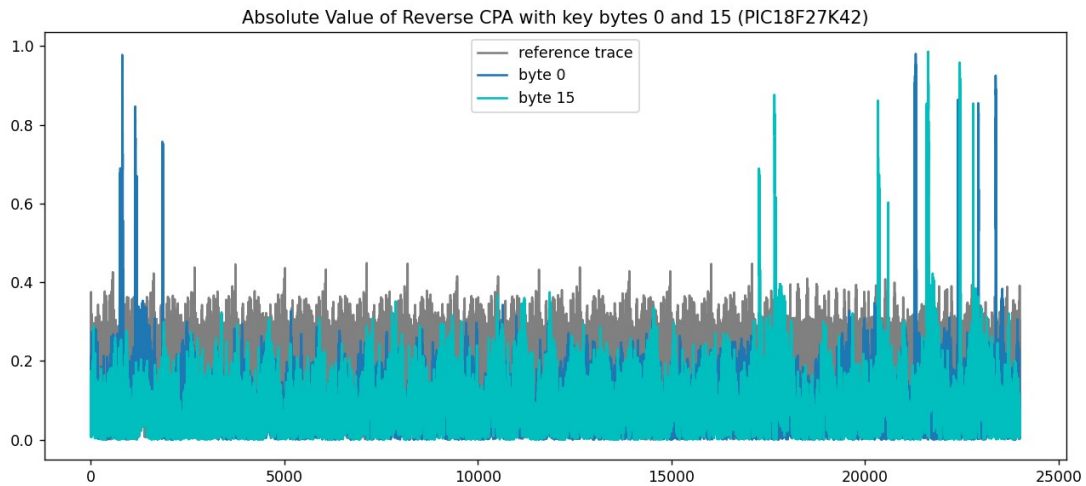


Figure 5.19: Reverse CPA with key bytes 0 and 15 on PIC18F27K42.

The plot shows that two distinct operations take place: the first one, from sample 0 to 18,000, and the second one right after. The second operation is not fully caught, as the *ChipWhisperer*TM has a limitation on the number of collectable samples in a single trace⁶.

The handling of all sixteen bytes during the first portion of the trace suggests that the wave between samples 0 and 18,000 corresponds to the the *SubBytes* operation. It is fully collected, since even the spikes regarding the last byte in the matrix scanned by columns (i.e., byte 15) are present.

Similar analysis were performed for other devices by *Microchip*, on PIC18F47K42, PIC18LF45K42, PIC18F26K42 and PIC18F46K20. Figure 5.20 depicts the correlation spikes for PIC18LF45K42 and PIC18F46K20.

⁶The *ChipWhisperer*TM also features decimation options, which allow to discard some samples and enlarge the collectable trace. However, the decimation might cause an undersampling of the trace, with consequent loss of spikes in the current consumption.

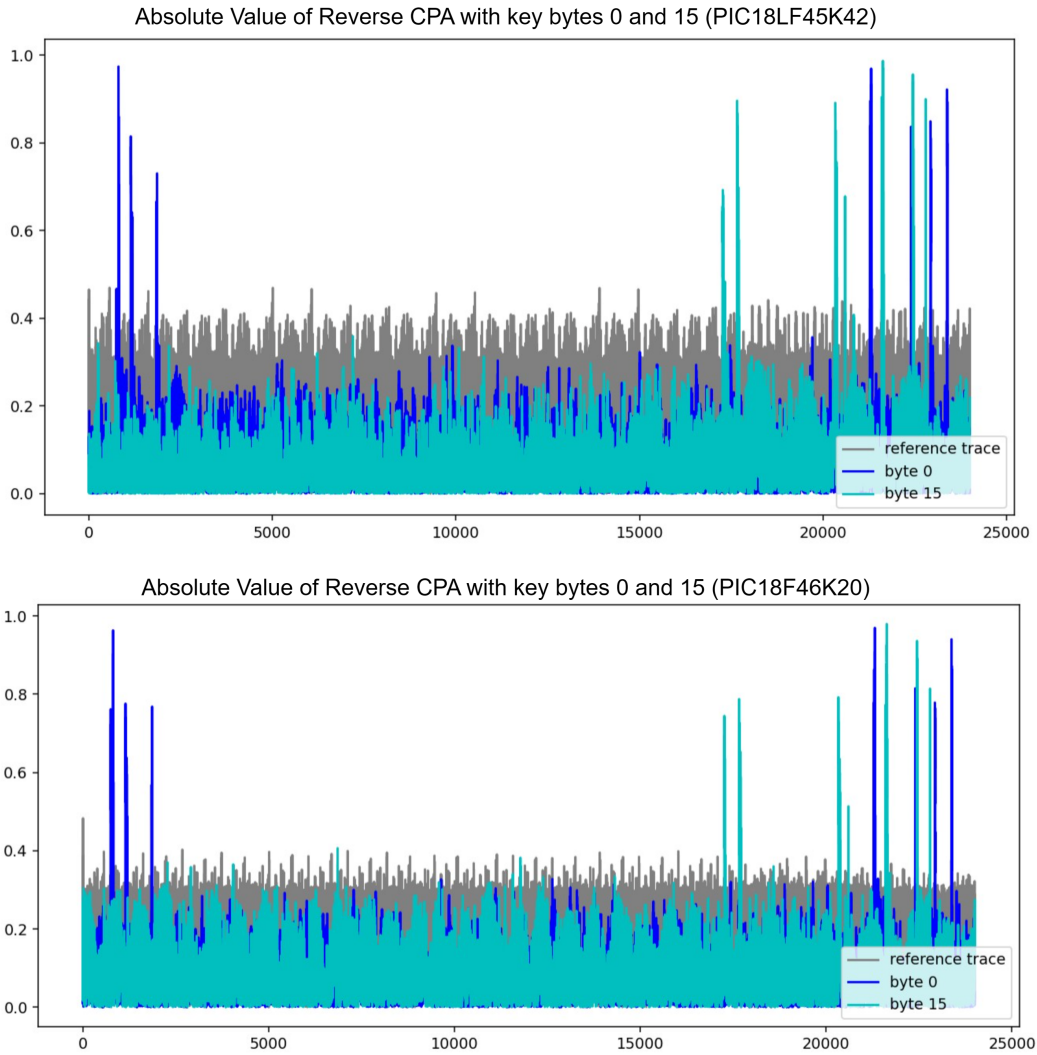


Figure 5.20: Reverse CPA with key bytes 0 and 15 on PIC18LF45K42 and PIC18F46K20.

The results of Reverse CPA on these two microcontrollers highlight similarities with the first device taken into account, as their first and last correlation peaks are located at the same instants in time with respect to PIC18F27K42, around samples 0 and 18,000.

5.4.3 Correlation Power Analysis

Once the SQI computation confirmed that the setting is optimal and the Reverse CPA demonstrated that the traces are sufficiently clean, the first real attack can be performed in order to retrieve the key of the encryption. For doing so, multiple rounds of Correlation Power Analysis are launched, with different number of traces. The results are in Table 5.1.

	Number of Traces			
	20	30	40	50
Round 1	10/16	15/16	16/16	16/16
Round 2	11/16	14/16	16/16	16/16
Round 3	12/16	15/16	16/16	16/16
Round 4	13/16	14/16	16/16	16/16
Round 5	12/16	14/16	16/16	16/16
Success Rate on 5 Rounds	72.5%	90%	100%	100%

Table 5.1: Key bytes recovered with CPA on PIC18F27K42 in dependence to number of traces.

CPA manages to recover all key bytes on PIC18F27K42 100% of the times during the 5 rounds when at least 40 traces are taken into account. Below 40, at least one byte is not recovered. This analysis is useful to set a threshold of minimum number of traces for a successful attack and compare it with Deep Learning Side-Channel Analysis techniques, in the next Chapter.

5.5 Custom Datasets

Although Deep Learning promises to eliminate the manual feature extraction needed by Machine Learning algorithms, some preliminary operations on traces are necessary.

The first one consists in performing a coarse grain cut of power traces in order to provide lighter waves as input data to the neural networks, since the physical resources employed during the thesis would require a high time for processing the raw data. The resulting traces contain samples between 0 and 18,000, as pointed out by the Reverse CPA in the previous Section.

The second operation is the normalization between 0 and 1 of the power traces' amplitude (i.e., the values in the vertical axis), as suggested by Elie Bursztein during the demonstration of a Deep Learning Side-Channel Attack at *DefCon27* [62].

The key employed during the collection of traces is in Table 5.2. The same key is used for both datasets since the work only focuses on the neural networks' capability of attacking devices from different families with respect to the microcontrollers used in the profilation phase. Multiple keys would introduce complex patterns whose effects are out of the scope of this thesis.

Byte Number	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Byte Value	0x2B	0x7E	0x15	0x16	0x28	0xAE	0xD2	0xA6	0xAB	0xF7	0x15	0x88	0x09	0xCF	0x4F	0x3C

Table 5.2: Key used in both datasets.

The acquisition campaign produced two datasets called K42_K20 and K42_K42. Their structure is inspired by ASCAD-F, as they both feature 50,000 traces for profilation and 10,000 for attack evaluation. In both, the profiling traces are constituted by waves from PIC18F27K42, PIC18F47K42 and PIC18LF45K42 (each contributing for one third of the 50,000 traces), devices all belonging to the PIC18 K42 family. The difference in the two datasets stems from the attack traces, as in K42_K20 they are collected from PIC18F46K20 and in K42_K42 from PIC18F26K42.

The two datasets are used as different benchmarks for the neural networks. K42_K42 represents the easiest setup, as proflilation and evaluation take place on similar devices, and it constitutes the baseline for the various analysis: if an attack fails in K42_K42 it should fail in K42_K20 as well, due to the presence of heterogeneous attack and profile traces, which constitute a harder problem for the neural network to generalize.

5.6 Deep Learning Models

In recent years, several solutions were developed to easily build the most popular neural networks described in Section 4.2.2. **Keras** [87] is an open-source Python library built on top of *Google's* TensorFlow [88]. It allows users to define the structure of Deep Learning models through few lines of code, thanks to a straightforward API. The definitions of an MLP and a CNN are respectively in Figures 5.21 and 5.22.

```

4 def simple_MLP():
5
6     # Create a Sequential model
7     model = Sequential()
8
9     # Add a dense layer with 64 units and input shape of (input_dim,)
10    model.add(Dense(64, input_dim=10, activation='relu'))
11
12    # Add a final dense layer with 3 units and softmax activation function
13    model.add(Dense(3, activation='softmax'))
14
15    # Compile the model
16    model.compile(loss='categorical_crossentropy', optimizer='RMSprop', metrics=['accuracy'])

```

Figure 5.21: Definition of a simple MLP in Keras.

The MLP consists of an input layer of 10 nodes, a single hidden layer with 64 perceptrons and an output layer with 3 classes. The activation functions in the hidden and output layers are ReLU and SoftMax, respectively. The Cross Entropy cost function is employed in combination with RMSprop to optimize the accuracy metric.

```

4 def simple_CNN():
5
6     # Define the model as a sequence of layers
7     model = Sequential()
8
9     # Add the first convolutional layer with 32 filters, a 3x3 kernel, and ReLU activation function
10    model.add(Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28, 28, 1)))
11
12    # Add a max pooling layer to reduce the spatial dimensions
13    model.add(MaxPooling2D(pool_size=(2, 2)))
14
15    # Add another convolutional layer with 64 filters, a 3x3 kernel, and ReLU activation function
16    model.add(Conv2D(64, kernel_size=(3, 3), activation='relu'))
17
18    # Add another max pooling layer
19    model.add(MaxPooling2D(pool_size=(2, 2)))
20
21    # Flatten the output of the convolutional layers
22    model.add(Flatten())
23
24    # Add a fully connected layer with 128 units and ReLU activation function
25    model.add(Dense(128, activation='relu'))
26
27    # Add an output layer with 10 units (one for each class) and softmax activation function
28    model.add(Dense(10, activation='softmax'))
29
30    # Compile the model with categorical cross-entropy loss and Adam optimizer
31    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

```

Figure 5.22: Definition of a simple CNN in Keras.

The model works on input data arranged as a matrix of 28x28 pixels. It features two convolutional layers, each with max pooling, and a fully connected layer. Both these stages use ReLU as the activation function. The output layer consists of 10 nodes and uses SoftMax to provide the probability density function among the different output classes. The Cross Entropy cost function is employed in combination with Adam to optimize the accuracy metric.

5.6.1 Neural Networks under Assessment

As pointed out in Section 5.1, three levels of complexity are employed for building the models used in the work, stemming from the different degrees of commitment by the hacker. Unfortunately, the computing capabilities of the available hardware do not allow to study the behavior of Convolutional Neural Networks, as their training is much heavier than the one required by Multilayer Perceptrons.

Basic MLP

The neural network in Figure 5.23 is the product of the hypothetical attack with low degree of commitment. It represents the simplest Deep-MLP as it features only two hidden layers, minimum for a Deep Neural Network according to DLSCA researchers [22].

```
6 def basic_MLP(classes, number_of_samples):
7
8     # Create a Sequential model
9     model = Sequential(name="basic_MLP")
10
11     # Add a first dense layer with 200 units and input shape of (number_of_samples,)
12     model.add(Dense(200, activation='elu', input_shape=(number_of_samples,)))
13
14     # Add a second dense layer with 200 units
15     model.add(Dense(200, activation='elu'))
16
17     # Add a final dense layer with <num_classes> units and softmax activation function
18     model.add(Dense(classes, activation='softmax'))
19
20     # Compile the model
21     model.compile(loss='categorical_crossentropy', optimizer=Adam(lr=0.001), metrics=['accuracy'])
```

Figure 5.23: Definition of the first MLP.

The main knob is represented by the number of perceptrons per layer, that is set thanks to heuristics. It cannot be too low because the model would not have a fair degree of complexity needed by the problem. On the other hand, it cannot be too high because the attacker might not have the means (i.e., the hardware or even the time) to train a large model.

Another important choice regards the learning rate, as a low α would require much time for the convergence of the cost function whereas a high value would not let it converge at all. Also in this case heuristics is leveraged, as a quick research can provide the hacker with a reasonable magnitude for the parameter.

Obviously many more parameters could be set, such as the type of activation functions or the employed optimizer, which might cause different results. In this scenario their careful selection is not taken into account as the underlying hypothesis is that the attacker does not intend to spend time tuning the neural network.

ASCAD MLP

The second model investigated during the work, in Figure 5.24, corresponds to the neural network tuned by the researchers of the ANSSI on the ASCAD-F dataset presented in Section 4.3.3.

```
7 def ascad_MLP(classes, number_of_samples):
8
9     # Create a Sequential model
10    model = Sequential(name="ascad_MLP")
11
12    # Add a first dense layer with 200 units and input shape of (number_of_samples,)
13    model.add(Dense(200, activation='relu', input_shape=(number_of_samples,)))
14
15    # Add four more dense layers with 200 units each
16    for i in range(4):
17        model.add(Dense(200, activation='relu'))
18
19    # Add a final dense layer with <num_classes> units and softmax activation function
20    model.add(Dense(classes, activation='softmax'))
21
22    # Compile the model
23    model.compile(loss='categorical_crossentropy', optimizer=RMSprop(lr=0.00001), metrics=['accuracy'])
```

Figure 5.24: MLP tuned for ASCAD-F dataset.

Here, in contrast with the first scenario, no hyperparameter is selected by the attacker. The underlying assumption is that the hacker builds a dataset with a similar structure to the one proposed in ASCAD. As it was developed by the ANSSI, the tuning process is more accurate than the one any individual attacker could perform, due to the high amount of resources the French Cybersecurity Agency could dedicate (e.g., expensive GPUs for hyperparameters search that hobbyists cannot afford to buy). In other words, this scenario requires partial knowledge of the handled data but could theoretically outperform any self-tuned model.

GridSearch MLP

The last neural network tackled in the work is the one originated by the Grid Search on the K42_K20 dataset, which takes advantage of the information provided by the whole dataset. The intervals within the hyperparameters are searched are listed in Table 5.3. The operation took place thanks to KerasTuner, an additional library built on top of Keras [87].

	Interval
Number of Layers	[2,6] (step: +1)
Neurons per Layer	[50,800] (step: +50)
Learning Rate	[0.00001, 0.1] (step: x10)
Activation Function	[ReLU, SeLU, Sigmoid]
Optimizer	[SGD, RMSprop, Adam]

Table 5.3: Intervals used for the Grid Search.

These intervals result into 3,600 possible combinations of hyperparameters, each corresponding to a new neural network to evaluate. The best model was selected after a long search (around 48 hours), using the Guessing Entropy as metric.

Chapter 6

Results

6.1 Classic SCA and DLSCA on PIC18F27K42

The first relevant result is not strictly correlated to the objective of the thesis but highlights the strength of DLSCA techniques. It regards the number of traces needed to recover the whole key with Deep Learning algorithms, displayed in Figure 6.1. Two copies of PIC18F27K42 were used for the collection of two groups of traces, a profiling and an attack set, with 10,000 and 1,000 traces respectively. The `Basic_MLP` defined in the previous Chapter was employed.

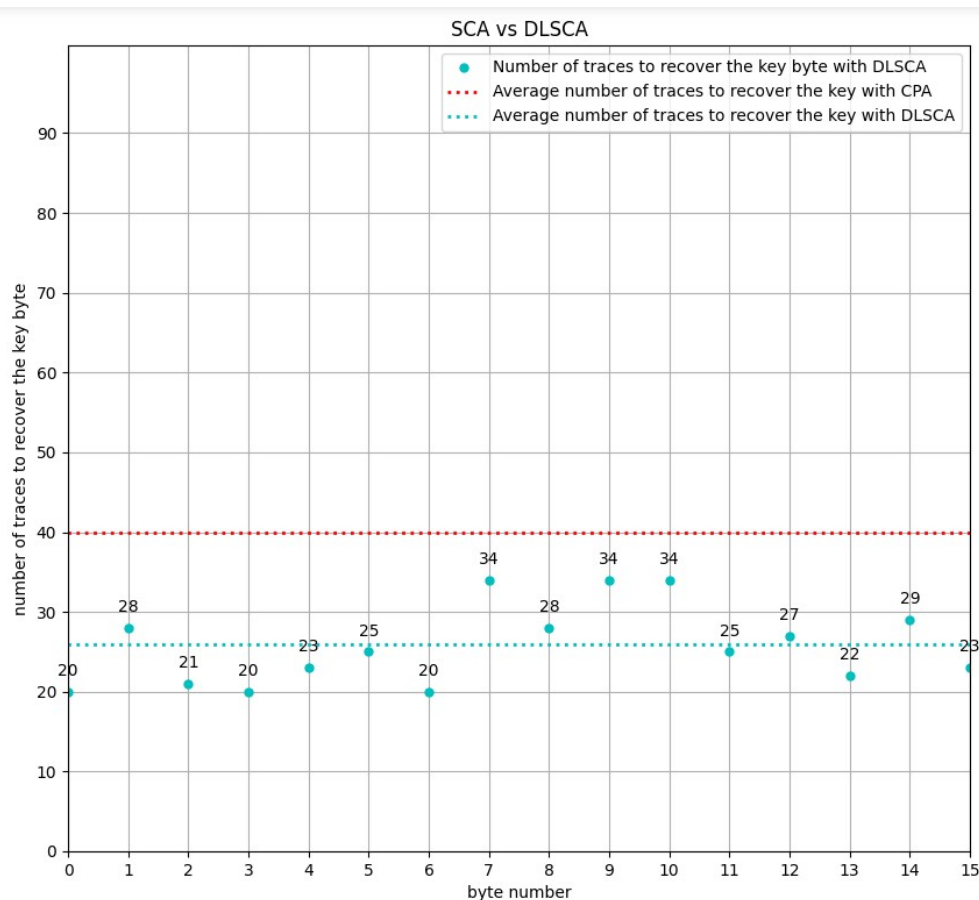


Figure 6.1: SCA and DLSCA on PIC18F27K42.

Each dot in the Figure is computed as an average of the results from 10 attack

rounds, all obtaining a Partial Success Rate of 100%¹. The red line stands for the average number of traces needed to recover the whole key with CPA and a Success Rate of 100%, as pointed out in Table 5.1. This plot demonstrates that DLSCA outperforms classic Correlation Power Analysis as the recovery of any byte needs on average a minimum number of traces of 26, while with traditional SCA it is around 40.

6.2 Cross-Family Attacks

This Section demonstrates the effect of the different MLPs presented in the previous Chapter on the K42_K20 and K42_K42 datasets. For each neural network, the recovery of all key bytes is taken into account under the conditions specified by Table 6.1.

	Min	Max	Step
Number of Epochs	5	25	x5
Batch Size	5	5000	x10

Table 6.1: Number of Epochs and Batch Size investigated in the work.

If a model is underfitting the data, then a low number of epochs could exacerbate the problem as not enough training is given. On the other hand, an overfitting neural network with many epochs could further worsen the performance.

Similar considerations hold for the batch size, as a low value could lead to both an overfit of the data and high time for the cost function to converge. On the other hand, high values could result into poor generalization.

Both parameters highly impact the performances each model can achieve. For this reason, a large number of in-between cases are considered, although only the most relevant results are displayed. In the following plots, each dot represents the Partial Guessing Entropy (PGE) averaged over 5 attack rounds.

6.2.1 Basic MLP

The configuration of the `basic_MLP` that achieves the lowest PGE for each byte is depicted in Figure 6.2. It features a batch size equal to 500 and 5 epochs.

¹A high number of rounds was employed as the minimum number of traces to crack the key is a metric subject to high variance, since an attack can be particularly lucky and recover the key byte with very few traces.

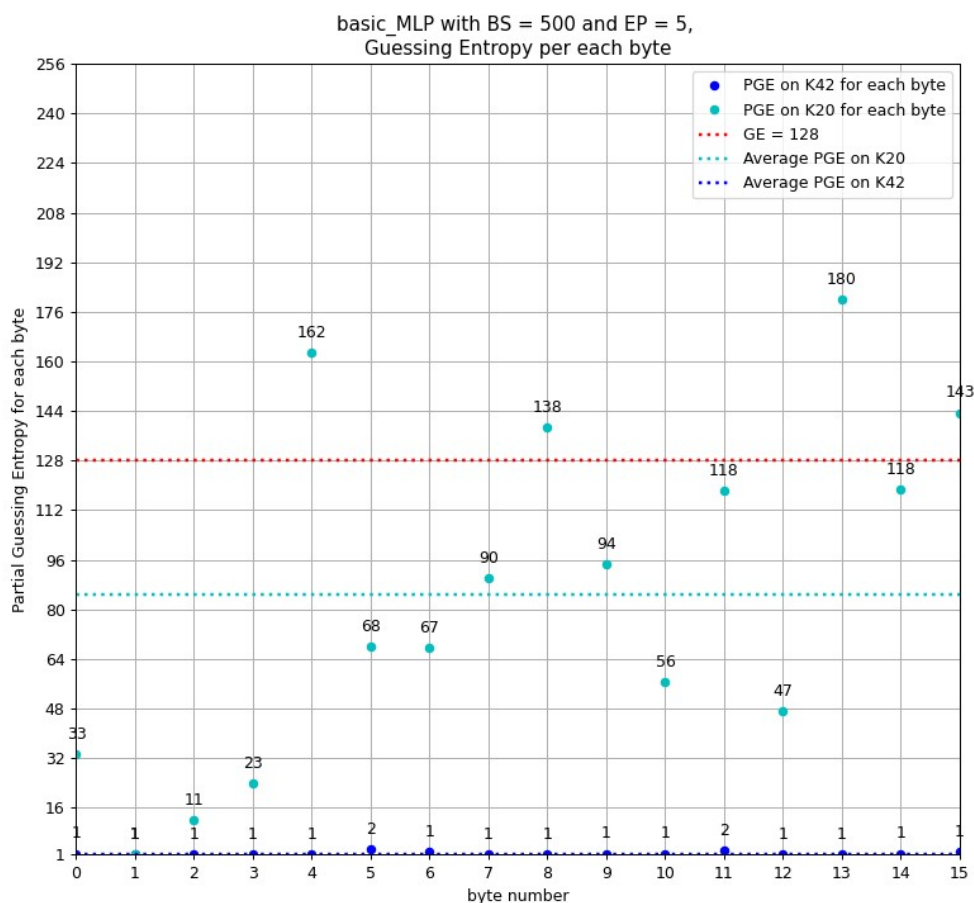


Figure 6.2: basic_MLP with BS=500 and EP=5 on K42_K42 and K42_K20.

The dashed line in red stands for the threshold below which the PGE is better than random guessing, since the possible byte values are 256 and without prior knowledge the attacker could predict any value between 0x00 and 0xFF.

The blue dots represent the results obtained with training and attack on the K42_K42 dataset. The plot demonstrates the similarities between the devices used for proflitation and the microcontroller under attack, as the campaign is nearly always successful with unitary PGE, although it is slightly higher for bytes '5' and '11'.

The cyan dots are obtained on the K42_K20 dataset. Their values demonstrate that the cross-family inference is more complex than the one on devices belonging to the same family. The average of the sixteen PGEs, represented by the dashed cyan line, is a equal to 84.72. This means that although recovering different key bytes could be simpler or harder, on average it is more convenient than random guessing.

The outcomes of the other combinations of parameters are listed in Table 6.2.

		Batch Size			
		5	50	500	5000
Epochs	5	133.8	149.6	84.7	124.5
	25	140.2	126.7	135.0	123.2

Table 6.2: Average PGE on 16 bytes for basic_MLP on K42_K20.

In red there are the configurations that result into an average of PGEs higher than

128 and in yellow the ones very close to random guessing. As expected, a neural network without prior knowledge of the problem needs a fine tuning of the available hyperparameters as only one combination, in green, achieves significant benefits.

The plot in Figure 6.3 displays the minimum number of power traces needed to reach the unitary PGE on each byte. This metric is only valid when the Guessing Entropy is equal to 1, thus it is only computed for the K42_K42 dataset. Also here, each dot represents the average on 5 different attack rounds.

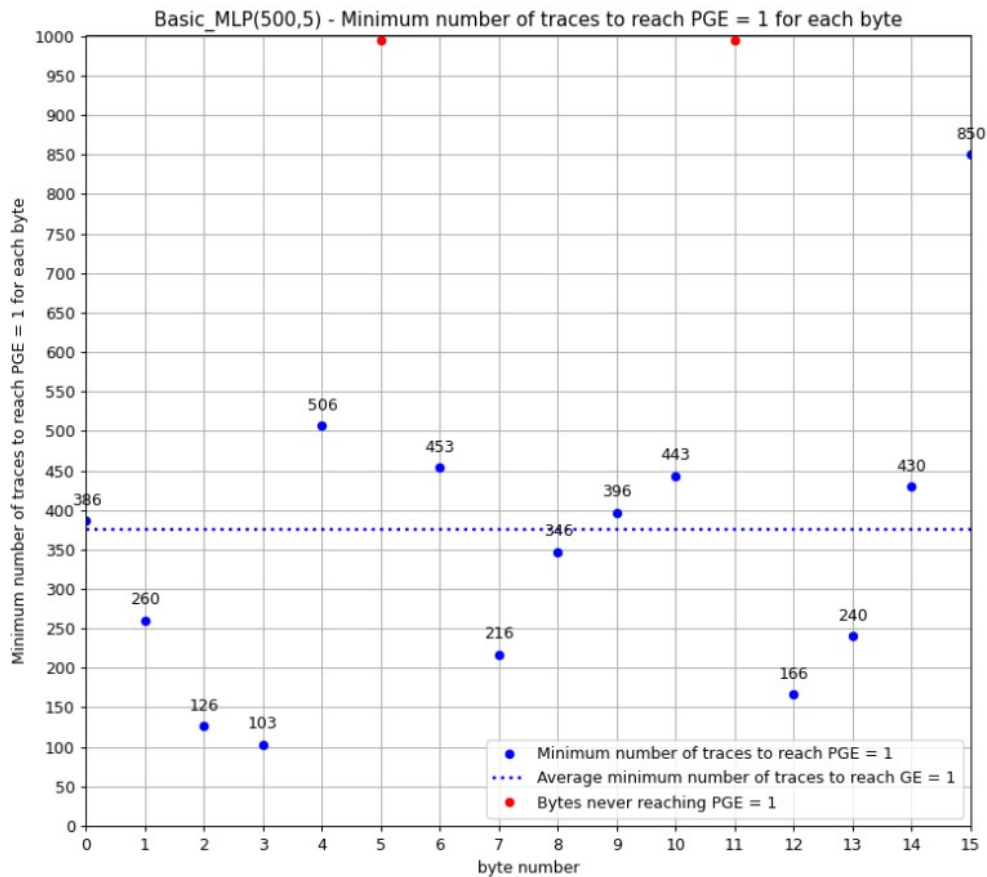


Figure 6.3: Minimum number of traces to reach $PGE = 1$ with `basic_MLP(500,5)` on K42_K42.

The dots show a high variance, and the computation of the metric is skipped for bytes '5' and '11' since their PGE equals 2. On average, the minimum number of traces to achieve $PGE = 1$ across the remaining 14 bytes is around 380.

6.2.2 Ascad MLP

Also in the case of the second MLP the plot illustrated in Figure 6.4 demonstrates the higher complexity of the cross-family inference.

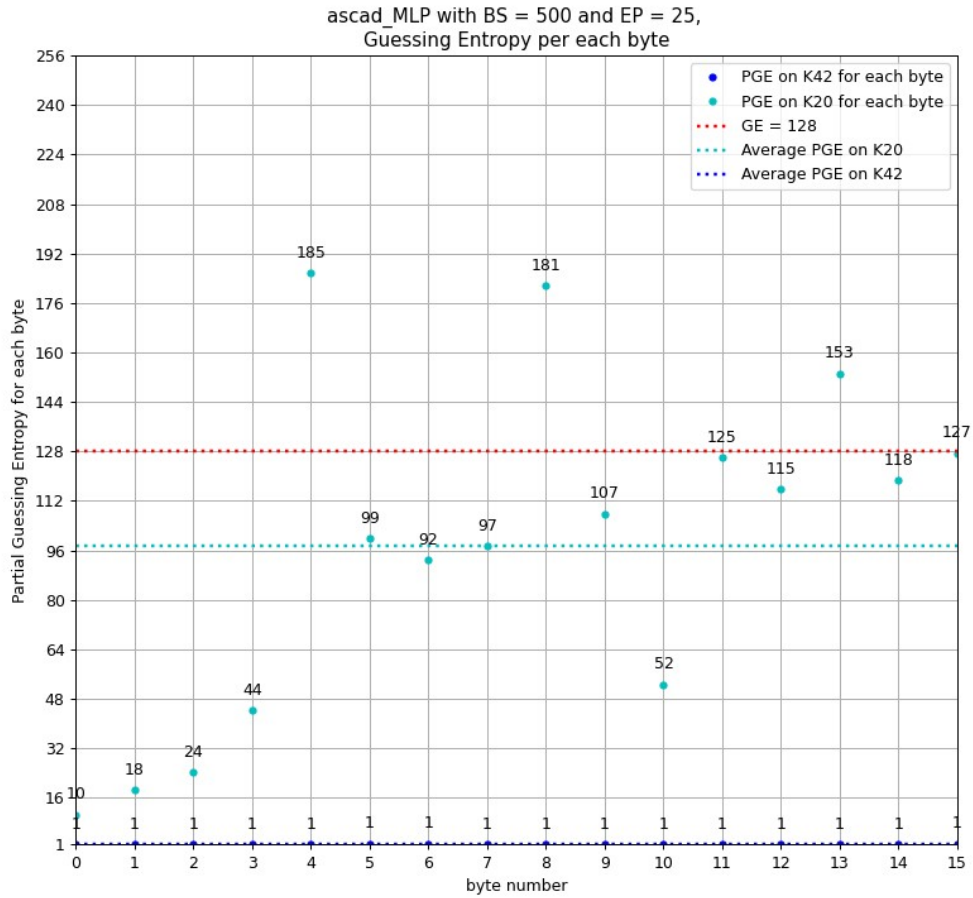


Figure 6.4: `ascad_MLP` with `BS=500` and `EP=25` on `K42_K42` and `K42_K20`.

Surprisingly `ascad_MLP` does not perform better than `basic_MLP`, as on average bytes achieve a higher PGE for the cross-family inference (around 97, as the dashed cyan line suggests). Nevertheless, on average it is still more advantageous than random guessing. Furthermore, the attack results on the `K42_K42` dataset are slightly improved, as the correct key byte is always classified as first in the ranking of predicted keys.

The outcomes of the neural network with all combinations of batch size and epochs on `K42_K20` are in Table 6.3.

		Batch Size			
		5	50	500	5000
Epochs	5	136.8	102.8	127.4	124.9
	25	146.2	147.0	97.3	127.9

Table 6.3: Average PGE on 16 bytes for `ascad_MLP` on `K42_K20`.

Here, although achieving a worse average PGE, there are two configurations that gain considerable advantage, highlighted in green. Just like with the first model, a high batch size originates results close to random guessing. Additionally, a very low batch size leads to very high Guessing Entropy, with peaks of 198 when 25 epochs are taken into account, probably stemming from the overfit of the training data.

For the `K42_K42` dataset the minimum number of traces for unitary Guessing Entropy, depicted in Figure 6.5, shows less variance although achieving a similar average value to

the first MLP.

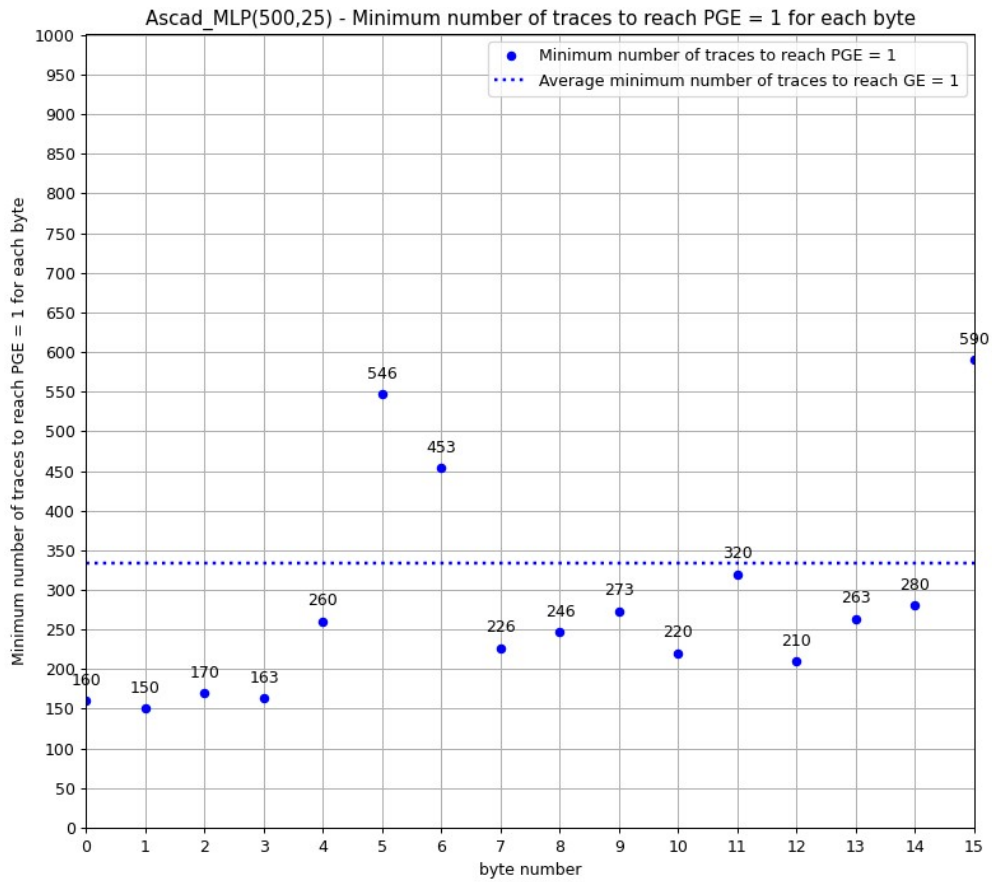


Figure 6.5: Minimum number of traces to reach $PGE = 1$ with `ascad_MLP(500,25)` on `K42_K42`.

6.2.3 GridSearch MLP

The last model, whose PGE plot is in Figure 6.6, shows the best result. This confirms the expectations, as the model was finely tuned to address the problem.

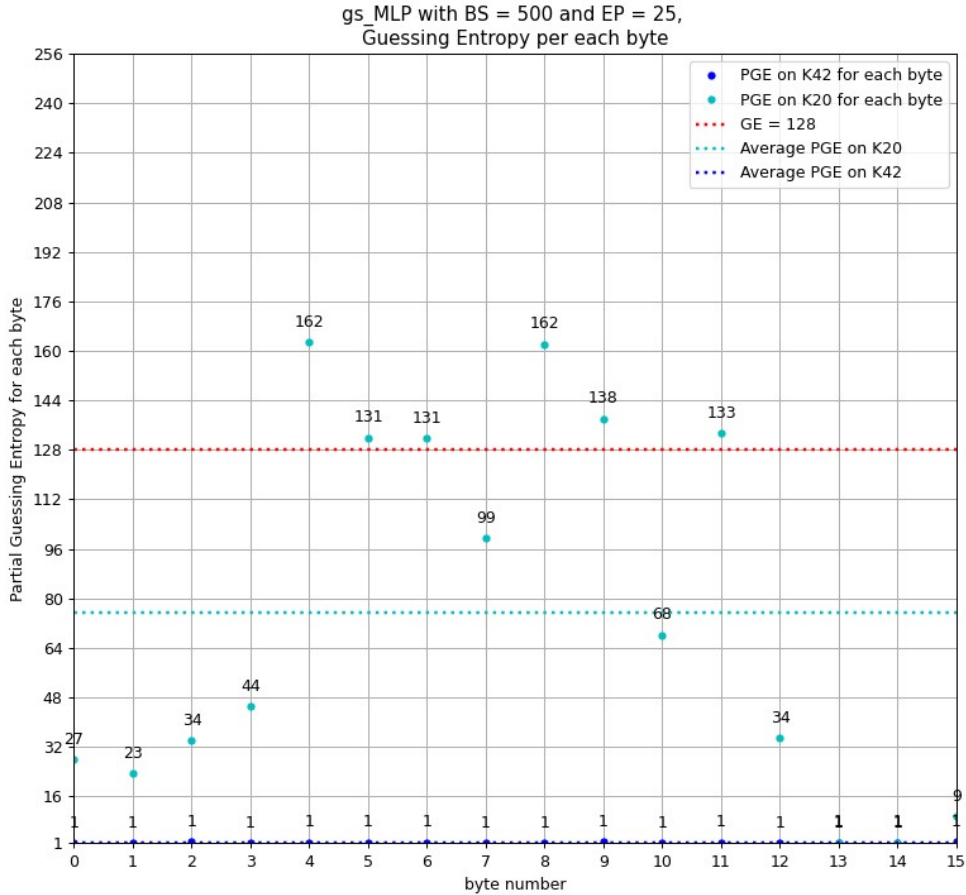


Figure 6.6: `gridsearch_MLP` with $BS=500$ and $EP=25$ on `K42_K42` and `K42_K20`.

Nearly all dots are below or close to the threshold of random guessing. The model seems to be particularly adequate for catching the behavior of the first and last bytes. Interestingly, bytes '13' and '14' even achieve a PGE of 1.

As in the previous cases, also this model is able to generalize well on the `K42_K42` dataset, as attacks always result into the unitary Guessing Entropy.

The average Partial Guessing Entropies achieved with all the tested combinations of hyperparameters are listed in Table 6.4.

		Batch Size			
		5	50	500	5000
Epochs	5	122.6	101.1	124.4	126.9
	25	145.8	131.4	75.3	129.7

Table 6.4: Average PGE on 16 bytes for `gridsearch_MLP` on `K42_K20`.

Just like in the case of `ascad_MLP`, two settings carry considerable benefits, highlighted in green. Although three results are above the threshold of random guessing, only the training with a batch size of 5 and 25 epochs achieves a very high PGE.

Finally, the minimum number of traces to reach the unitary PGE is illustrated in Figure 6.7. This plot shows less variance with respect to the ones obtained by the other models, although its average value is around 400, slightly higher than `basic_MLP` and `ascad_MLP`.

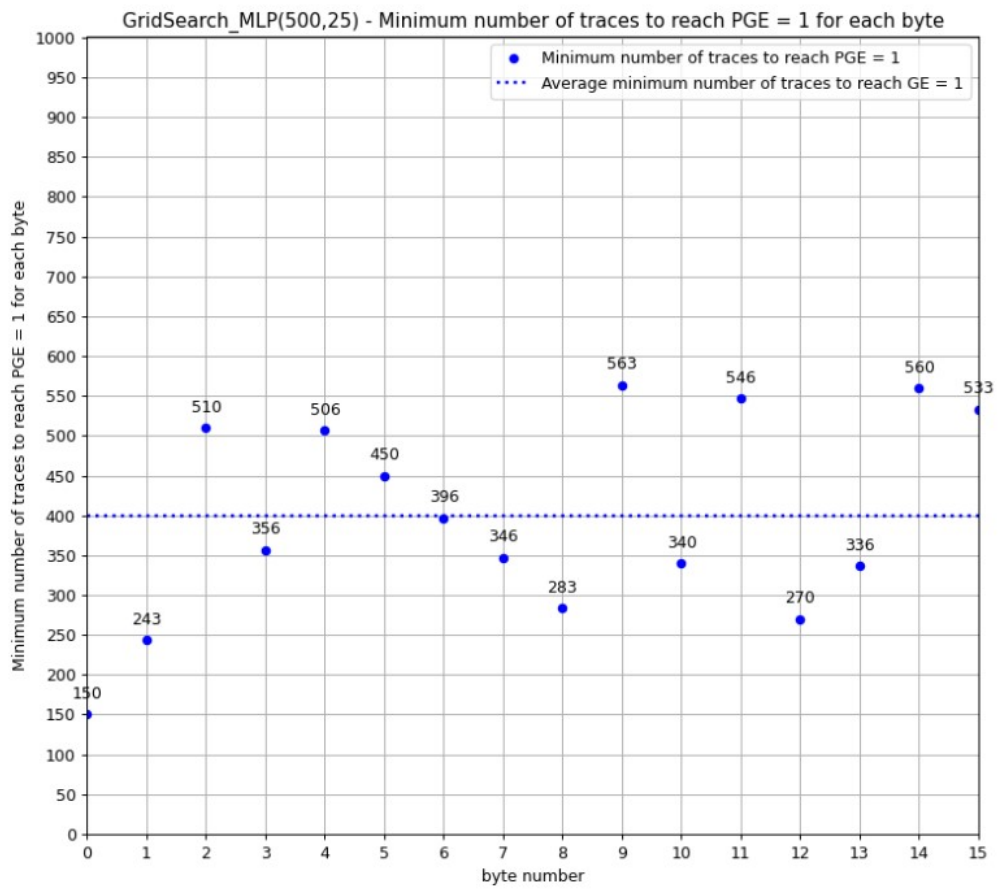


Figure 6.7: Minimum number of traces to reach $PGE = 1$ with `gridsearch_MLP(500,25)` on K42_K42.

Chapter 7

Conclusions

7.1 Results Digest and Comments

The minimum average number of traces obtained with `basic_MLP` (26 rather than 40) demonstrates that DLSCA outperforms traditional SCA methods, although requiring time for the training of the neural networks and expertise of the adjustable hyperparameters. Deep Learning constitutes a series of highly versatile tools that can be applied to numerous research fields, and the time spent to perform Side-Channel Analysis can be an acceptable trade off as models are able to extract features automatically when multiple attack scenarios are taken into account.

A summary plot displaying the best results achieved by the different Multilayer Perceptrons in the previous Chapter is in Figure 7.1.

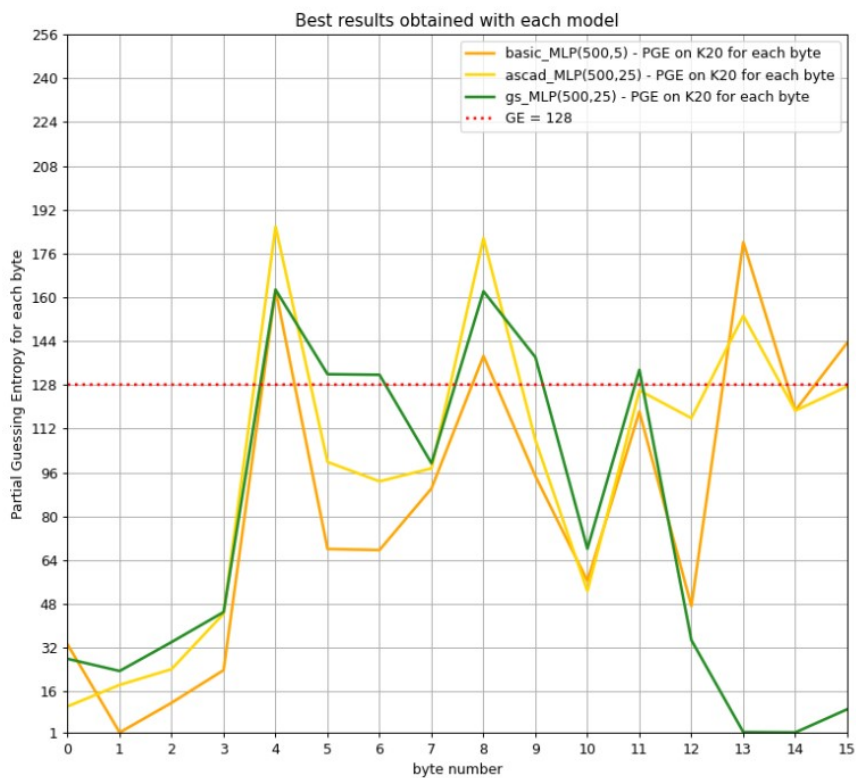


Figure 7.1: PGE achieved by all the best configurations of MLPs.

The graph demonstrates that all models reach a low PGE when guessing the first 4

bytes. Both `basic_MLP` and `ascad_MLP` achieve similar results between bytes '4' and '11', although the former is always below the latter in the interval. A significant difference can be noticed for bytes between '12' and '15', where `gridsearch_MLP` clearly achieves the best results.

As expected, a finely tuned neural network manages to highly reduce the search space of the single key bytes, although all models achieve improvements with respect to random guessing, as pointed out in the previous Chapter. This consideration highlights the strength of Deep Learning Side-Channel Analysis techniques, which can be leveraged even without prior knowledge of the device under attack.

The Table 7.1 recapitulates the average PGEs obtained in dependence to the investigated Batch Sizes and Epochs.

		Batch Size				
		5	50	500	5000	
Epochs	5	133.8	149.6	84.7	124.5	Basic_MLP
	25	140.2	126.7	134.0	123.2	
	5	136.8	102.8	127.4	124.9	Ascad_MLP
	25	146.2	147.0	97.3	127.9	
	5	122.6	101.1	124.4	126.9	GridSearch_MLP
	25	145.8	131.4	75.3	129.7	

Table 7.1: Average PGE on 16 bytes for `basic_MLP`, `ascad_MLP` and `gs_MLP` on K42_K20.

Very low batch sizes nearly always worsen random guessing. This is a probable consequence of overfit, as all results with batch size equal to 5 and 25 epochs suggest. As previously mentioned, small batches update the tuneable hyperparameters very often, with consequent risk of adapting too closely to the training data and not being able to generalize to new attack data. The phenomenon is exacerbated by the high number of epochs, as multiple iterations on the same training data are performed.

On the opposite, models with high batch sizes do not manage to extract interesting features out of the training traces, as the resulting PGEs fluctuate around the threshold of random guessing.

Further considerations regard the models configured with batch sizes of 50 and 500. While `ascad_MLP` and `gridsearch_MLP` feature two configurations that allow to highly reduce the search space of the key by a significant quantities, the first model only improves random guessing with one setting. This means that the simplest neural network, although not assuming prior knowledge of the device under attack, needs a higher effort during the model tuning process.

The selection of the neural network is not straightforward, as it is subject to a trade off. The least advantageous model is probably the `ascad_MLP`, as it requires a time-costly training and does not achieve the performance obtained by the `gridsearch_MLP`. On the other hand, the first and third networks achieve significant benefits, as the former requires a fast training although needing a more accurate hyperparameter search and the latter highly decreases the key search space.

7.2 Future Works

Deep Learning Side-Channel Analysis is a research field whose relevance is rapidly increasing, as the number of studies carried out in recent years suggest. This thesis

proposes a study that faces one of the many challenges arising: the portability of the attacks. Multiple future works in this direction are possible, such as the definition of a larger dataset for the training of neural networks. As a matter of facts, a look at the whole picture in Figure 7.1 outlines a trend followed by all models, with low bytes being simpler to attack than higher bytes for most neural networks. This phenomenon might be due to a biased dataset that does not allow models to extract features from traces to infer all key bytes equally, but rather is very representative for attacking some bytes and less for the others.

Furthermore, the work employed a Virtual Machine running 10 Xeon™ E5-2650 cores by *Intel* for a total of 20 threads, 32GB of RAM and a 300GB disk [89]. An improvement to the thesis regards the usage of GPUs, as some computationally expensive operations - such as the Grid Search - would benefit from parallelism. Employing GPUs would also allow to study the complexity of cross-family attacks when countermeasures are adopted since CNNs are widely used in the literature when jittering is employed, as they are invariant to traces translation.

Finally, new studies with more devices from different families can be carried out, as well as the elaboration of a metric that could establish the degree of dissimilarity among the microcontrollers, in order to provide a wider look to the advantages and drawbacks of cross-family inference.

Bibliography

- [1] Matt Crypto. AddRoundKey operation for AES. Available at <https://it.wikipedia.org/wiki/File:AES-AddRoundKey.svg>, August 2006. Latest access in February 2023.
- [2] Matt Crypto. SubBytes operation for AES. Available at <https://it.wikipedia.org/wiki/File:AES-SubBytes.svg>, August 2006. Latest access in February 2023.
- [3] Matt Crypto. ShiftRows operation for AES. Available at <https://it.wikipedia.org/wiki/File:AES-ShiftRows.svg>, August 2006. Latest access in February 2023.
- [4] Matt Crypto. MixColumns operation for AES. Available at <https://it.wikipedia.org/wiki/File:AES-MixColumns.svg>, August 2006. Latest access in February 2023.
- [5] The Wikipedia Community. ECB mode encryption. Available at https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:ECB_encryption.svg.
- [6] The Wikipedia Community. CBC mode encryption. Available at https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CBC_encryption.svg.
- [7] The Wikipedia Community. CTR mode encryption. Available at https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#/media/File:CTR_encryption_2.svg.
- [8] Carlton Shepherd, Konstantinos Markantonakis, Nico van Heijningen, Driss Aboulkassimi, Clément Gaine, Thibaut Heckmann, and David Naccache. Physical fault injection and side-channel attacks on mobile devices: A comprehensive survey. 2021.
- [9] Asanka Sayakkara, Nhien-An Le-Khac, and Mark Scanlon. Leveraging electromagnetic side-channel analysis for the investigation of iot devices. *Digital Investigation*, 29:S94–S103, 2019.
- [10] Michael Hutter and Jörn-Marc Schmidt. The temperature side-channel and heating fault attacks. volume 8419, 11 2013.
- [11] Khamis-M. Schneegass S. Abdelrahman, Y. and F. Alt. Stay cool! understanding thermal attacks on mobile-based user authentication. 2017.
- [12] Jasper van Woudenberg and Colin O’Flynn. *The Hardware Hacking Handbook - Breaking Embedded Security with Hardware Attacks*. No Starch Press, CA, 2022. ISBN: 9781593278748.
- [13] The Wikipedia Community. Pearson correlation coefficient. Available at https://en.wikipedia.org/wiki/Pearson_correlation_coefficient.

- [14] Benjamin Timon. How to implement a side-channel attack with Jupyter-Lab. Available at https://www.youtube.com/watch?v=KATyt7Nuanw&t=532s&ab_channel=eShard.
- [15] Elie Bursztein and Jean-Michel Picod. A hacker guide to deep-learning based AES side channel attacks. Available at <https://elie.net/static/files/a-hackerguide-to-deep-learning-based-side-channel-attacks/slides.pdf>.
- [16] aprendendoaestudar.com. Biological Neuron. Available at <https://aprendendoaestudar.com.br/2019/04/>.
- [17] The Wikipedia Community. The Sigmoid Function. Available at https://en.wikipedia.org/wiki/Sigmoid_function.
- [18] Sandeep Vivek. Introductory Guide on the Activation Functions. Available at <https://www.analyticsvidhya.com/blog/2022/03/introductory-guide-on-the-activation-functions/>.
- [19] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing, Staffordshire University*, 4:5, 2005.
- [20] Farnaz Ghassemi Toudeshki. Loss functions. Available at <https://medium.com/@farnazgh73/loss-functions-54a4d1d00681>.
- [21] Universidade Federal de Santa Catarina. What model to choose? Available at <https://geam.paginas.ufsc.br/files/2020/02/train-test-select.pdf>.
- [22] Stjepan Picek. CRC Seminar Series. Available at https://www.youtube.com/watch?v=-741KfLx8Ko&t=386s&ab_channel=ATRC.
- [23] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ascad database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.
- [24] Guilherme Perin, Lichao Wu, and Stjepan Picek. AISY - Deep Learning-based Framework for Side-Channel Analysis. Cryptology ePrint Archive, Report 2021/357, 2021. <https://eprint.iacr.org/2021/357>.
- [25] eShard. The SCARed Framework. Available at <https://eshard.gitlab.io/scared/knowledge/essentials/introduction.html>.
- [26] Elie Bursztein. Deep-Cryptanalysis, Fashion or Revolution? Available at <https://elie.net/static/files/deep-cryptanalysis-fashion-or-revolution/deep-cryptanalysis-fashion-or-revolution-slides.pdf>.
- [27] Jacob Beningo. Why Won't the 8-bit Microcontroller Die? Available at <https://www.embedded.com/why-wont-the-8-bit-microcontroller-die/>, May 2022. Latest access in February 2023.
- [28] Microchip. Curiosity HPC Documentation. Available at https://ww1.microchip.com/downloads/en/DeviceDoc/Curiosity_HPC_Schematics_rev2.pdf.
- [29] NewAE. Chipwhisperer Starter Kits. Available at <https://rtfm.newae.com/Starter%20Kits/>.
- [30] NewAE. Chipwhisperer API. Available at <https://chipwhisperer.readthedocs.io/en/latest/index.html#api>.

- [31] The European Commission. Digital Economy and Society Index (DESI) 2022. Available at <https://digital-strategy.ec.europa.eu/en/policies/desi>.
- [32] Zhou YongBin and Feng DengGuo. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing. 2005.
- [33] Loic Masure, Cécile Dumas, and Emmanuel Prouff. A Comprehensive Study of Deep Learning for Side-Channel Analysis. 2019.
- [34] github.com/AISyLab. The AISY Framework. Available at https://github.com/AISyLab/AISY_Framework.
- [35] Faiqa Maqsood, Muhammad Ahmed, Muhammad Mumtaz, and Munam Shah. Cryptography: A comparative analysis for modern techniques. *International Journal of Advanced Computer Science and Applications*, 2017.
- [36] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.
- [37] Samuele Yves Cerini. Empirical evaluation of the resilience of novel s-box implementations against power side-channel attacks. Aprile 2021.
- [38] M.R. Z'aba and M.A. Maarof. A survey on the cryptanalysis of the advanced encryption standard. *Proceedings of the Postgraduate Annual Research Seminar*, 2006.
- [39] Wikipedia Community. AES implementations. Available at https://en.wikipedia.org/wiki/AES_implementations, February 2023. Latest access in February 2023.
- [40] github.com/kokke. Tiny-AES-C. Available at <https://github.com/kokke/tiny-AES-c>.
- [41] Digikey Inc. Digikey Microcontrollers List. Available at <https://www.digikey.it/it/products/filter/embedded/microcontroller/685>.
- [42] Mouser Inc. Mouser Microcontrollers List. Available at <https://www2.mouser.com/c/semiconductors/integrated-circuits-ics/embedded-processors-controllers/microcontrollers-mcu/>.
- [43] NIST. Advanced Encryption Standard Algorithm Validation System(AESAVS). Available at <https://csrc.nist.gov/projects/cryptographic-algorithm-validation-program/block-ciphers#AES>.
- [44] NATO. The TEMPEST Standard. Available at <https://www.ia.nato.int/niapc/tempest/certification-scheme>.
- [45] Paul Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. 1996.
- [46] Gabriel Goller and Georg Sigl. Side channel attacks on smartphones and embedded devices using standard radio equipment. 2015.
- [47] Ross Anderson. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2020. ISBN: 1119642787.
- [48] Vojtech Miskovsky Petr Socha and Martin Novotny. A comprehensive survey on the non-invasive passive side-channel analysis. 2022.

- [49] Muench M. Hayes T. Francillon A. Giovanni Camurati G., Poeplau S. Screaming channels: When electromagnetic side channels meet radio transceivers. 2018.
- [50] Schmidt J. Hutter M. The temperature side channel and heating fault attacks. 2014.
- [51] Dan Maloney. Audio eavesdropping exploit might make that clicky keyboard less cool. Available at <https://hackaday.com/2022/05/06/audio-eavesdropping-exploit-might-make-that-clicky-keyboard-less-cool/>.
- [52] Hongsheng Wang, Daogang Ji, Yang Zhang, and Kaiyan Chen. Optical side-channel dependency analysis on microcontroller chip. In *Proceedings of the 2014 International Conference on Mechatronics, Control and Electronic Engineering*, pages 147–150. Atlantis Press, 2014/03.
- [53] R. Anderson, M. Bond, J. Clulow, and S. Skorobogatov. Cryptographic processors-a survey. *Proceedings of the IEEE*, 94(2):357–369, 2006.
- [54] Benjamin Timon. Non-profiled deep learning-based side-channel attacks with sensitivity analysis. 2019:107–131, 2019.
- [55] Can Aknesil and Elena Dubrova. Towards generic power/em side-channel attacks: Memory leakage on general-purpose computers. In *2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC)*, pages 1–6. IEEE, 2022.
- [56] Alla Levina, Pavel Borisenko, and Roman Mostovoy. Sca as mobile security threat. In *2017 20th Conference of Open Innovations Association (FRUCT)*, pages 236–241. IEEE, 2017.
- [57] Younis A Younis, Kashif Kifayat, and Madjid Merabti. Cache side-channel attacks in cloud computing. 2014.
- [58] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO’ 99*, pages 388–397, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.
- [59] Thanh-Ha Le, Cécile Canovas, and Jessy Clédière. An overview of side channel analysis attacks. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security, ASIACCS ’08*, page 33–43, New York, NY, USA, 2008. Association for Computing Machinery.
- [60] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 16–29, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [61] Manfredi Di Lorenzo. Benchmark analysis of smart ticketing systems. the transport companies point of view. 2022.
- [62] Elie Bursztein. A hacker guide to deep-learning based AES side channel attacks - DEFCON 27. Available at https://www.youtube.com/watch?v=Db8mj5KFz8E&t=1584s&ab_channel=ElieBursztein.
- [63] K. Tiri, M. Akmal, and I. Verbauwhede. A dynamic and differential cmos logic with signal independent power consumption to withstand differential power analysis on smart cards. In *Proceedings of the 28th European Solid-State Circuits Conference*, pages 403–406, 2002.

- [79] Rajit Sanghvi. A Complete Guide to Adam and RMSprop Optimizer. Available at <https://medium.com/analytics-vidhya/a-complete-guide-to-adam-and-rmsprop-optimizer-75f4502d83be>.
- [80] Petro Liashchynskiy and Pavlo Liashchynskiy. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. Available at <https://arxiv.org/pdf/1912.06059.pdf>.
- [81] Alan Jovic Shivam Bhasin Francesco Regazzoni Stjepan Picek, Annelie Heuser. The Curse of Class Imbalance and Conflicting Metrics with Machine Learning for Side-channel Evaluation. Available at https://www.youtube.com/watch?v=Zgms_hs366M&ab_channel=TheIACR.
- [82] ANSSI. The ASCAD Dataset. Available at <https://github.com/ANSSI-FR/ASCAD>.
- [83] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *NDSS 2020-Network and Distributed System Security Symposium*, pages 1–14, 2020.
- [84] Texas Instruments. TMS320C54x, TMS320LC54x, TMS320VC54x Fixed-Point Digital Signal Processors. Available at https://www.ti.com/lit/ds/symlink/tms320lc542.pdf?ts=1680685393843&ref_url=https%253A%252F%252Fwww.google.it%252F.
- [85] NewAE. Chipwhisperer Firmware Examples. Available at <https://github.com/newaetech/chipwhisperer/tree/develop/hardware/victims/firmware>.
- [86] Microchip. PIC18F27K42 Datasheet. Available at <https://ww1.microchip.com/downloads/aemDocuments/documents/MCU08/ProductDocuments/DataSheets/PIC18%28L%29F26-27-45-46-47-55-56-57K42-Data-Sheet-40001919G.pdf>.
- [87] Google. Keras. Available at <https://keras.io/>.
- [88] Google. Overview of TensorFlow. Available at <https://www.tensorflow.org/overview?hl=it>.
- [89] Intel. Processore Intel Xeon E5-2650 - 20 MB di cache, 2,00 GHz, QPI Intel 8,00 GT/s. Available at <https://ark.intel.com/content/www/it/it/ark/products/64590/intel-xeon-processor-e52650-20m-cache-2-00-ghz-8-00-gts-intel-qp.html>.