TÉCNICO LISBOA

Deep Learning based Side Channel Attacks on Intel CPU

Guilherme Rodrigues Lourenço

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisor(s): Prof. Ricardo Jorge Fernandes Chaves
Prof. Aleksandar Ilic

Examination Committee

Chairperson: Prof. Pedro Filipe Zeferino Aidos Tomás
Supervisor: Prof. Aleksandar Ilic
Member of the Committee: Prof. Alberto Manuel Ramos da Cunha

November 2022

Declaration

I declare that this document is an original work of my own authorship and that it fulfils all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

This chapter of my life will soon come to an end, and like so, I wish to express my true appreciation to all those who have supported me throughout all of these six years, four in the Military Academy and the last two in Instituto Superior Técnico.

Firstly, I am honored to express my deep gratitude to my family. I would like to thank my parents, my mother, Alexandra, and my father, António, for all their love, kindness, and incredible support. I would also like to thank my brother, Gonçalo, for always cheering me up and also for the incredible patience that he has.

Words cannot express my gratitude to my girlfriend, Helena, for never letting me lose hope about this project and always helping me whenever I needed it, which happened many times. Since the beginning, you were the one who saw me at my best but also, many times, at my worst but never let me or us down. Thank you for everything.

To my closest friends, Gonçalo, Serra, Lucas, Carmona, Marta, and Maria, it was magnificent to feel your constant support, celebrating with me all the small milestones and also sharing the saddest days. To all the QNFEF members, Catarina, Mariana, Pedro, PC, Zé, I sincerely thank you for your friendship and all the good memories we have together, even in the most stressful of times.

I would also like to thank Moura, Rocha, Catarino, Dias, Jin, Gomes, Pereira, Silva, Gonçalves, and all the friends I made at Military Academy, for without you, this journey would have been much harder and a lot less fun. Thank you for all the late nights and experiences we shared over these six years.

Last but not least, I would like to give my warmest thanks to my advisors, Prof. Ricardo Chaves and Prof. Aleksandar Ilic, for all the work they invested into this Thesis and also to André Silva for always answering my emails and finding time to help me out. I also thank Military Academy for raising in me the necessary capabilities and mentality to face all the challenges and also INESC-ID for giving me chance to utilize the resources and working space I needed to pursue this project.

Thank you all.

Resumo

Hoje em dia, as implementações criptográficas estão cada vez mais expostas aos chamados *side-channel attacks*, que analisam as fugas (de potência, tempo, etc.) produzidas pela implementação criptográfica para descobrir informações pessoais. Até há pouco tempo, os *side-channel attacks* eram quase exclusivamente realizados com métodos estatísticos tradicionais que visavam as supostas fugas. Ultimamente, porém, os métodos de aprendizagem profunda têm mostrado grandes resultados em várias áreas da tecnologia, pelo que é natural que possam também ser utilizados para realizar *side-channel attacks*.

O objetivo deste trabalho é investigar a possibilidade e capacidade de um *side-channel attack*, utilizando os contadores de energia do processador para adquirir as chaves de encriptação utilizadas pelo mesmo. Para o conseguir, vamos implementar um *side-channel attack* de potência num processador Intel, com o objetivo de conquistar acesso à interface *Running Average Power Limit* que mostra valores directamente associados ao consumo de energia do dispositivo, e de seguida iremos aplicar técnicas de aprendizagem profunda, utilizando diferentes arquitecturas de redes neuronais, para descobrir a chave secreta na totalidade.

Nesta dissertação, os dados provenientes das medições do *software* foram adquiridos a partir de versões simplificadas do *Advance Encryption Standard*. Estes dados são posteriormente fornecidos às redes neuronais para treino e implementação do ataque. Quando colocados em condições de teste, os resultados obtidos pelos modelos de redes neuronais mostraram que ocorrem fugas no processador e que nas mesmas condições em que foram adquiridos os dados utilizados para criar os modelos, é possível descobrir o *byte* secreto pretendido.

Palavras-chave: implementações criptográficas, side-channel attacks, segurança, aprendizagem profunda, redes neuronais

Abstract

Nowadays, cryptographic implementations are increasingly exposed to the so-called *side-channel attacks*, which observe the leakage (power, time, etc.) produced by the cryptographic implementation to discover internal secrets. Until recently, *side-channel attacks* were almost exclusively performed with traditional statistical methods that target the supposed leakages. Lately, however, deep learning methods, have shown great results in several areas of technology, so it is natural that this novel technology might also be implemented to *side-channel attacks*.

In this work, we aim to investigate the possibility and capability of a *side-channel attack* using the power measured by the processor to acquire the encryption keys used by that device. To achieve this, we are going to introduce a software-based power *side-channel attacks* into an Intel processor with the objective of conquering undeserved access to the Intel *Running Average Power Limit* interface that shows values directly associated with the power consumption of the device, and then we will apply deep learning techniques using different deep neural network architectures in order to be able to discover the whole key.

For this Thesis, the data coming from the *software* measurements was acquired from simpler versions of the *Advance Encryption Standard* and then provided to the neural networks for training and implementation of the attack. The results obtained by the originated neural network models, when put under test, showed that leakage occurs and that along the same conditions of the data used to create the models, it is possible to discover the targeted secret byte.

Keywords: cryptographic implementations, side-channel attacks, security, deep learning, deep neural networks.

Contents

List of Tables

List of Figures

Acronyms

AES Advanced Encryption Standard

ALU Arithmetic Logic Unit

CNN Convolutional Neural Network

CPA Correlative Power Analysis

CPU Central Processing Unit

CU Control Unit

DEMA Differential Electromagnetic Analysis

DES Data Encryption Standard

DL Deep Learning

DNN Deep Neural Network

DUT Device Under Test

ECC Elliptic Curve Cryptography

EMA Electromagnetic Analysis

FFNN Feed-Forward Neural Network

FPR False Positive Rate

HD Hamming Distance

HDF Hierarchical Data Format

HW Hamming Weight

LS-SVM Least Square Support Vector Machine

LSTM Long Short-Term Memory

ML Machine Learning

MLP Multi-Layer Perceptron

MSR Model-Specific Register

NN Neural Network

PA Power Analysis

PP Power Plane

RAPL Running Average Power Limit

RNN Recurrent Neural Network

RSA Rivest-Shamir-Adleman

SCA Side-Channel Attack

SEMA Simple Electromagnetic Analysis

SGX Software Guard Extensions

SIMD Single Instruction Multiple Data

SIMD Single Instruction, Multiple Data

SPA Differential Power Analysis

SPA Simple Power Analysis

SVM Support Vector Machine

TA Template Attacks

TPR True Positive Rate

TSVM Twin Support Vector Machine

Chapter 1

Introduction

This first chapter introduces and motivates the topic of this MSc Thesis entitled “Deep Learning based Side Channel Attacks on Intel CPU”. Firstly starts by clarifying the context and motivation, leading to the description of this work’s objectives and finalizes with the structure of this document.

1.1 Context and Motivation

The world is in the middle of a transition phase, where all type of documents and information from the less relevant to the most critical, in the vast majority, are kept online. Nowadays it is unimaginable to spend one single day without using any device that is not connected to a network, either using it to transfer data from one device to another or to just simply save personal information in it. For these reasons, the cyberspace and the equipment used to store the information are getting more and more attention from the users and the attackers. That is why security has been an increasing and significant concern in computing and communications systems. Security is the process of setting up several practices to keep the personal data safe from unauthorized access, preventing the information from being misused, destructed, modified, or even just disclosed [1].

Cryptographic algorithms, including public-key ciphers, symmetric ciphers, and hash functions, create a set of characteristics that make it possible to construct security mechanisms that target specific objectives [2]. However, these security mechanisms, by themselves, might not be enough security solutions for the real world threats. It should be assumed that attackers will not directly get through the computational complexity and break the cryptographic primitives employed in security mechanisms. In other words, the attackers are able to bypass altogether or significantly weaken the theoretical strength of security solutions by using alternative information that leads to the recovery of part of, or the full, employed secret key [3]. This type of attack is called a side-channel attack (SCA), and it works because there is a correlation between the physical measurements taken during computations, such as power consumption, electromagnetic radiation, and computing time and the internal state of the processing device, which is itself related to the secret key that in several cases happen to be out of the security systems control. The correlation between the side channel information and the operation is related to

the secret key that the SCA tries to find [4].

Recently machine learning (ML) has become one of the most versatile and efficient new technologies for processing large amounts of data and finding underlying patterns that are difficult to recognize using other methods. The purpose of Machine Learning is to predict future outcomes based on substantial amounts of previously learned examples.

Several studies [5, 6] and real life cases [7, 8] have shown that Machine Learning techniques, and more particularly, deep learning (DL) techniques, are remarkable at generalizing and performing classification tasks such as spam filtering, predicting if bank transactions are legitimate, assigning a diagnosis to a patient with specific symptoms, and image, document, and malware classification. deep learning techniques have also been applied to time series data classification with outstanding results [9–11], and more specifically, these techniques have also been experimentally applied to SCAs with promising results [12, 13].

For this reason DL is a subject of interest in this work, and it is speculated that employing DL classification techniques to perform a SCA on data that has been acquired solely using software may have a positive impact.

1.2 Objectives and Expected Contributions

The objective of this MSc Thesis is to use DL methods in order to perform a power analysis side-channel attack by studying the possibility of using the processor's power measurement counters. It is possible to separate the final objective into three smaller goals:

1. Obtain power measurements from the AES implementation and finding a leakage that relates to the used secret key;
2. Designing a DL architecture capable of relating the leakages with the secret key ;
3. Perform a side-channel attack with the best DL model.

1.3 MSc Thesis Report Outline

In order to achieve the final objective, this report will be structured as follows: in Chapter 2, cryptography, computer architecture, and deep learning methods are briefly overviewed; in Chapter 3, first, the basic concepts and tools of power analysis and other types of attacks are reviewed, after some countermeasures and the attacks based on information retrieved by software, and finally deep learning approaches for SCAs; in Chapter 4, the methodological approach for the MSc Thesis is presented; in Chapter 5 the experimental setup is described; in Chapters 6 and 7, the results are presented and discussed and, finally, Chapter 8 concludes this report and highlighting some future work.

Chapter 2

Background

As mentioned in Chapter 1, several goals need to be considered in security, primarily how it works, the weaknesses, and improvements that can be done. So, knowing how cryptography works and how exactly the PC components architectures are, makes the difference when it is time to understand the problem.

2.1 Security

Privacy and security have been proven one of the most challenging areas, with many published works about them in the last years, although privacy and security are not entirely defined. Both industrial parts and research sectors have proposed alternative and different definitions. Heterogeneity and size are two main factors that characterize the recent technologies [14]. Security is needed to implement efficient algorithmic schemes and protocol implementations in different devices and a considerable number of applications. Other methodologies have to be created with the final objective of supporting heterogeneity and scalability, keeping users anonymous, and taking care of personal data protection. In addition to the convenience of having both applications and services supported by IoT, the challenge of security is also an excellent principle for trust.

There are two different concepts, Symmetric and Asymmetric Cryptography, that can be combined to provide confidentiality, authentication, and non-repudiation to a message, which will subsequently be discussed in depth.

The following sections will present the main ways to secure and at the same time access sensitive data.

2.2 Cryptography

Cryptography is the area of information security that protects sensitive data from unauthorized users. The kind of data covered by cryptography goes from resting data files to messages, passwords, and many others. The secure transmission of information through insecure channels is allowed to ensure

confidentiality. Moreover, a cipher and a secret key are required to achieve these goals. A cipher is a cryptographic algorithm responsible for performing encryption or decryption.

- **Encryption** is the process that converts data that usually is in a readable form, called plaintext, into an unintelligible form, called ciphertext;
- **Decryption** works oppositely; it transforms the ciphertext into the original plaintext form. The secrets represent the keys used by the cipher's encryption and decryption processes and are composed of large numbers.

There are several cryptographic services relevant to this work, that are guaranteed by two key types: symmetric and asymmetric.

2.2.1 Symmetric Cryptography

At least two communicating parties share symmetric keys. Symmetric keys are smaller, and the operations are faster than with asymmetric keys. The same key is used to encrypt and decrypt (see Figure 2.1). The main disadvantage of this system is that sharing a secret key between different parties is usually not simple, especially thinking in a realistic way, where there are no secure channels to communicate.

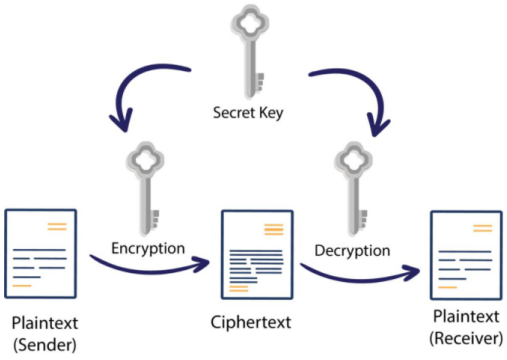

Figure 2.1: Symmetric Cryptography [15].

Advanced Encryption Standard

One of the most popular symmetric-key algorithms is the Advanced Encryption Standard (AES). This algorithm uses 128-bit blocks for block cipher modes, and the keys can be 128, 192, or 256 bits. The algorithm performs 10, 12, or 14 transformation rounds, depending on the size of the cipher key used. Those transformations are usually applied on 16 bytes of data, called the state, represented as a 4x4 byte "state matrix". A round is composed of several operations that should transform the input into an output based on those transformations. Each round key is derived from the cipher key, using a key expansion procedure. Every round uses the same number of bits for the key since the beginning, and each round requires a different 128-bit key, referred to as "round key". During a round, four types of operations may be performed:

- **AddRoundKey:** In this operation, each byte from the state is combined with a byte from the round key using an XOR operation;
- **SubBytes:** In this operation, each byte from the state is replaced (in a non-linear fashion) by another byte, according to a lookup table. The process of replacing each byte by resorting to this lookup table is known as S-Box;
- **ShiftRows:** This operation applies a left shift to each state matrix row, except for the first one. The shifting amount applied to each row follows the formula $n - 1$, where n is the row index, e.g., for the second row, all bytes are shifted one position to the left;
- **MixColumns:** In this operation, each column from the state matrix is linearly mixed, i.e., all the bytes from a column are combined to obtain a new column.

The last round is the only that it is not wholly equal to the others, since the last operation is not performed.

2.2.2 Asymmetric Cryptography

Asymmetric keys go in pairs to each party, where each pair has one private and one public key. The private key is meant to be known only by its owner. The public key is shared with the other parties (see Figure 2.2). Once two sides have traded public keys, asymmetric and symmetric keys can be mixed in a hybrid encryption scheme. The scheme has the leverage of the faster symmetric encryption to cipher the data and the asymmetric encryption to encrypt the symmetric key, providing authentication. Alternatively, it can be used to share symmetric keys for usage with an authenticated-encryption scheme.

Asymmetric keys, as mentioned before, are more extensive, and the operations work more slowly when compared to symmetric keys. When encrypting a message with the public key, only the owner of the private key will be able to decipher the content, offering confidentiality to the message. There are three popular algorithms for public-key encryption, Rivest-Shamir-Adleman (RSA), elliptic curve cryptography (ECC), and discrete logarithm problem (El Gamal System).

Due to the innovation in quantum computers, the RSA algorithm could become obsolete in the future because the probability of surviving an attack from these computers is low.

ECC is a more recent algorithm based on the Elliptic Curve Discrete Logarithm Problem. ECC encryption is significantly harder to break than RSA; the key sizes demonstrate this since ECC keys are smaller for the same level of security.

2.3 Cryptanalysis

To find the weaknesses of a cryptographic system, it is relevant to attack the system through Cryptanalytic attacks. The attacks are based on the algorithm nature and on the capacity to understand the general characteristics of the plaintext, which goes from a regular document written in any language to a code written in any language. Moreover, the plaintext should be known before starting the attacks.

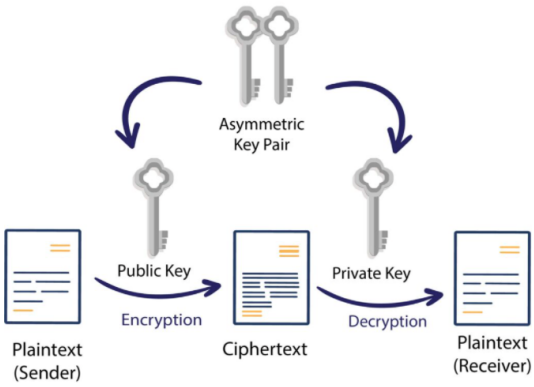

Figure 2.2: Asymmetric Cryptography [15].

Starting an attack into a cipher or a cryptographic system always has an unknown result; it might be a complete or only partial success. If the security is compromised, the attacker can extract various amounts of information. The objective is to optimize the time and computational resources it takes to obtain sensitive information. It is nonviable for most cases since existing algorithms usually use 128 or more bits size keys that make the search computationally expensive. There are many types of cryptanalytic attacks, but they can be divided into two different groups: with or without physical computer access [16].

- **Black-Box Model:** This module is based on the assumption that the attacker has no physical access to the key or even to the internal system information and behavior; it can only notice the external information (see Figure 2.3). With no possibility to see the code execution and the dynamic encryption operations, the objective is to reveal information about the system in plaintext or ciphertext format. It has three possible levels of attacks:

  1. The first level of attack is called “passive attack”, which is only capable of observing the input and output of the black box system;
  2. The second attack level is called “active attack”, which involves direct interaction;
  3. The third level of attack is called “adaptive attack”, which tries to collect information by choosing a ciphertext and then obtaining its decryption using an unknown key.

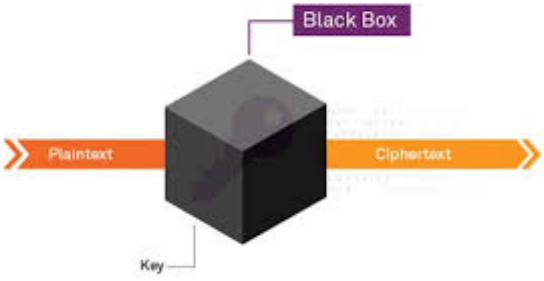

Figure 2.3: Black-Box Model [16].

- **Grey-Box Model:** The grey box model is also called the SCA or the partial access attack. The attacks rely on information obtained from a physical implementation of a cryptosystem instead of a brute force attack. The attacker obtains information from power consumption, electromagnetic radiation or fault analysis and uses that information to break the system. The grey box attack stands by the opportunity to weaken the security of a system, whether it is by accessing the inner work, side effect, or execution of an algorithm. The grey box scenario assumes that the attacker has partial physical access to the key or to what it is "leaking", the so-called side-channel information (see Figure 2.4).

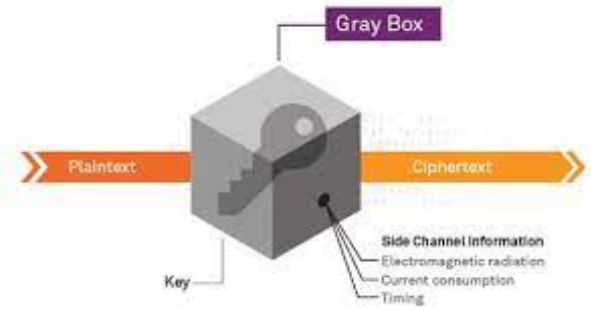

Figure 2.4: Grey-Box Model [16].

- **White-Box Model:** In this model, the attacker has full control of the targets execution environment, assuming that:

  1. Attacks with full privilege have complete access to the implementation algorithms;
  2. Dynamic execution can be observed, and important data such as cryptographic keys can be seen;
  3. Detailed algorithms in the system are completely visible and alterable.

  In this model, the attacker can get the cryptographic key once he has the information about all the algorithms and can also observe the dynamic execution, which will let him recover the keys from memory (see Figure 2.5). The attacker has the highest attack potential over the cryptography system when using this model attack. In contrast with previously described scenarios, the white-box scenario offers far more severe threats while assuming attackers have complete visibility and control over the whole operation. As previously mentioned, the attacker can freely observe the dynamic of the whole system; from the code execution to the internal algorithm, details are wholly exposed and alterable at will. Despite this fully transparent methodology, white box cryptography integrates the cipher in a manner that does not reveal the key.

  As exposed in the white box scenario, the key is assumed as part of the implementation, but the white box cryptography algorithm is protected, as the key is not present in memory and cannot be extracted – not even dynamically. Like so, the most secure cryptographic model is the one that defends against the majority of malicious threats – precisely what white box cryptography attempts to achieve.

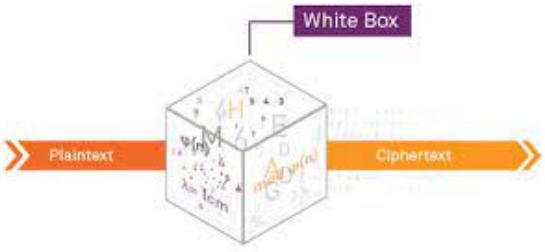

Figure 2.5: White-Box Model [16].

2.4 Side-Channel Attacks

It is possible to see by the chapters written above that security exists to protect the users from possible attacks nowadays; however, plenty of those attacks still happen, and the main contributors to that are the SCAs, represented in Figure 2.6. Security in electronic devices must not rely only on cryptographic algorithms proven to be mathematically secure. An attacker may use information leaked from side channels about the underlying operations through power consumption, electromagnetic radiation, heat, execution timings, or even sound.

These attacks can be much more powerful than traditional black-Box attacks and present an ever-increasing threat to cryptosystems. In fact, most actual cryptographic algorithms have been broken with SCAs. In an SCA, the attacker exploits the leakages of the system, intending to gather information about the executed operations. If enough information is collected, their later analysis will reveal the secret key.

SCA's can be separated by their characteristics:

- Active attacks exploit abnormal behaviors in the devices, sometimes by tampering with them, in order to obtain secret information in a way that would not be possible under normal usage;
- Passive attacks, observe and analyze the device behavior under normal functioning conditions;

and by their capacity of intrusion:

- Non-invasive attacks, leave the device intact and resort to other components to gather information. It is restricted to observing the activity, with the possible aid of tools that do not alter the device, such as timers and oscilloscopes;
- Semi-invasive attacks unpack the chip as well, but there is no direct contact with the chip surface;
- Invasive attacks require a action directly on the device by the attacker, usually removing the chip package so that the chip surface can be probed.

The origin of these attacks dates back to 1996, when Kocher [17] broke RSA and Diffie-Hellman systems (asymmetrical encryption algorithms) based on the time it took to perform the operations. Three years later, as described in [18], he used power measurements to break the Data Encryption Standard (DES), the most commonly used symmetrical encryption algorithm at the time and proved capable of

doing so on other algorithms. Given the importance of this family of attacks, under which most cryptographic algorithms have failed to resist when not adequately protected, a big concern is given to the corresponding countermeasures. These can range from software changes to extensive hardware modifications.

This work emphasizes power consumption attacks in a non-invasive way and the corresponding countermeasures.

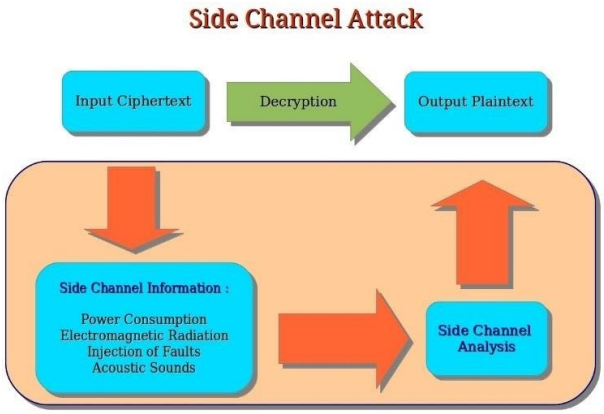

Figure 2.6: Side-Channel Attack [19].

2.5 Computer Architecture

There are specific components in a computer that are essential to its functions, namely the Central Processing Unit (CPU). The CPU holds a limited number of storage locations, known as registers, a Control Unit (CU), and an Arithmetic Logic Unit (ALU), as seen in Figure 2.7. The clock is responsible for synchronizing the internal operations of the CPU with other components in the system. The CU organizes the ordering of steps responsible for doing system instructions. The ALU performs arithmetic operations such as addition and subtraction and logic operations such as AND, OR, and NOT [20].

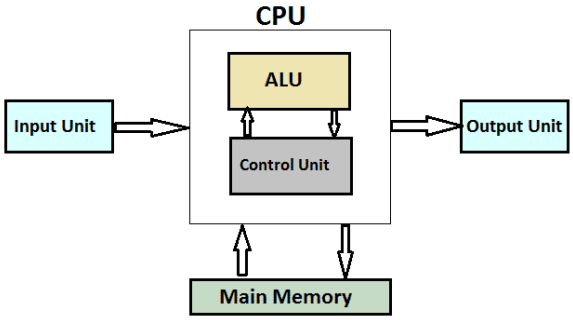

Figure 2.7: CPU Internal Parts [21].

The needing for an existent connection between the user and the components turned out being accomplished based on programs written in specific programming languages, creating the bridge to a lower-level programming language, called Assembly, which coordinates the computational architecture

[22]. After that, depending on the processor used, the Assembly code is turned into an executable (by an assembler) containing machine code instructions. Micro-architecture is the digital logic that allows an instruction set to be executed. The x86 is a family of instruction set architectures created and supported by Intel. All of this combined, implementation of registers, memory, arithmetic logic units, multiplexers, and any other digital logic blocks form the processor.

All modern Intel processors are based on the same Intel Core architecture, where each new generation only introduces the distinctions from the previous versions of Intel microprocessors, mainly in the number of components each microprocessor has and the number of operations it can perform. Each Intel processor also comes with a new feature, the Running Average Power Limit (RAPL), that allows not only to measure the power consumed but also their performance [23].

In an attempt to further increase the processing speed, most computers have a microprocessor chip with several processors inside it, called cores. These cores can vary without changing the micro-architecture, exemplified in Figure 2.8, the RAPL feature has a specific domain, the Power Plane 0 (PP0), which measures the power consumption of the cores. The other components that can also be seen in Figure 2.8 are the “Gen9” which corresponds to the integrated graphic unit of the processor that works as a dedicated core just for the image processing and output, the “System Agent” which has the function of handling tasks important to the processing, such as memory management and control and the “Ring Interconnect” used for intra-processor communication between all the components previously referred [24]. It is essential to understand that there are different types of microprocessors, which cause different results in terms of performance. In the scope of this thesis, we will focus on Intel’s most recent used microprocessor, which will be presented in-depth.

All modern Intel processors are based on the same Intel Core architecture, where each new generation only introduces the distinctions from the previous versions of Intel microprocessors, mainly in the number of components each microprocessor has and the number of operations it can perform. To be able to perform the operations, the values used are stored in the registers. The CPU will later read and process those stored values.

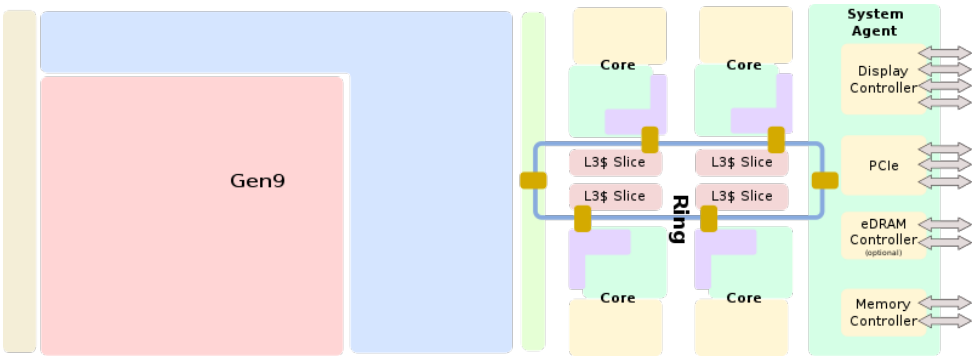

Figure 2.8: General diagram of Intel’s processor with 4 cores [25].

2.6 Deep Learning: Basics, Classification and Review

After starting by presenting the main concepts and tools on Deep Learning, this chapter will review the most relevant SCA solutions in the literature adopting this technological paradigm.

2.6.1 Basic Concepts and Tools

Deep learning is a subdivision area of machine learning that uses tools and algorithms based on artificial neural networks, also known as neural networks (NN). These NNs are inspired by biological brains, and the neurons that compose them. Each NN is made up of neurons, or nodes, which conduct signals with information to each other, just like biological neurons transmit information through synapses. The nodes all work together to precisely recognize, classify, and describe objects within the data. Neural networks are effective because they are able to perform complex tasks very efficiently, notably by learning from a set of examples. There are several types of learning methods, of which the most relevant ones are:

- Supervised methods: where a model learns from labeled examples and like so, it knows the ground truth labels for specific input data and can understand the relationship between the input (data) and output (labels);
- Unsupervised methods: learn a model from unlabeled examples, where the inferences are achieved by extracting features and identify patterns.

Similar to an organic neuron, the nodes in an NN receive input signals which are processed, and their single output is transmitted to other neurons. The activation function in most cases is non-linear and plays a truly important role in the approximation of complex functions as well as the convergence of the learning process. The most used activation functions are:

- The rectified linear unit (ReLU), that gives an output that is the maximum between 0 and the input value;
- The logistic function, that gives as output a value between 0 and 1 based on the input value;
- The softmax function, which is a generalization of the Logistic function to multiple dimensions.

In Figure 2.9 it is possible to see the theoretical model of a neuron (k). Each neuron receives the input signals (x_m), which are multiplied by the corresponding weights (w_{km}) in a linear fashion. These weights may be negative or positive and are learned to achieve the best combination of impacts given to each specific input. After the inputs are multiplied by their respective weights, they are added together with a bias (b_k), thus allowing the neuron to represent any linear function. The activation function ($\phi(v_k)$) receives the previous linear sum and produces the final output (y_k) of the neuron.

In this context, a DL model consists of the hyper parameters, which are different parameters that define the structure of the NN and its type of training, that is, how many layers, how many nodes in each layer, which type of layers, as well as the loss function, activation functions and optimization algorithms

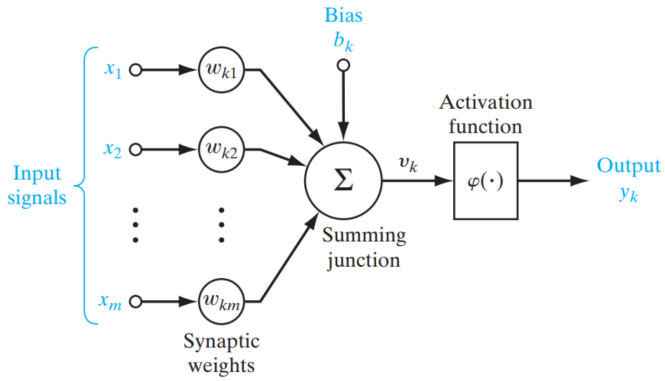

Figure 2.9: Model of a neuron [26].

to use, among other hyper parameters. The input and output layers of a deep neural network (DNN) are denominated visible layers. The DL model ingests the data for processing through the input layer, the final prediction or classification is made on the output layer.

Neural networks are created based on many neurons, that are all linked generating a connected graph organized in layers. These layers represent groups of neurons usually at the same depth in the network and operating in parallel, and they may be of different types. The most common ones are:

- Fully connected layer that happens when each neuron of one layer connects to all the neurons of the next layer;
- Convolutional layer;
- Pooling layer.

Based on the type of graph created by all the links, an NN may be classified as:

- Feed-Forward neural network (FFNN);
- Recurrent neural network (RNN).

FFNNs have inputs that go from the input layer to the output layer, without any cycles, which means that the output from one layer is only capable of affecting a future layer and never previous ones. Layers are formed by gathering many neurons and combining them.

RNNs have cycles in the network, which allows the output from one layer to not only influence the next layers but also influence the same layer itself, or even the layers before. This type of NN is very useful with inputs that have temporal correlation, given that it can combine past information with current information to define the neuron output.

Since the structure of the networks may be complex and diverse, it is usual for networks to be classified based on their properties. This is commonly done according to the operations applied at the majority of the layers, either by their structure or the connections between those layers. Overall it could be easier to consider an NN as a structured set of layers that implement operations, instead of characterizing them by the architecture that may or may not fall within a distinct class of networks.

2.6.2 Deep Learning Layers

Layers are constituted by differentiable operations over matrix inputs and can be stacked in depth to create DNNs. In this section, the most relevant layers for SCAs which have already been mentioned will be explained further.

Fully Connected Layer

A fully connected layer of size n learns that same amount of linear combinations from the outputs of the previous layer, and outputs n activations. This allows the layer to learn how to adjust, during training, the importance of the activation of the output of the previous layer to its own. With this operation, it can learn feature maps from the features extracted in the network up until this layer. In a fully connected layer, a position of the output usually specializes in different combinations of features that identify a particular higher-level feature, such as the existence of a square or circle in an image.

Convolution Layer

This type of layer is one of the most important for tasks related to extracting a set of low-level and/or high-level features since it can capture local patterns within an n-dimensional signal, by applying convolution operations to the input. In the particular case of SCAs, convolutions are especially useful in extracting structural redundancy within their input. A convolution layer has a group of filters which are specified by convolutional kernels with weights that are adjusted during training. Each layer implements convolution operations per kernel and the number of kernels in a layer is specified by the number of channels of the convolution layer, where each channel calculates the output based on its kernel. When a kernel is applied to different parts of the input data, multiple outputs are generated, which combined generate the input to the activation function, also known as activation map. The size of the kernel determines the receptive field of the convolution and the granularity of features that it can capture. The size of the full input data is bigger than the convolutional kernel which is defined by the width and height setting, called filter support. The kernel slides across the input according to a parameter called stride, which defines the number of positions the filter skips before processing the next input position. When the kernel stride increases, the size of the activation map decreases. Padding is the parameter that controls the border added to each side of the input before the application of the convolution, in order to guarantee that the filter is applied to the borders of the convolution layer input data. The kernel for an n-dimensional convolution is an n-dimensional matrix, and each application of this kernel will consider an input with the same size.

In Figure 2.10, two convolution operations are illustrated, where the input data is a 6x6 matrix, the kernel has a height and width of 3, and moves with a stride of 2. Each convolutional layer can use more than one kernel at the same time, which will generate one activation map for each kernel and thus produce an output with volume, with the depth corresponding to the number of kernels used.

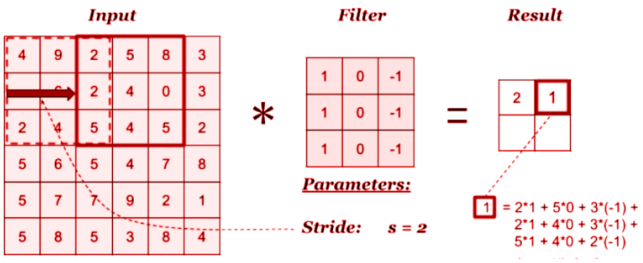

Figure 2.10: Representation of two convolution operations with stride 2 [27]

Pooling Layer

Pooling layers usually appear after convolutional layers with the objective of not only reducing the data volume but also the number of weights in the next layers. Therefore, with the pooling layer it is possible to reduce the computational power (and memory) necessary to process the data, furthermore, this layer can also be used to extract dominant, high-level features. The most common types of pooling operations are:

- The max pooling, which is the most popular pooling method;
- The average pooling.

In Figure 2.11, it is demonstrated how a 2x2 pooling filter moves through the input with a stride of 2, resulting in two positions being skipped between each pooling.

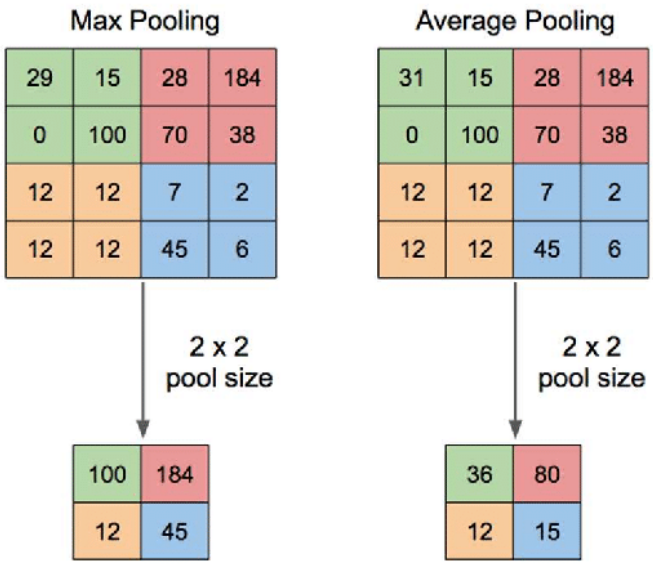

Figure 2.11: Representation of pooling operations with stride 2 [28]

Each type of pooling returns the maximum or average value of a spatial window, respectively, that is over the input layer. Naturally, the size and stride of this window define how much reduction the output data suffers with respect to the input data. After pooling, the output data has smaller dimensions than the input data but it maintains with the same depth.

2.6.3 Deep Learning Network Classes

Fully Connected Neural Network

A Fully Connected neural network (FCNN) is a type of feedforward neural network, where all the layers are fully connected, which implies that each neuron is connected to all neurons from the following layer. This type of NN has three separate types of fully-connected layers:

- The input layer receives the data, that could have been pre-processed and sends it to the remaining network;
- The hidden layers which may be as many as are needed, are all the layers between the input and output layers, responsible for receiving data, processing it, and passing it to the following layer;
- The output layer is the last layer in an NN and provides the final result in a suitable shape for the task being solved.

Multi-Layer Perceptron Network

Multi-Layer perceptron (MLP) is the specific name given to a simpler FCNN architecture composed with at least one hidden layer, that generates a set of outputs from a set of inputs. MLP is a supervised NN that uses backpropagation for training the network. In Figure 2.12 represents an FCNN network with a single hidden layer.

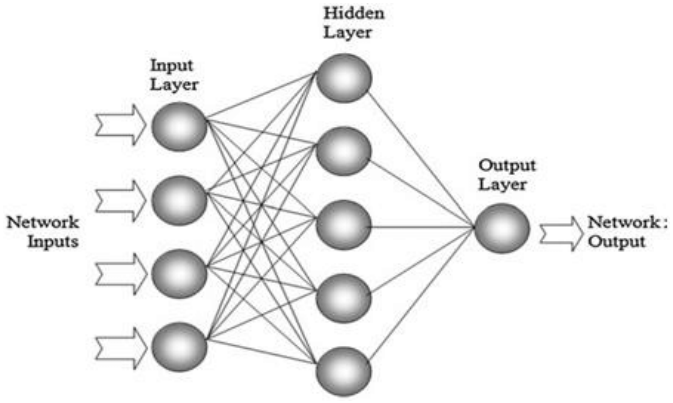

Figure 2.12: Architecture of a Fully Connected neural network with a single hidden layer [29].

Convolutional Neural Network

Convolutional neural network (CNN) is a neural network architecture composed by five different types of layers: input, convolution, pooling, fully connected and output layers in order to produce a hierarchical decomposition of the input signal.

Recurrent Neural Network

Recurrent neural network (RNN) is a neural network architecture typically better at handling data evolving in time, like sequential or time series data. In an RNN, past information is able to influence

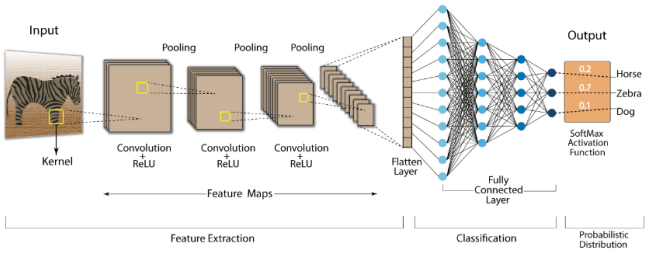

Figure 2.13: CNN architecture example with an image as input, three convolutional layers, three pooling layers and four fully connected layers [30]

current outcomes by taking outputs resulting from prior inputs and inserting them back into a network layer where current signals are being processed. Overall, an RNN propagates the input signal forward as a regular network, but signals are also propagated back to the same neuron that produced it or to a previous one with the objective of keeping a context of what was previously processed. RNNs are able to model functions which have a dependency in time between the input and the output, thus creating cycles which give the network the capability to model dynamic temporal behaviors.

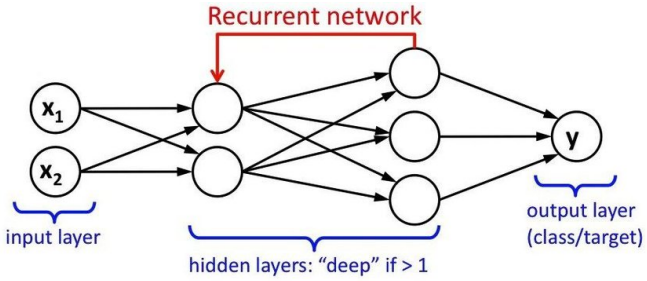

Figure 2.14: RNN architecture example [31].

There are more specific types of RNNs, such as the long short-term memory (LSTM) that is mostly used for natural language processing, speech recognition and many time-based data analysis problems, e.g., action recognition. This type of RNN was specially created to enhance the long-term memory of RNN because, given data with many time steps, it proves difficult for RNNs to recall earlier inputs. Therefore, the LSTM cells are considered important because they are capable of “remembering” inputs for long periods of time.

Convolutional Long Short-term Memory Network

In case of having data collected during successive periods of time, it is usual to characterize them as a time series, and the approach of using ConvLSTM layers is more than acceptable. It is a combination of RNN explained in Section 2.6.3 with CNN explained in Section 2.6.3 networks. It is a model based on LSTM, where the input layers are a mixture of convolutinal layers with time, offering the filtering capacity of the CNNs and the long-term memory of the LSTMs .

2.6.4 Neural Network Training

Optimization and Loss Functions

The process of forward propagation consists in every input going through the whole network where each node takes the input value, computes the output value and sends it to the next layer. This process ends when the values derived from the input reach the output layer.

Another process called back propagation uses algorithms, such as gradient descent, in order to calculate errors in predictions and then modify the weights and biases of the function by moving backwards through the layers with the objective of training the model. All together, forward propagation and back propagation give a neural network the capability to make predictions and correct for any errors accordingly. By repeating the process, the algorithm becomes gradually more accurate. All of these parameters are gathered when the training occurs with the appropriate loss function to achieve a relevant outcome during the learning process.

A dataset is needed (usually with labeled samples) for the learning of a neural network model which can then be generalized for unseen cases.

The optimization of an NN depends on two elements:

- The optimization procedure, which describes the steps and mathematical operations used to configure the weights of an NN in each training operation, with the objective of minimizing the loss function used for that network;
- The loss function calculates the distortion between the input and the output, giving to the training process the objective of minimizing this loss.

The most frequent optimization procedures that are used in research are based on the gradient descent technique. Gradient descent optimization is divided into two phases:

- The forward phase, where the training data is propagated through the network in the forward direction typically represented as left to right, and the corresponding transformations are applied, producing an output at the last layer. At the end of the forward pass, the loss function is calculated, evaluating the performance of the network;
- The backward propagation phase that propagates the error signal backwards which results from the gradient of the loss function that is calculated with relation to each weight of the network.

The gradient of the error with respect to each weight is used to adjust the weight in the direction of minimizing the loss function. To perform the gradient descent, it is necessary to train the network with the entire dataset for each iteration, in order to minimize the global loss of the network. This process is expensive in time and resources if large datasets are used, because a large memory space is needed to store all the data during training and the process of convergence is also slowed down. There are ways to overcome this problem which are based on a more efficient version of the gradient descent technique applied over one or a few training examples at a time, called a batch. The resulting procedure is called

stochastic gradient descent. This version is a probabilistic and randomized estimation of the gradient descent technique which approximates the true gradient of the loss function by considering just part of the training examples in each iteration. The stochastic gradient descent also has different versions, with the objective of improving both the results and the efficiency of the training procedure. One of the most used version, which could be relevant for the proposed work is the Adam Optimizer. This optimizer computes adaptive learning rates for each parameter and is based on the idea of momentum: a residual gradient from the previous iterations, that is used to accelerate the approximation to the solution when the gradient changes in the same direction during several sequential iterations, speeding up the initial convergence during training.

2.6.5 Deep Learning Frameworks

There are various deep learning frameworks that are extensively used for research and production, the most well known ones being Tensorflow and PyTorch. There is a significant difference between these frameworks, which can be characterized by the type of execution:

- The static graph computation has the objective of running and training networks, which means that a neural network must be completely defined before execution, and then can be trained inside the framework's environment, with the user having no control of the execution flow;
- The dynamic graph computation gives the user full control of the environment during the network's training and its structure can change from one iteration to the next, which means that the networks may be dynamically defined during training.

With Tensorflow, the networks are defined statically, where trainable weights and bias are defined as variable Tensors. The inputs are marked with placeholder Tensors that are replaced during training with the input data. The network is then defined as a connected graph of operators (logits) that modify the aforementioned inputs, and produce an output.

In the case of the PyTorch framework, the networks are defined as dynamic graphs. This means that the user has a set of methods that can be applied during execution to create a neural network.

Chapter 3

State of the Art

In this chapter, relevant SCA solutions will be reviewed, firstly analyzing all the most important statistical approaches and then looking into the new deep learning strategies that have been used recently.

3.1 Statistical Approaches

3.1.1 Power Analysis

Power Analysis (PA) is one of the most widely known and used branches of side-channel attacks nowadays. Of all types of SCAs known today, the literature on power analysis attacks and the relevant countermeasures is the largest. Power analysis attack is the current main research focus of side-channel attacks.

This attack was first proposed by Kocher, Jaffe, and Jun in 1999 [18], where the power consumption was measured with an oscilloscope and used to completely get over the Data Encryption Standard (DES) algorithm, that at the time was the encryption algorithm with the most security capabilities. The attack is better on measuring the evolution of the power consumption by a device under any sort of operation. Power traces (see Figure 3.1) often leak meaningful amounts of information about the processed cryptographic key [2].

After getting the results from the measurements, it is crucial to map data values that are processed by the device under test (DUT) into predicted power consumption values, so that later on it may be possible to compare them with actual power consumption values. Power consumption models (also called leakage models) allow doing just that. The most known models are the Hamming-weight (HW), and Hamming-distance (HD)[33]. A summary of their differences will be shown in Table 3.1.

1. **Hamming Weight:** The Hamming weight model is the simplest power model. It is based on the assumption that a bit set to 0 does not lead to significant power consumption, while a bit set to 1 does. As such, this model describes the power consumption of a device, at a given moment, as the number of bits in the data being processed that are set to 1. E.g., the Hamming weight of the logic value ‘1001’ is 2, resulting from having two bits that are set to 1;

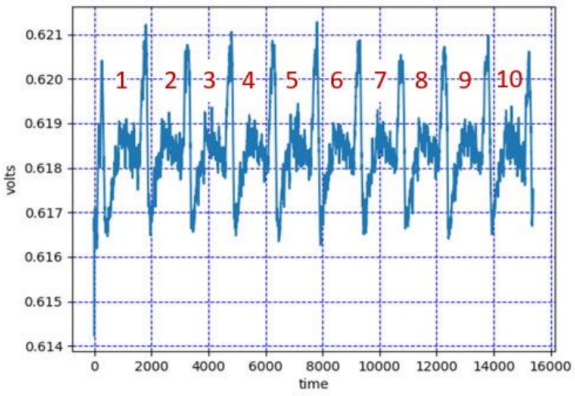

Figure 3.1: Power trace during one encryption with the AES algorithm [32]. The pattern shows 10 similar oscillations representing 10 rounds of the AES algorithm.

2. **Hamming Distance:** The Hamming distance model was proposed by Brier *et al.* in [34] as a generalization to the previous model, making more realistic assumptions based on the CMOS power consumption characteristics discussed previously. This model describes the power consumption of a device as the number of state transitions that occur between two consecutive logic values, i.e., how many bits have changed from 1 to 0 or from 0 to 1, e.g., the Hamming distance between logic values '1000' and '0000' is 1, resulting from having one bit that changed its value.

Table 3.1: Mapping differences between HW, HD power models.

Transitions	HW	HD
$0 \rightarrow 0$	0	0
$0 \rightarrow 1$	1	1
$1 \rightarrow 0$	0	1
$1 \rightarrow 1$	1	0

The most common will be described below.

Simple Power Analysis (SPA)

The objective relies on essentially guessing, through only a few power traces, which specific instruction is being executed at a particular moment in time and what the input and output values are. Attackers typically identify these patterns by visually inspecting the recorded power traces. The attacker needs an exact knowledge of the implementation to execute such an attack. In Figure 3.2 it is possible to have a better view of the one round of AES encryption. It is clearly identified where all the cycles that happen during one round of AES. This specific attack has not been used lately due to its low efficiency.

Differential Power Analysis (DPA)

The attacker exploits the statistical methods in the analysis process, requiring many power traces and, like so, does not need the knowledge of the implementation details. DPA is one of the most potent SCA attacks; however, it does not require many resources. There is another significant difference

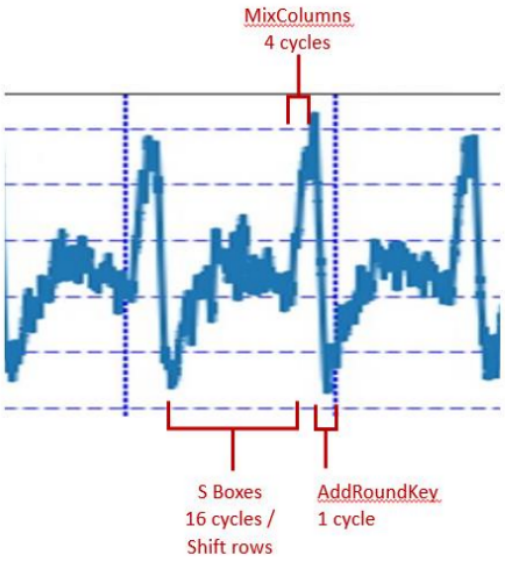

Figure 3.2: One round of AES, as a detail of Figure 3.1 retrieved from [32].

between the SPA and DPA attacks: how the power traces are recorded. In SPA attacks, the power consumed by the device is frequently analyzed along the time axis. The attacker searches for patterns in a single trace. In DPA attacks, the traces form along the time axis are not that relevant. DPA attacks focus on analyzing how the power consumption behaves at particular moments in time depending on the data that is being processed [3]. In its classic approach, the adversary gets a large set of T_i, C_i trace-ciphertext pairs. The adversary also elects a selection function D that takes a ciphertext and a guess of part of the key, which later products one bit. It works based on if the guess is correct, this bit reflects something that is actually noticeable in the computation, but if the guess is wrong, then D will be random across the ciphertexts [2]. After that, the attacker creates a guess K_g and uses this guess, and according to the value of the most significant bit (MSB) of the selection function D output, it is possible to split the set of traces into two sets:

- One for which $D(C_i, K_g) = 0$;
- One for which $D(C_i, K_g) = 1$.

Finally, a distinguisher δ_D is applied to them. The one used originally by Kocher *et al.* consisted of calculating the trace corresponding to the difference of the averages of each set S_0 and S_1 as:

$$\delta_D = S_0 - S_1. \tag{3.1}$$

He averages the traces in each set, and then looks at the difference between these average traces:

$$\delta_D[j] = \frac{\sum_{n=1}^{N} MSB(D(Ci, K_p))T_i[j]}{\sum_{n=1}^{N} MSB(D(Ci, K_p))} - \frac{\sum_{n=1}^{N}(1 - MSB(D(Ci, K_p))T_i[j])}{\sum_{n=1}^{N}(1 - MSB(D(Ci, K_p)))}. \tag{3.2}$$

If K_g was wrong, these two sets are uncorrelated, the δ_D will be a slight noise near zero, and the differential trace becomes flat as the sample size increases. However, if K_g was right, the δ_D will

approach the effect of the target bit on the power consumption, which will spike in the zones where D is correlated to the processed values.

There are other analyses that usually are used to extract the information from the power traces, looking at subtle statistical correlations between the secret bits and power consumption, such as Correlative Power Analysis (CPA), Electromagnetic Analysis (EMA), and Template Attacks (TA).

Correlation Power Analysis

Instead of comparing partitions, the CPA attack undertakes the correlation factor between hypothetical power consumption values and the measured power consumption samples.

It stands by the Pearson Coefficient of Correlation and not on the Difference of Averages given the way of differentiating applied to the power traces set. Many advantages are counted, including the no need for a larger number of power traces [4]. There are four steps to a CPA attack:

1. Write down a model for the user's power consumption. Having the model, the next step is to look at one specific point in the encryption algorithm;

2. Make the user encrypt several different plaintexts. Register a path of the user's power consumption during each of these encryptions;

3. Try to discover subkeys, considering every possible option for the subkey. For each attempt, the power consumption will be discovered using the known plaintext and the guessed subkey based on the chosen model. Calculate the Pearson correlation coefficient between the modeled and actual power consumption. This is done for every data point in the traces. Decide which subkey guess has the best correlation with measured traces, by measuring the degree of correlation between two variables and assuming values between -1 and 1, where $\rho = 1$ means that there is a perfectly linear positive relationship between the variables, $\rho = -1$ means that there is a perfectly linear negative relationship, and $\rho = 0$ means that the two variables are linearly independent from each other, represented by this coefficient (ρ);

4. Put together the best subkey guesses to obtain the entire secret key by comparing the hypothetical power consumption values for each hypothesis with the measured power consumption samples at each time instant, using the following:

$$r_i, j = \frac{\sum_{i=0}^{N}(h_i - \overline{h}) * (W_j - \overline{W})}{\sqrt{\sum_{i=1}^{N}(h_i - \overline{h})^2 * (W_j - \overline{W})^2}}. \tag{3.3}$$

   The chosen value is the highest correlation coefficient (in absolute value) originated from using the correct sub-key. If the coefficients are too close to each other, then it is known that not enough power traces were collected in order to estimate the key correctly.

3.1.2 Electromagnetic Analysis

As electrical devices, the components of a computer often generate electromagnetic radiation as part of their operation. An attacker that can observe these emanations and can understand their causal relationship to the underlying computation and data may be able to infer a surprising amount of information about this computation and data [35]. It is possible to divide these Electromagnetic Analysis (EMA) attacks into two main categories:

- Simple Electromagnetic Analysis (SEMA);
- Differential Electromagnetic Analysis (DEMA).

These can be advantageous in cases where the power consumption side-channel is unavailable or when countermeasures to prevent its exploitation have been applied [2].

The collection of the signals is usually done by either applying a near field probe for the closest captures, or antennas for the furthest ones. In what concerns capturing isolated direct emanations, usually requires the use of tiny probes located really near to the signal source. Otherwise the capture of unintentional emanations is usually much easier, as those signals often have a better propagation, and even allowing captures to be performed at a distance. One of the main implementations of this method as to do with the military, all the military branches, once since the beginning of the long range communications that the electromagnetic emanations have been exploited and showed great potential and great results.

3.1.3 Template Attacks

Template Attacks were introduced in 2002 by Chari, Rao, and Rohatgi [36], and take a different approach to key extraction. These attacks are considered to be stronger than the others mentioned above in sections 3.1.1, 3.1.1, 3.1.1, and 3.1.2 in a theoretical information way, once every leaking information is potentially used. Frequently, these attacks require much fewer samples than correlation attacks [37]. An exact copy of the aimed device is necessary to produce a model of its noise leakage to create the templates for the attack. This is effortlessly achieved when the device in question is mass-produced and easily accessible. A typical template attack is then composed of two phases:

1. **Profiling Phase:** Also known as training phase, where the attacker uses his replica of the aimed device to model the leakage regarding different operations. Firstly a large number of power traces are gathered using the clone system, which is under complete control of the attacker's side to ensure that enough traces are recorded to give the information about each subkey value. Like so, this number is only restricted by the time and available storage. The traces correspond to the power consumption of the device while performing the operation that is being modeled, and the idea is to consider each operation as performing the encryption with some different bits, depending on the key and plaintext;
2. **Attack Phase:** The power traces obtained for the aimed device are processed and compared with the model constructed in the first phase to obtain the secret key.

3.1.4 Attacks based on energy measurement acquired by software

Until recently, power analysis attacks have had two limitations [24]:

1. Instead of aiming at the more complex and high-performance desktop and server CPUs, that kind of analysis aims at small embedded microcontrollers;
2. No software-based attack has been successfully applied, until now, on x86 to catch cryptographic key bits.

There are not many scientific studies regarding vulnerabilities introduced by Intel's RAPL and similar features, as this area is a relatively recent case of study. More recently, O'Flynn *et al.* [38] was able to reach secrets treated in the secure world on an ARM TrustZone-M platform using an onboard ADC, and Mantel *et al.* [39] could differentiate RSA keys by measuring the power consumption on Intel processors. Even more recently, Moritz Lipp *et al.* [24] showed that software-based power SCAs are very powerful against INTEL Software Guard Extensions (SGX), examining the Side-Channel leakage and being able to demonstrate the existent difference between instructions, the Hamming weight of operands and other data, recovering AES-NI and RSA keys in unprivileged and privileged attack scenarios.

Software Tools for CPU Energy Monitoring

There are several programs capable of measuring Intel's RAPL, which differ mainly on overheads and sampling frequencies as shown in [22].

Some of the considered programs were:

- **PAPI:** Gives a generic interface that not only reads the model-specific register (MSR) through the kernel module with the same name but reads from other performance registers as well. All of this comes together in a single, consistent and portable interface that carries out power and performance measures without being limited to a certain number of components [40];
- **Power Capping Framework/Powercap:** Enables the energy readings through the pseudo-file usually located at /sys/class/powercap/intel-rapl:0/energy_uj. Instead of what PAPI does, powercap is used only for energy-related measurements, and any user can read the file without root privileges [41];
- **Custom Framework:** Developed to reduce the overheads as much as possible. Instead of constantly opening and refreshing a file, like with powercap, a custom kernel module is used together with a driver to read those values directly. Furthermore, it may also provide the user with a better environment to run experiences by forcing each thread to a core, reducing the context switching, increasing the process's priority, and clearing the CPUs pipeline before starting [22];
- **IPPET:** Is a prototype power monitoring utility that uses Intel-specific energy MSRs to break down power consumption per process and displays them in real-time on a web browser [42]. IPPET uses the EnumProcesses API to get a snapshot of all running processes and then spread the

CPU power over non-idle processes running during the interval. IPPET can display the power data using a built-in web server by connecting their browser to localhost. For security reasons, remote connections to the localhost server are not allowed at this time.

3.1.5 Countermeasures and Counter-countermeasures

Because side-channel attacks are based on the relationship between leaked information through a side-channel and the secret data (ciphertext), two major types of countermeasures can be applied:

1. Eliminating or reducing the leakage of such information;
2. Eliminating the relationship between the leaked information and the secret data.

In other words, these countermeasures try to make the leaked information unrelated, or even uncorrelated, to the ciphertext, generally by doing something closest to a randomization of the ciphertext that changes the data to anything that can be undone after the cryptographic operation, i.e., the decryption. For each type of attack, there must be a countermeasure that better fits the situation.

Nowadays, there are many strategies, whether in hardware or software, being proposed to fight back SCAs, related to what is written above [2]:

- De-correlate the output traces on individual runs, e.g., by introducing random timing shifts and wait states, inserting dummy instructions, and randomization of the execution of operations;
- Replace critical assembler instructions with ones whose “consumption signature” is hard to analyze, or re-engineer the critical circuitry which performs arithmetic operations or memory transfers;
- Make algorithmic changes to the cryptographic primitives so that attacks are probably inefficient on the obtained implementation, e.g., masking data and key with random mask generated at each run.

As it is possible to tell, there are Software-based and Hardware-based countermeasures, although the first ones are related to algorithmic techniques that have better results, are versatile, very powerful, and probably the cheapest ones. Software-based countermeasures include introducing dummy instructions, randomizing the instruction execution sequence, balancing HWs of the internal data, and bit splitting. Hardware-based countermeasures include clock randomization, power consumption randomization or compensation, randomization of instruction set execution, and/or register usage.

However, various signal processing techniques can reduce the effect of these countermeasures. Despite all of these solutions and capabilities of these countermeasures to considerably reduce the frequency and performance of the SCAs, it is nevertheless complex to achieve secure implementation with as little extra cost as possible. A combination of hardware and software countermeasures would probably give an outstanding security/cost ratio.

To choose among all the countermeasures, it is essential to be aware of the value of the data and the attacker’s power. To measure the strength of the defense, there are three primary parameters:

1. The adversary’s power, which is measured by his resources and knowledge;

2. The attack's power, which depends on the quality of the tools and the strength state-of-the-art techniques;

3. The countermeasure's effectiveness.

3.2 Deep Learning Approaches

This work aims to improve the performance of deep neural network models based on side-channel attacks. This section will take a closer look at related publications on this subject over the last few years. Recent studies about supervised learning models including support vector machine (SVM) based side-channel attacks will also be reviewed in this section. Until today, multiple works have investigated the application of ML and DL techniques to defeat both unprotected and protected cryptographic implementations. Practical results on several datasets have demonstrated the ability of these attacks to perform successful key recoveries.

3.2.1 Support Vector Machine

Support Vector Machine is a supervised ML method created to provide a linear binary classification that aims to classify binary data using a specific hyperplane [43]. Moreover, authors in [44] and [45] claim that the SVM-based attack outperforms the template attack whenever highly noisy traces exist. Deriving from the previously mentioned method, authors in [46], distinguished power traces of an unmasked AES implementation using least square support vector machine (LS-SVM), where the number of classes is defined by computing the Hamming weight of the output of the S-Box. Also deriving from SVM, twin support vector machine (TSVM) [47], is learned by solving two smaller quadratic programming problems, however it has the capability to be approximately four times faster than the classical SVM [48].

3.2.2 Deep Learning Based Profiling SCAs

Analyzing the most recent trend in the ML area, the recent works have focused more on DL algorithms such as MLPs [49], [50] or CNNs [51]. Martinasek *et al.* have compared methods based on MLP against more classical SCA approaches such as Templates Attacks or Stochastic Attacks. The target is an unprotected implementation of the AES-128 algorithm running on a PIC 8-bit microcontroller.

These studies show that techniques coming from ML are worthy alternatives when compared with the original profiling attacks published in [52] and can often surpass them. It was additionally proved that ML techniques could be used with power traces that have large dimensions, while for the classical approaches it is difficult when the power traces have dimensions greater than 100 and, furthermore, they are robust to signal deformation. The counterpart to the great efficiency of these attacks is that they are difficult to parameterize, which has slowed down their deployment in the security evaluation industry. Moreover, while the papers present promising attack results, they do not give precise information about the parametrization of the algorithms, not even about their training. This is an important limitation which does not allow for the reproducibility of the analyses and hence hampers the development of deep

learning approaches in the embedded security community. More generally, a common framework to study and compare the effectiveness of Machine Learning methods against embedded implementations of cryptographic algorithms was missing. To surpass this limitation, a common dataset was created, which will be explain in more detail in section 3.2.4.

Based on the new common dataset, some experiments were done by Soroor Ghandali *et al.*[43] ending up with a concrete new architecture called Deep K-TSVM for profiled side-channel attacks which includes two main modules:

- Feature extraction
- Classification modules

This uses a deep learning architecture of Restricted Boltzmann Machine (RBM) to map datasets and also to learn the pattern of inputs, and it uses TSVM as the classifier. The deep neural network training was separated in two different stages:

1. Pre-training the parameters with RBM using the captured power traces from the copy version of the victim device;
2. Fine-tuning all parameters using gradient descent.

Soroor Ghandali *et al.*[43], claim that Deep K-TSVM performs better compared to the state-of-the-art profiling attacks such as MLP and CNN based side-channel attacks.

Ghandali *et al.* [43] created a new deep learning architecture called "Deep K-TSVM" consisting of In and Out and several hidden layers, in which the output of the last hidden layer is taken as the input to the TSVM to find two decision boundaries. It is assumed by the authors that the leakage model is based on the label that is achieved by calculating the HW of the output of the S-Box values at the first round which generates nine different classes, or achieved based on ID classification model (originating 256 classes) of the AES-128. The architecture of the proposed model is shown in Figure 3.3, All models predict a class by showing a score value, and the maximum of each prediction's score is computed to unify the class label. In this specific training procedure, 2^n classes were defined for the model, where n is the number of bits to create the ID classification model.

Also, Ryad Benadjila *et al.*[13] performed several tests using MLP and CNN as represented in Figure:3.4, in order to beat and improve the SCA. Their results show that in the context of SCA CNN are more effective than MLP even if they are more difficult to train.

Further in this work in section 3.2.5, both architectures will be compared and studied in order to classify them from the most effective to the less.

3.2.3 Performance Metrics

In the ML community, many evaluation frameworks are frequently applied either to evaluate the performances of a model or to choose the best suitable parameters to a parameterized group of models [13]. These methods also try to give an estimation of the performance of a metric, which does not

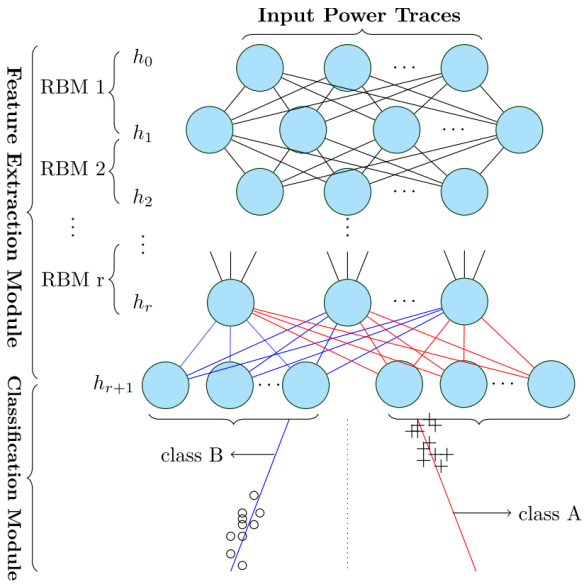

Figure 3.3: Architecture of the Deep K-TSVM

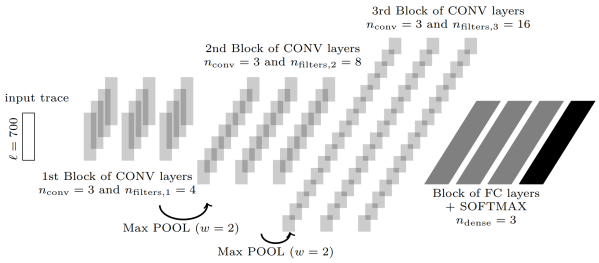

Figure 3.4: Architecture of the CNN

depend either on the content of the training set or even on the content of the test set, it only relies on their sizes. From the all existing methods, accuracy and loss function are the most common metrics to evaluate deep learning models:

- Accuracy is applied to measure a specific learning algorithm performance by computing the rate of correct classifications;
- Loss function is usually implemented to optimize a specific learning algorithm, and can demonstrate the progress of the learning model over several iterations.

The rank function, or key rank, is also a metric used in SCA for evaluating the performance of an attack. For all the input data, the rank function starts by giving a score to each key candidate (K) and ranking the score of all key candidates. Overall, the key rank is the index of the targeted key byte inside the list coming from an ID classification model as a list of outputs that sorts the order of elements in the output list. During the multi-trace attack, the attacker inputs many power traces into the trained model. Then, the rank function starts ranking the score of the real key among all the output participants, which in this study are 256 possible values per byte. When the rank function evaluates to zero, the real key is recovered and the attack succeeds. Nonetheless, these metrics are extremely dependent on the small changes in parameters and hyper parameters which control the learning process, which means that

weights, learning rates and layer-size will probably have to be adjusted to achieve a good performance model during the network training phase. In the works presented here, the performances of the models were evaluated with three different metrics: the rank function, the accuracy and the computational time.

3.2.4 Datasets

A dataset in ML is a group of data portions that can be analyzed by a computer as a single unit for analytical and prognostic purposes. The collected data should be homogeneous and perceptible for a machine that looks at data in a different way than humans do. To do so, it is essential to preprocess the data by cleaning and completing it, as well as giving the data appropriate labels that can be read by a computer as well as understood by humans. A decent dataset must fulfill some quality and quantity standards. For smooth and fast training, it is important to make sure that the dataset is relevant and well-balanced as well as updated with data, every time it is possible.

In this work scenario [13], it was necessary to find a way of creating a dataset that could retain the observations and the metadata (plaintext, ciphertext, key and mask values). An open platform that includes all sources of the target implementation was created. This open database is called ASCAD. The authors decided to use the version 5 of the Hierarchical Data Format (HDF5) [53] as the type of file to store the dataset which is a multi-purpose hierarchical container format that is able to store large numerical datasets and their meta-data. The development of HDF5 is done by the HDF Group, a non-profit corporation which has all the tools open sourced. An HDF5 file contains the datasets organized by groups and subgroups, which can be considered a file system within a file, where files are subgroups and folders are groups. Moreover, HDF5 also offers the capacity of a lossless compression of datasets in order to minimize the occupied space.

To start creating the new dataset, the authors used an already existing one called MNIST [54], that is well known in the ML image classification community, allowing any new Machine Learning algorithm to be justly classified based on the state-of-the-art results. The database is split in groups, each one containing data and labels:

1. The training dataset group, contains the samples used during the training phase. This group is created by the raw images in a file and their labels with the same index in another file;

2. The test dataset group is formed the same way as the previous one but with less samples, being created by the raw images in a file and their labels with the same index in another file;

Based on what was written above, Ryad Benadjila *et al.* [13] came up with a novel approach corresponding directly to the needs of testing ML algorithms in with SCA. It is organized inspired by the MNIST dataset and was extracted from the raw ATMega8515_raw_traces.h5 data file that had some issues when considering a classification problem, namely not being clear which dataset is to be used for training and for testing, the accuracy computation, and not having the correct labels for the SCA classification. Therefore, Ryad Benadjila *et al.* came up with ASCAD, its structure is presented on Figure 3.5 and is divided in two main groups:

- One for training (Profiling_traces) which contains N_p information;
- One for testing (Attack_traces) which contains N_a information.

Each main group, is composed by three HDF5 datasets;

- The traces dataset is holding the raw traces, which were cropped to save only the 700 samples window of interest, containing the relevant information;
- The labels dataset is holding the labels for each trace. Depending on the case, the value of the byte from S-box is chosen as the label of interest, leading to the 256 possible classes;
- The metadata dataset is holding the information related to each trace in a HDF5 structure, as explained before.

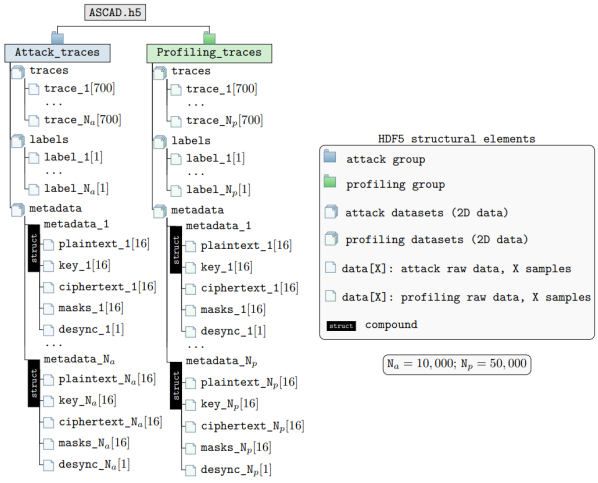

Figure 3.5: Architecture of the ASCAD dataset

In an exclusive ML perspective, this metadata is not crucial since the labels are the principal source of information to check the efficiency of an algorithm. However it is useful to maintain this data from an SCA perspective since the plaintext byte is necessary to extract the estimated key from the label values, and the real value of the key byte is useful for the key ranking with regard to each class probability. Even though only the plaintext byte and key byte chosen are useful for key ranking, normally all the other metadata is kept, the other plaintext and key bytes, the ciphertext and the masks, in order to exist completeness.

3.2.5 Previous deep learning Approaches Results

Having all the above been said it is now possible to compare the results obtain by [13] and [43], which have used the same dataset for the experiences.

The Deep K-TSVM [43] outperforms recent works, such as [55] which requires 15000 training traces in their MLP model and 20000 training traces in their CNN model to reach a key rank lower than 20.

This same model was compared with four other different models:

1. An MLP with and without PCA;
2. A 5-layer CNN;
3. A NeroSVM, which is made of two different but related modules called the feature extraction module, composed by several different MLPs, and the classification module, where the MLPs output layers are combined with a binary SVM. [56];
4. A CNN-SVM, which is similar to a 5-layer CNN but in the output layer a SVM classifier is used.

As said before, even though accuracy is not a common method for evaluating the performance of DL based side-channel attack techniques it was still used, in addition to the key rank average, to evaluate the robustness of the deep model to some previous work.

It can be concluded that the proposed model behaves better when applied to the dataset using the pre-training method RBM. The Deep K-TSVM outperforms the models referred before even with a single trace, since it has the lowest key rank as it is possible to see in Figures 3.6. The 5-layer CNN behaves better than the remaining models, more precisely, the 5-layer CNN requires about 200 power traces to reach a stable key guess while the CNN-SVM requires at least 400 of them for the same result. The MLP with PCA behaves worse than the MLP without PCA. This is because the PCA may remove some data components which are informative for linear classification representation, and thus it may negatively impact the accuracy of the non-linear profiling model of the MLP network.

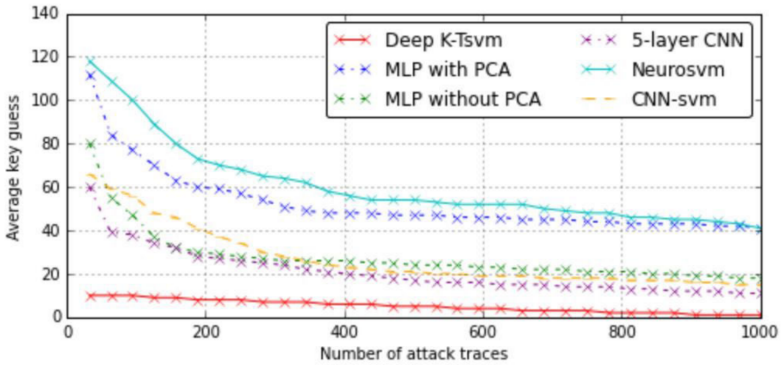

Figure 3.6: Comparison between results

In doing so, they showed that their model with a single power trace of the AES-128 is able to recover the secret key with a success rate of more than 99 $\%$ and a key rank of less than 10.

On the other hand, Benadjila *et al.* [13] were developing studies towards a synchronized and a desynchronized ASCAD dataset.

Focusing on the synchronized dataset, just like the previous study [43], MLP_{best} corresponds to the most efficient network the authors succeeded to train on ASCAD database. The MLP_{best} architecture on the SCA dataset was created with a labeling of the traces modified to take the Hamming weight of the value instead of the real value itself. This led to a model that is less complex than the model trained on the full values. Eventually, assuming that the deterministic part of the leakage corresponds to a Hamming weight may be an incorrect abstraction and induces error in the modeling. This model not only was compared with the CNN_{best} which will be described below but also with two more models:

1. An VGG-16, used for image recognition based on convolutional layers;

2. A Template attack;

As can be seen in figures the rank functions converge to 0 within 20 epochs and only 4 traces are required to determine the correct key. The authors were also able to get an accuracy of 0.028, without noticing any overfitting with 200 epochs, which can be noticed in Figure 3.7

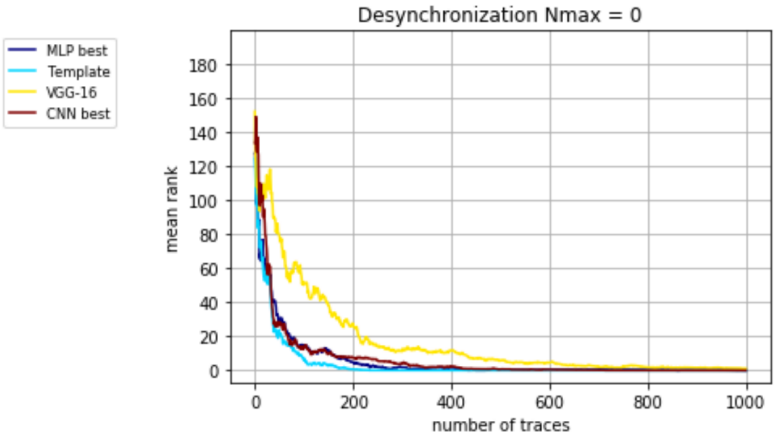

Figure 3.7: Results on a Synchronized dataset

Moving to the desynchronized dataset, several CNN configurations were trained and the most efficient one, was nominated as the CNN$_{best}$. This model not only was compared with the previous

This CNN$_{best}$ architecture has five blocks and one convolutional layer by block, a number of filters equal to 64, 128, 256, 512, 512, respectively with kernel size 11, ReLU activation functions and an average pooling layer for each block, with two final dense layers of 4096 units. CNN$_{best}$ was trained with a batch size equal to 200 by using RMSprop optimizer with an initial learning rate of 10^{-5}, making the other tested models on highly desynchronized traces outperformed and also achieving one of the best performances on small desynchronized traces with only 75 epochs, demonstrated in Figure 3.8.

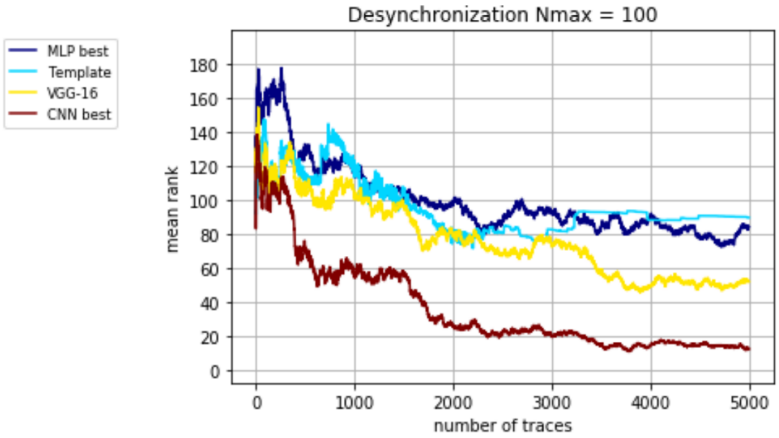

Figure 3.8: Results on a Desynchronized dataset

Chapter 4

Proposed Method

This chapter presents the methodology adopted to achieve the objectives of this MSc Thesis.This research supported by DL techniques aims at exploiting the correlation between the power consumption and the data processed in a CPU, in particular while performing cryptographic operations. Unlike most methods that use DL techniques for this type of analysis, this work proposes a software-based energy measuring approach, in an attempt to completely recover the secret key from the power consumption profiles.

Three central problems have been identified in previous works, namely:

1. The difficulty in detecting small leakages (which requires good acquisition) due to the randomization of a significant portion of the plaintext, which is a crucial step to be able to recover the complete key [22];

2. The difficulty of acquiring precise and differentiated energy and power measurements using only a software-based tool [24];

3. The classification performance, which while providing the capability of distinguishing some templates, does not allow for correct matching of all templates with a limited or feasible number of power traces and plaintext pairs [22].

Nonetheless, the use of Machine Learning techniques to aid in the classification of the obtained leaked information will be implemented, as an alternative to the standard statistical techniques. One of the main motivations to use ML techniques is the recent results in several areas of technology, and their promising performance applied to SCAs [12] with data collected via an oscilloscope.

The major contributor for a successful Power Analysis algorithm training and a subsequent classification with the objective of fully recovering a secret key in a realistic scenario is the presence of a large amount of leakages in the information acquired by the software.

To complete the primary objective of this work two main phases, represented in Figure 4.1, need to be outlined:

1. Collecting data through software;
2. Evaluating the data with DL models and performing the SCA.

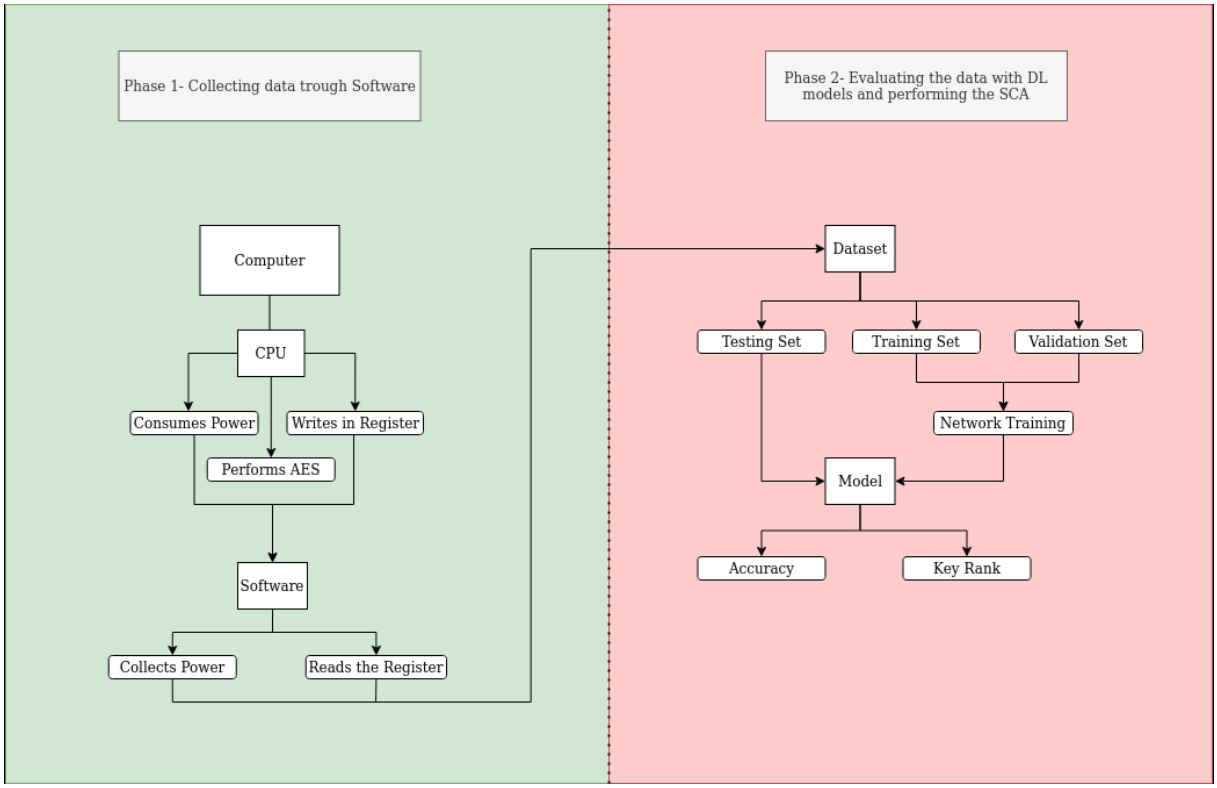

Figure 4.1: Conceptual architecture of a deep learning-based possible attack.

For the first main phase, based on previous works [22] and research in Section 3.1.4 that has already studied the performance of several power acquisition frameworks, it was decided that instead of using a specific framework for power acquisition, the RAPL feature that comes in every Intel processor will be used directly to read the custom kernel module (the MSRdrv), thus the need for an intermediary program, like PAPI [40] or powercap [41], is eliminated. This choice is expected to minimize the power consumed by the CPU, and decrease the time needed to perform the acquisitions.

For the second phase of this work, it was defined that besides the models that have already been used, new models will also be applied that, as shown in Chapter 2, might get promising results. The majority of previous DL-based SCAs used MLP networks and CNNs with good results, but with margin to improve. In this work we will try to successfully apply CNNs, providing them with the hypothetically necessary and suitable layers, and the best performing hyper parameters. To have a more complete study, based on all the characteristics already presented in Section 2.6.3, it is our objective to also implement a different type of neural networks, RNNs, specifically because it is a type of network that has shown a good ability to recognize patterns in temporally related data. Having the final results, we might be able to gather enough information to us allows to perform a comparison between implementations.

4.1 Data collection

The first step to validate this Thesis is to identify and define a leakage. It is considered a leaking operation when there is information coming from the power registers that can be used to differentiate between two equivalent operations using different data, which in this case corresponds to successfully recognizing different key/plaintext pairs during two separate encryptions of the same algorithm.

Because the AES is a complex algorithm, it was decided that firstly, a simplified version of this algorithm should be utilized to make sure that we are acquiring just the necessary leaking operations, by only studying the portion of the algorithm where the leakage is most directly tied to the data being encrypted, which is the operation of the S-Box load in the SubBytes. Other ways to further simplify the AES algorithm towards a more realistic scenario, have also been adopted:

- Instead of the normal 10 rounds, starting by only one round;
- Setting the same bytes for the whole key and using a plaintext with the same bytes for each of HW levels;
- Increasing the number of HW levels as the experiment evolves.

It is crucial to know some of the specifications of the RAPL feature, namely the sampling and reading rate and resolution. In our approach, there is also another important parameter, the noise produced by the operation of reading the registers and giving its values to the user, and other vital CPU functions, which might be relevant information for later post-processing the data in order to achieve better outcomes.

The bigger the amount of computational resources used to read the registers storing the power values, the lower the reading rate will be. If this reading rate gets lower than the sampling rate, the power value in the register might be updated, and the low reading rate might not allow the program to read the new value.

Hence, it is recommendable to reduce the computational resources utilized at the same time that reading the register, with the purpose of getting the best reading rate possible and the less skewed power consumption data. There are other important aspects that should be taken into account:

1. The memory allocation must be prevented while doing acquisitions, like so the obtained values should be stored in a static array;
2. The more heavy functions should only be executed at the end of the acquisitions.

In these standard power analysis attacks, an oscilloscope is commonly used, providing a much higher sampling rate, meaning that the power traces are acquired with a higher time resolution, offering the capability of examining and differentiating diverse moments of the algorithm, because each clock cycle might be represented by more than one power sample. Thus, there is the need for other approach than the ones applied in the standard power analysis attacks.

In our study, due to acquiring power samples only by using software, the number samples in each clock cycle will be significantly lower, thus each sample will be representative of several encryption

operations rather than each encryption being represented by several samples. Having the objective of being able to measure the power consumed at the cores, the domain under study is the PP0 [23]. The power measures are stored in the MSR PP0 ENERGY STATUS register, which will henceforth be simply referred to as “power register”. Even though in the majority of SCAs the leakage happens through the relative differences of the measures instead of their absolute value that comes directly from the raw readings of this power register, in this experiment the absolute value will be used.

4.2 Sampling Rate

The average sampling rate (F_S) has a very important role in this first part of the work, since in contrast to usual Power Analysis attacks the average sampling rate is much lower than the CPU's clock frequency. It is therefore relevant to carefully specify it according to the framework, to maximize the number of readings that can be made without reading irrelevant and noisy data. Another prime aspect to characterize is the average reading rate (F_R), which is the frequency at which the power register is read. This reflects the time and computational power that the interface spends while reading the power register, resulting in noise within the measures.

It should be noted that the only way to know when the power register is updated is to read it repeatedly and verify between readings if a different value is provided. So, to obtain the necessary rate values, a program was designed to acquire a large number of samples ($S = 1000000$), to guarantee a meaningful average calculation. Every time the power register is read the counter that saves the number of reads (R) is incremented, and each time the read value is different from the previous one and different from zero it is considered a correct sample, and thus its acquisition time is saved. This cycle is performed until the requested number of samples is achieved. The total time (t) needed to acquire all the samples is computed based on the saved time intervals.

Having the number of samples, reads, and total time, the average sampling and reading rates can finally be calculated, according to the following equations retrieved from [22]:

$$F_S = \frac{S}{t}, \quad F_R = \frac{R}{t}. \tag{4.1}$$

The Average Sampling and Reading Intervals are given by the following equations retrieved from [22]:

$$I_S = \frac{t}{S}, \quad I_R = \frac{t}{R}. \tag{4.2}$$

This algorithm and subsequent calculations will be performed in both available machines to determine which one has the most favorable average Sampling and Reading Rate for this experiment to continue successfully.

4.3 Processing the data

When it comes to evaluating the obtained data, it is possible to immediately discard the correlation attacks that focus on striking specific moments of the algorithm, since the time resolution for this method is not available. On the contrary, using NNs to attack is a possibility, with some adaptations to this specific problem.

The datasets are constructed by repeating an encryption a large number of times, with a key/plaintext pair that results in a specific pre-defined Hamming weight of the S-Box output at a certain key position, while acquiring the respective power consumption.

For each pair of key/plaintext that is acquired, a single main power trace is generated, that contains millions of samples. However, for NN training, it is best to have many examples of the same case, rather than one case characterized by many samples, thus the main power trace will be separated into several smaller power traces, which will be used for training and testing by the NN.

Following the previous definition of leakage, after the data acquisition strategy defined in Section 4.1, it is necessary to perform the following steps to conclude the analysis:

1. Process the data;
2. Create the dataset where the data is divided into three different parts:
   (a) The training data, D_{train}, which is used for training the NN models;
   (b) The validating data, D_{valid}, which is used for validating the NN models during the training process;
   (c) The testing data, D_{test}, which is used for evaluating the performance of the trained NN models;

   Each part contains the power traces (T) which are the power measurements and their corresponding key values (k), plaintext values (p), and labels (l), which express which Hamming weight was obtained at the S-Box output;
3. Train and evaluate the model.

There are three principal adjustments planned in order to get the most balanced, and less ambiguous dataset, namely:

- Eliminating all the captured power values that are zero, because the chosen Sampling Rate might not take into consideration the occasionally slower written values;
- Implementing the moving average algorithm to all the power values in order to smooth out eventual fluctuations and highlight possible existent cycles;

- Randomize the order of the power traces because in general the NNs perform better when the chances of a biased dataset are reduced.

4.4 Evaluating the data with DL models

The ultimate objective of this project is to create a successful attack using Deep Learning methods, which can only be achieved by successfully training and testing NNs. When attempting to solve a problem using DL methods, such as a SCA, it is important to correctly understand the problem formulation and its specifications so that the most suitable NN architecture is chosen for the resolution.

Given that we are striving to perform a successful SCA, the data to be processed is the CPU power consumption over a certain time interval. Taking this into account there are three different types of NNs that could theoretically achieve good results, that have been mentioned in Section 2.6.3:

- The CNN: Even though it is predominantly used to process visual input data (such as images and video), it is one of the most reliable types of NNs when it comes to recognizing patterns, either in spatial or temporal spaces. Different CNNs will be trained with 4, 5, 6, 7, and 8 convolutional layers, with 64 kernels per layer. We chose to use five different NNs with a different number of layers each to study which architecture is best for this problem, the number of layers are based on [13], where the best model achieved had five convolutional layers;

- The RNN: This type of NN is specialized in processing time dependent sequences of data, and oftentimes it provides better results than CNNs when processing data with sequential inputs. Different RNNs will be trained with 2, 3, and 4 fully connected RNN layers, with 100 units each. For this type of NNs the architectures have less layers because they are more complex and difficult to train, and there are less different architectures due to the higher training time needed for RNNs;

- The ConvLSTM: This type of NN is a merging of the two previous types. The cells in this type of NN combine the convolutions used in CNNs, and the LSTM architecture (a type of RNN which saves information about the previous points in a sequence to make better decisions about posterior points), and are mainly used to process video data, but can be used to process any data with time dependency, like the acquired power traces in this work. Different ConvLSTMs will be trained with 1, 2, and 3 ConvLSTM layers, with 256 kernels each. As this type of layer is also more complex these architectures were defined with fewer layers than the CNNs.

In this stage the three types of NNs previously mentioned will be trained and tested, with different configurations, to determine which type of architecture is better suited for this specific type of data.

Even though different types of NNs may be used, the goal is always to classify the targeted byte at the output of the S-Box operation. As such, when designing the NNs for SCAs involving AES with 128 bits, the classification can be done in two different ways:

- Using an identity model: In this case, the NN should have 256 different outputs which correspond to all the possible values a byte can have, and it would indicate in each output the probability of the byte for the analyzed power trace;

- Using the Hamming weight model: In this case, the NN should have 9 different outputs that correspond to the 9 different levels of Hamming weight a byte can take, and like wise, the NN would indicate the probability of each output, for a given power trace.

In this work, the Hamming weight model will be adopted, therefore the NNs that will be used will have 9 different nodes in the output layer. Although the identity model may give more specific results, it is more complex and difficult to train, and would require a different strategy for acquiring the data, since NNs should be trained with large amounts of data for each possible label.

After the NN architectures are chosen and defined, the NN has to be trained to recognize the different characteristics between encryptions that result in different Hamming weights, and the trained model is later used to classify an attack trace, which corresponds to the sampling performed in the machine under-attack while the cryptographic operations are occurring. For the training phase, the processed data (as mentioned in Section 4.3) is repeatedly submitted to the NN architecture, and after each epoch of training the NN weights are updated to reflect what it has learned. There are several hyper parameters that have to be defined for the training phase, namely the:

- Batch size: Number of examples, in this case number of power traces that are submitted at once to the NN;
- Epochs: The number of times the full training set is passed through the NN;
- Loss function: This function compares the output of the NN to the true output of the used examples and calculates how well the NN is evaluating the data. The goal is to minimize the result of the loss function, epoch after epoch;
- Optimization Algorithm (Optimizer): The strategy that is used to update the weights after each epoch, once the output of the loss function is known;
- Learning Rate: Defines the amount that the weights are updated after each epoch;
- Layers: How many layers a NN should have, and how many units should be in each layer.

The model is the organized group of weights that results from training a NN with certain hyper parameters. Even if an architecture is trained twice with the same parameters it generates two different models, which might not have the same performance, because weights are initially assigned randomly and the training data is shuffled for training. It is important to monitor the evolution of the loss function and the accuracy over the training and validation sets during training, to ensure that the model is not overfitting to the training data, which will likely lead to a poor performance on the testing set. It is relevant to note that NN training is substantial time consuming, so there may not be a chance to train all the proposed architectures.

After the training stage generates the final model, the testing stage occurs. It is necessary to classify the model's performance by choosing the most suitable metrics to the specific problem being analyzed. The testing of the model is done solely with data from the testing set, which is data that has not been presented to the model before. Therefore, it is possible to calculate the degree of effectiveness the

model can achieve for unseen data, which demonstrates how well the model can generalize and its usefulness. In this work we are going to use the following metrics:

- Key Rank: To classify the performance of an actual attack;
- Confusion Matrix: To classify the performance of the testing;
- Accuracy: To classify the total percentage of power traces that were correctly classified.
- Receiver Operating Characteristic (ROC) Curve: Classify the performance of the testing for binary classifications.

The confusion matrix is a square matrix that describes the performance of a classification model by displaying the number (or percentage) of examples that the model predicted as one class, and the true class of those examples. As such, in the cell of row *i* and column *j* of the matrix, is the percentage of examples that the model classified as class *i*, which had a true class of *j*. The number of rows and columns is the number of possible classes, that in this case represents the number of different possible Hamming weights for classification. Using the confusion matrix for performance control has three advantages:

- Easy to read representation that plainly shows which values the model is confusing;
- A single value reflects aspects of the classification associated with the model, either false positives and negatives or the precision and sensitivity;
- The relationship between the confusion matrix and the probability of finding the correct key according to the number of plaintexts can be formulated.

The ROC Curve is a graph that represents the performance of a classification model at all classification thresholds by plotting two different parameters:

- True Positive Rate (TPR),
- False Positive Rate (FPR).

Like so, the classifications that give curves with larger areas beneath the curve, indicate a better performance.

These metrics allow us to classify the models in terms of correctness, and to perform comparisons in order to establish which model is better at prediction for the different types of data.

All the generated models will be evaluated using the accuracy metric, and for the best performing models the confusion matrix will be computed for further analysis. The best model at evaluating the complete 9 levels of Hamming weights will also be used to finally perform the full SCA, and the key rank will be calculated.

Chapter 5

Experimental Setup

This Chapter will start by describing which of two possible machines was used for data acquisition, by comparing the Sampling Rates obtained in each machine using the algorithm defined in Section 4.2. Then, both simplifications of the AES algorithm that will be used are explained, and subsequently the experimental setup for data acquisition in both scenarios is detailed. Following the data acquisition, the choice of hyper parameters for the NNs is detailed and explained.

5.1 Calculating the Sampling Rate

After discovering the sampling rate and reading rate used for the acquisitions, it is then possible to give the correct parameter to the software that reads the power registers. Only after doing this important step is it possible to consider the values acquired as correct. But there are other variants which might cause problems to the measurements, namely the complexity of the used machine. In this case, we tried using two different machines:

1. A desktop with an Intel I7 10^{th} generation processor with 3.8 GHz clock frequency;
2. An older laptop with an Intel I7 6^{th} generation with 2.6 GHz clock frequency.

For each of these machines the sampling rate and reading rate were calculated, and from these results the machine to be used was chosen. Figures 5.1 and 5.2 show the distribution of the sampling intervals acquired in the desktop and the laptop, respectively.

Looking at Figure 5.1 it is possible to see many instances where the Sampling Interval was larger than 100 μs, which results in an average sampling interval of 141.4 μs, even though there seems to be a substantial number of sampling intervals smaller than 100 μs. As such, the average sampling rate acquired in the desktop machine was 7.1 KHz. In this desktop machine, the average reading interval that was obtained during this experiment was 7.9 μs, which results in an average reading rate of 125.8 KHz.

Analyzing the Figure 5.2, it can be seen that the sampling intervals in the laptop machine have a much lower distribution, and most of these intervals are smaller than 100 μs, which was not the case for

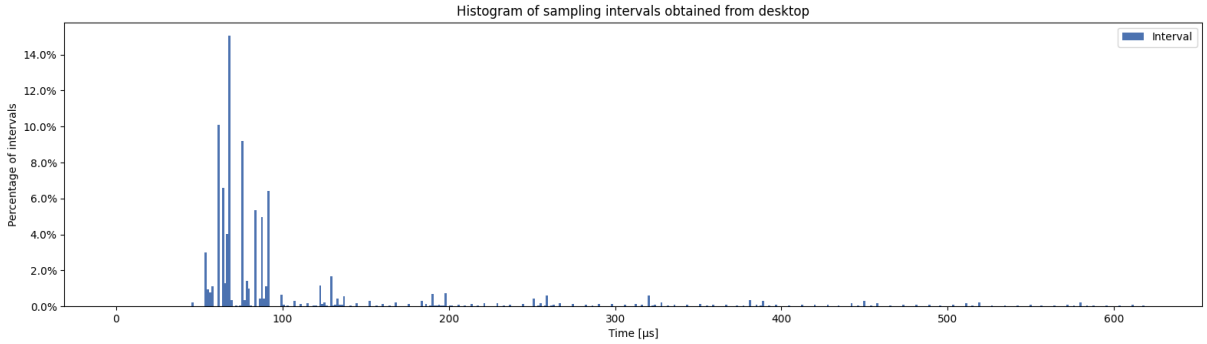

Figure 5.1: Sampling Interval Histogram acquired on the desktop. Average Sampling Rate of 7.1 KHz and Reading Rate of 125.8 KHz

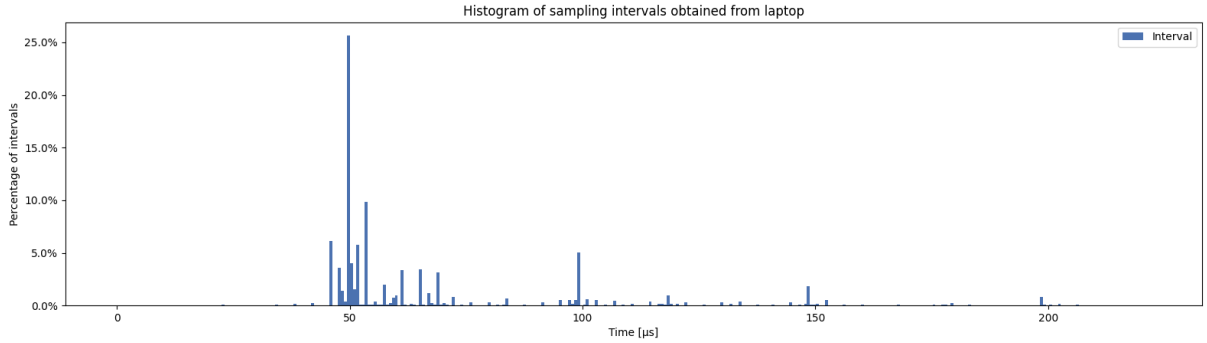

Figure 5.2: Sampling Interval Histogram acquired on the laptop. Average Sampling Rate of 16.7 KHz and Reading Rate of 365.0 KHz

the desktop machine. In the laptop machine the average sampling interval is 73.1 μs, which results in an average sampling rate of 16.7 KHz, and the average reading interval is 2.7 μs, which results in an average reading rate of 365.0 KHz.

In both machines the average reading rate is much faster than the average sampling rate, which indicates that it is possible to read the values inside the register more often than the register itself is updated with new, or valid, information.

Given the above data, we opted to continue this work and perform the acquisitions in the laptop machine due to its lower average sampling interval when compared to the desktop. These results might be justified by the fact that the laptop has an older less complex CPU architecture with fewer cores and lower operating frequency. This lower average sampling interval allows us to have a faster acquisition process, which leads to finer measurements, that can be the difference between a successful and an unsuccessful attack. Going forward, the sampling rate chosen to perform the acquisitions in the laptop was 100 μs, which is slightly higher than the calculated average sampling interval, but as can be observed in Figure 5.2, this sampling rate provides a larger safety margin since the majority of the values in Figure 5.2 are below this threshold.

5.2 AES simplifications

As this work will try to perform an SCA with DL methods and data acquired through software, it is necessary to start the experiments with a simplified scenario, that can subsequently evolve into a scenario that is more representative of real life conditions. As such, two different simplifications of the AES algorithm are proposed, a simpler one as a first experiment, described in Section 5.2.1, and a slightly more complex one to approximate our scenarios to real life cases, described in Section 5.2.2.

5.2.1 Simplified AES algorithm

As mentioned in Section 4.1 a simpler AES algorithm was created in order to increase our chances of successfully identify the most differentiating features of an AES operation along with software-based the power acquisitions. The use of a simplified AES provides the base work for trying to evaluate different DL methods and NN hyper parameters, with the objective of gradually increasing the complexity towards a real case scenario. It is structured in the following way:

1. To increase the leakage of the S-Box all the bytes of the plaintext and all the bytes of the key are the same and previously defined. If different values are given to different bytes within the plaintext or the key, the complexity can be gradually increased.

2. It performs only the first two operations of the first round of the AES, instead of the regular complete ten rounds, which provides the conditions to explore the leakage of the first S-Box without the following operations and rounds obscuring it. In order to increase the difficulty afterward, it is only necessary to complete the full first round, and then subsequently include more rounds.

3. As previously stated, the bytes will be represented with the Hamming weight model, since in this case they can only assume 9 different values. In the first case of the simplified AES that will be used, the Hamming weight possibilities will be reduced to two levels, 0 and 8, corresponding to the values of 0x00 and 0xFF as the output of the S-Box.

   To achieve this, the plaintext and the key are chosen by looking at the S-Box table, in order to produce the desired result, Figure 5.3 shows the key and plaintext combinations that produce this S-Box result. If the values of the plaintext and key are not previously defined, then the complexity increases, because of the new existent Hamming weights. The following experiments will gradually increase the possible values of Hamming weight available:

   (a) two levels (0 and 8);

   (b) three levels (0, 4, and 8);

   (c) five levels (0, 2, 4, 6, and 8);

   (d) nine levels (0, 1, 2, 3, 4, 5, 6, 7, and 8).

The obtained results with this simplification of the AES algorithm will be presented in Chapter 6.

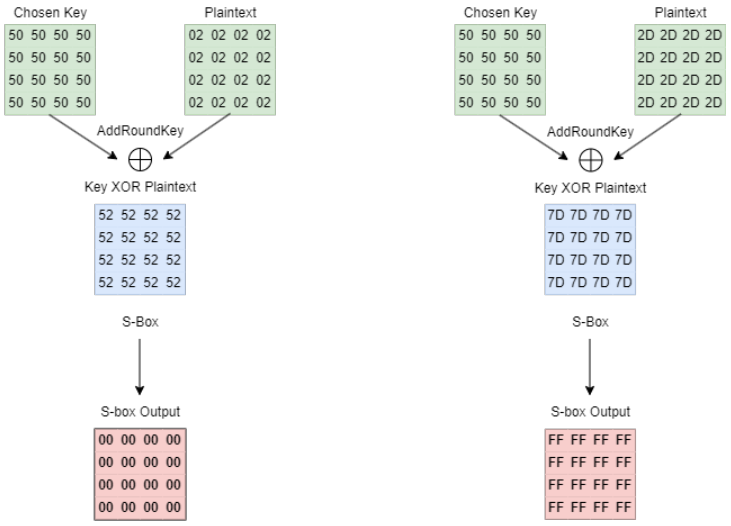

Figure 5.3: Key and plaintext combinations that produce the 0x00 (left) and 0xFF (right) outputs of the S-Box table.

5.2.2 One full round of AES

This particular simplification consists of a regular AES implementation except that instead of the regular encryption with ten rounds, this implementation will only have one full round. The Figure 5.4 represents all the operations included in the executed round.

The remaining characteristics of this AES implementation will follow the previous implementation in Section 5.2.1, as such all the bytes of the plaintext and all the bytes of the key will be same and the possible Hamming weight values available at the output of the S-Box operation will also be progressively incremented from two to nine levels.

The results achieved with this version of the AES algorithm will be presented and discussed in Chapter 7.

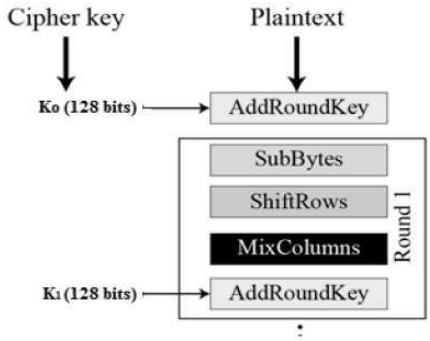

Figure 5.4: First round of a full AES implementation [57].

5.3 Collecting and Processing Data

Following the steps written above, we first started by running each implementation of the AES algorithm in order to obtain a suitable amount of power samples, so we could create a reliable dataset

for NN training. Generally when using NNs, the bigger the training dataset is, the better the results are going to be, because the NN gets to learn from many different examples of the same case, which will be beneficial when trying to classify new and unseen data. Controlling the measurement's environmental conditions, from the settings of the machine itself to the surroundings, is also critical to obtain less noisy and more consistent data, so that acquisitions at different times may produce similar data.

To acquire the data, the AES implementation is running on the computer while another program simultaneously reads the values from the power register and saves them in a static array. When the acquisition time is over, the values from the static array are saved to a file in the system. Later, the file is read, and the power values are extracted and processed.

For this dataset, we opted for separating the total number of acquired samples into blocks of a certain size, in order to create numerous smaller and distinct power traces for each acquired Hamming weight value, instead of a single large power trace for each Hamming weight value.

When choosing the most adequate number of samples that each smaller power trace should be comprised of, the paper [13] served as guidance for the initial tests, and as such, a size of 700 power samples per power trace was chosen. However, it was chosen to also test models trained with 900 samples per power trace because the complexity levels of our work are bigger than the one used as starting point. This way we will be able to evaluate the importance of the size of the power trace and analyze how the NN performance changes with a variation of the input size.

As described in Section 5.1, the laptop was chosen to perform the power acquisitions. As such, the program that uses the Rapl tool for power measurement was used simultaneously with the program that performs the simplified AES algorithm with the chosen keys and plaintexts. Therefore, the power consumed during the AES encryption is stored in an array in the power measurement program, and when the encryption ends, the power measurement program copies the millions of obtained values into the computer file system, to avoid costly operations during the power acquisition process.

After this file is created, the dataset can then be generated. To be able to produce a reliable dataset, there are a few important and necessary steps to take into account considering the acquired power consumption values:

1. Correcting all the null values that might have been read from the register;
2. Applying a moving average in order to eliminate eventual spikes and other disparities;
3. Separating the data in smaller power traces with 700 or 900 power samples per power trace;
4. Assigning a label to each power trace, the Hamming weight value at the output of the S-Box operation;
5. Making sure that all the labels have the same number of power traces in order to have a balanced dataset;
6. Randomizing the order of the created smaller power traces, in order to have get the most accurate results while training the NN;

7. Dividing the power traces into two groups, with 70% comprising the training and validation set, and 30% comprising the testing set;

After all these steps have been concluded, the dataset is created and ready to be used for training of the NNs.

5.4 Creating and Choosing the DL Model

The next step is the implementation of the NNs, so that we can assess if it is possible to obtain favorable results using DL methods, or if these methods are not appropriate for this type of work. To increase the chances of getting the best results, several different NNs architectures will be implemented so that we can compare the obtained results after training and testing with the same data.

Therefore, with every set of data that was acquired, three different types of NNs were used, with various different layers each, to ascertain which architecture could be more beneficial:

1. The CNN, trained with 4, 5, 6, 7, and 8 layers;
2. The RNN, trained with 2, 3, and 4 layers;
3. The ConvLSTM, trained with 1, 2, and 3 layers.

When a NN architecture is initialized for training, the weights that make up the NN are initialized randomly, or according to some specific pre-defined distribution. The training process consists of iteratively updating the values of these weights, so that the operations the NN performs each time have an outcome closer to the correct result. The collected data was initially divided into a training set (D_{train}) and a testing set (D_{test}), in our experiments the training set is composed of 70% of the acquired data, and the testing set has the remaining data. During the training process of the NNs architectures, the training set is used to teach to the NN the characteristics of the different types of power traces. For example, the NN should be able to learn the difference between a power trace that resulted in the value 0x00 as output of the S-Box, or the value 0xFF, as these two values should be linked to power traces with different characteristics. Even if the power traces only differ slightly between themselves, the NN should be able to find small and/or complex common patterns in the large amounts of data that are in the training set, which should allow it to correctly classify each power trace. During the training process, part of the training set, 10%, is used as validation (D_{val}), this set is not used for the adjustment of the weights, it is only during the training to validate that the NN is being tuned correctly, and generalizing well for unseen data.

During this training process there are various hyper parameters that can be modified, and each hyper parameter can have a bigger or smaller impact during training. Based on previous smaller quick tests and past research [13, 43], some hyper parameters were defined and fixed for all the experiments, since they were proven to obtain the best results on this type of data, and to maintain uniformity during the experiments.

As such, the batch size used during training was fixed to 50, this number could not be bigger because the space in the GPU used for training is limited, and a smaller batch size would result in slower

overall training. The number of training epochs was set to 500, but during all experiments the early stopping method was used, as such the training may stop after a number of pre-defined epochs have passed where there was no evolution in the acquired results for the validation set. Other important hyper parameters for training are the loss function and the optimizer, in this case, the chosen loss function was the Sparse Categorical Cross Entropy, as such, during the training process, the model will change and adjust the weights with the objective of minimizing the Cross Entropy obtained in the previous iteration results of the training data. It is necessary to use the Sparse Cross Entropy instead of a simple cross entropy because the processed dataset's labels are integers and not one-hot encoded, and it is necessary to use the Categorical Cross Entropy because in several datasets there are more than two categories of possible labels. Finally, another important hyper parameter to set is the learning rate, which determines the value by which the update of the weights is multiplied, thus a bigger value will lead to bigger changes, and a smaller value leads to smaller changes. Usually the learning rate starts out bigger and is reduced during training, so that initially the model can rapidly approach the possible solution, and as training progresses the model can approximate the best solution slowly to be more precise. During training, the metrics by which the model evaluates the validation set are also defined. In this case, the metric chosen was the accuracy, which indicates how many of the cases were correctly classified.

For the model evaluation, the metrics previously mentioned in Section 4.4 were used: key rank, confusion matrix, and accuracy.

Chapter 6

Simplified AES: Results and Analysis

In this Chapter the simplified AES will be implemented and the results for each different set of available Hamming weights will be analyzed. For each set of available HWs the acquired dataset is detailed and then the several previously defined NNs are trained and their accuracy results are analyzed. For the best model that can predict between nine different levels of Hamming weight the rank function will also be calculated to simulate an SCA.

6.1 Two levels of Hamming weight

For our first approach, described above in Section 5.2.1, the most vulnerable point of the AES implementation is the output of the S-Box at the first AES round. The power consumption captured at this point has the strongest correlation with the key. Knowing in advance the S-Box output table, it is possible to understand that in order to get the output values of 0x00 and 0xFF, it is necessary to get the values 0x52 and 0x70 from the previous calculation. In our case, we chose to establish for the key (k) the value 0x50 and vary the value for the plaintext (p) such that p $\oplus$ 0x50 = 0x00 and p $\oplus$ 0x50 = 0xFF, which results in 0x02 and 0xD2 respectively for the plaintext, to produce those values.

6.1.1 Dataset

Following the explanation in Section 5.3, for the two most distant levels of HW, we acquired in total 3502397 samples. With these samples, two different datasets were created, the first having 700 samples per power trace and the second having 900 samples per power trace, in each dataset the data was divided into two major groups of power traces, training and testing.

Figure 6.1 demonstrates a clear difference of the power consumed when loading the different values from the S-Box, with the loading of the bytes with value 0xFF consuming more than the bytes with value 0x00, as expected, since the CPU should spend more energy processing bits of value ‘1’, rather than bits of value ‘0’. With this acquisition method it is possible to create a reliable dataset, based on the power consumption, capable of training the NN properly in order to distinguish the operation of loading 0x00’s and the operation of loading 0xFF’s.

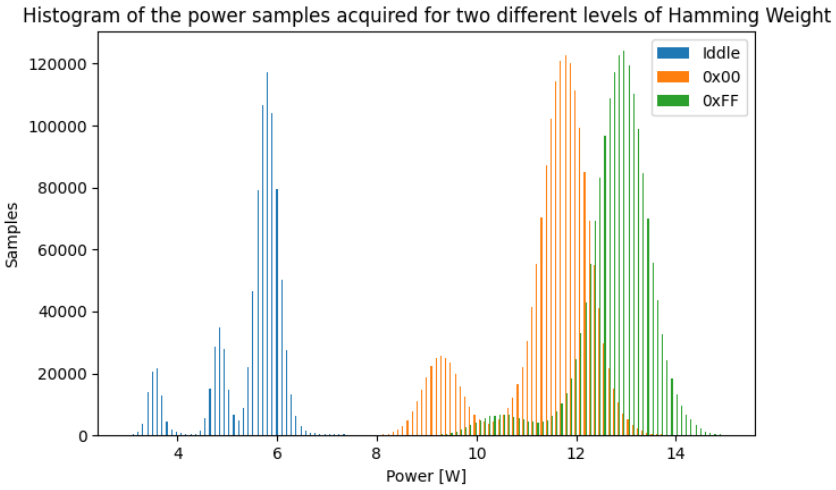

Figure 6.1: Power Histogram of Loading Idle vs 0x00s vs Loading 0xFFs.

6.1.2 Training and Results

The training part consists of implementing the NNs described in Section 5.4 with the purpose of achieving the best accuracy possible. Tables 6.1 and 6.2 show the accuracies obtained for each model that was trained.

For this specific case, the best accuracy resulted from implementing the four and seven layer CNN, that achieved 99.7% in accuracy on the testing dataset with 700 samples per power trace. The second best accuracy was 99.2% for the six layer CNN with 900 samples per power trace. For the RNNs trained with 700 and 900 samples per power trace, the best obtained results were 95.2% with two layers and 93.8% for four layers, respectively. For the ConvLSTM the value 98% was obtained with 3 layers for the dataset with 700 samples, and this same value was also acquired with 2 layers for the 900 samples per power trace dataset.

Table 6.1: Accuracy results for trained and tested CNN configurations for level two of HW.

Hamming Weight Levels	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
2	700	**99.7%**	99.1%	99.5%	**99.7%**	96.4%
	900	87.5%	99.0%	99.2%	98.4%	96.0%

Table 6.2: Accuracy results for trained and tested RNN and ConvLSTM configurations for level two of HW.

Hamming Weight Levels	Samples per Power Trace	RNN (layers)			ConvLSTM (layers)		
		2	3	4	1	2	3
2	700	95.2%	86.8%	73.7%	97.6%	97.2%	**98.0%**
	900	71.7%	81.1%	93.8%	97.8%	**98.0%**	97.5%

After all the different architectures that were tested, the best model ended up with an overall accuracy of 99.73%, with the 4 layers CNN architecture. As mentioned before the best model is further analyzed

with more metrics, in order to be possible to infer the quality and the precision of the training that was implemented, specially in this particular case where there are only two classes to classify, we are able to represent the results in two different and complementing ways, being possible to study not only the accuracy looking at the Confusion Matrix in Figure 6.2, but also the behavior of the trained model through the ROC Curve in Figure 6.3.

Looking at Figure 6.2, this accuracy can be analyzed more specifically:

- For the level zero of HW the model was able to correctly predict 99.73 % of the time, with only two misclassified cases;

- For the level eight of HW the model was able to correctly predict 99.73 % of the time, also with only two misclassified cases.

These results show that the model was capable of almost perfectly distinguish all the power traces, only missing 4 power traces in total of the testing dataset.

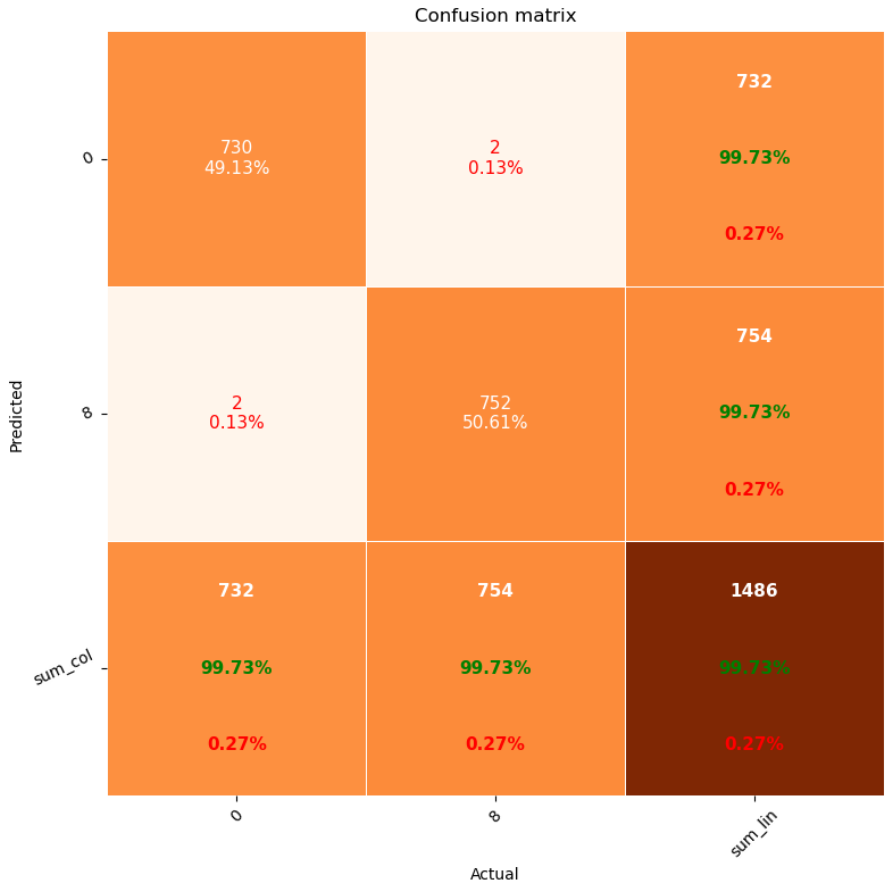

Figure 6.2: Confusion matrix for levels 0 and 8.

Looking at Figure 6.3, the ROC curve reflects the absolute confirmation of the success rate of this model, not only because the curve is standing on the true positive rate side but also because the area is equal to 1, which equates to the best possible performance.

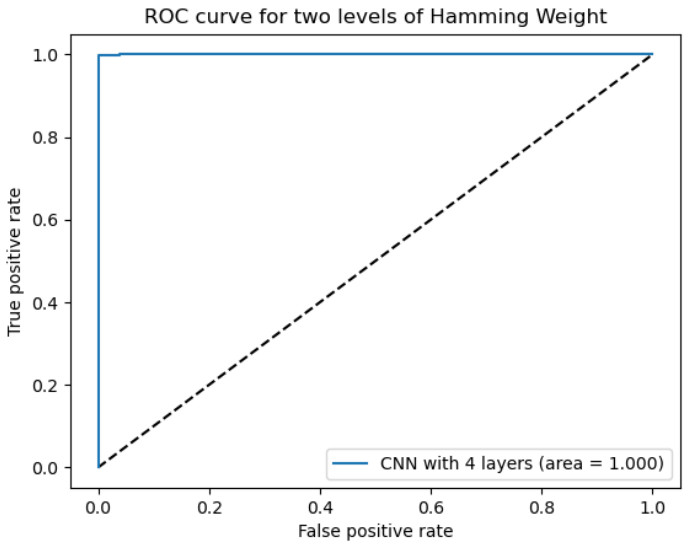

Figure 6.3: ROC curve for levels 0 and 8.

6.2 Three levels of Hamming weight

For our second experiment, instead of trying to classify only two classes, we tried to do it with three classes being the third class composed by half ones and half zeros, the Hamming weight level 4.

Using the same method as before, it is possible to understand that in order to get the output values of 0x00, 0x0F and 0xFF, it is necessary to get the values 0x52, 0xFB and 0x70 from the previous calculation. In this case, the key continues to have the value 0x50 and the plaintext will vary the value such that $p \oplus$ 0x50 = 0x00, $p \oplus$ 0x50 = 0x0F and $p \oplus$ 0x50 = 0xFF, which results in the values 0x02, 0xAB and 0x2D respectively for the plaintext, to produce those values.

6.2.1 Dataset

Following the explanation in 5.3, for the this experiment, we acquired 7811739 samples and divided them into to three major groups of power traces, one for each label, composed by smaller power traces having each one of them 700 and 900 samples, depending on the dataset to be created.

The Figure 6.4 clearly demonstrates an increasing of complexity, however a clear difference of the power spent when loading the different values from the S-Box is still observed, with the loading of the bytes with value 0xFF consuming more than the bytes with value 0x00, having the bytes 0x0F in between them.

6.2.2 Training and Results

As we are moving forward, the increase in complexity caused by the new added level implies more difficulty for our NN to succeed as well as it succeeded with only two different levels of HW.

All the described NNs in Section 5.4 were trained, and the obtained results are in Tables 6.3 and 6.4. For the CNNs with 700 samples the best accuracy value was 94.3% with six layers, for the 900 samples

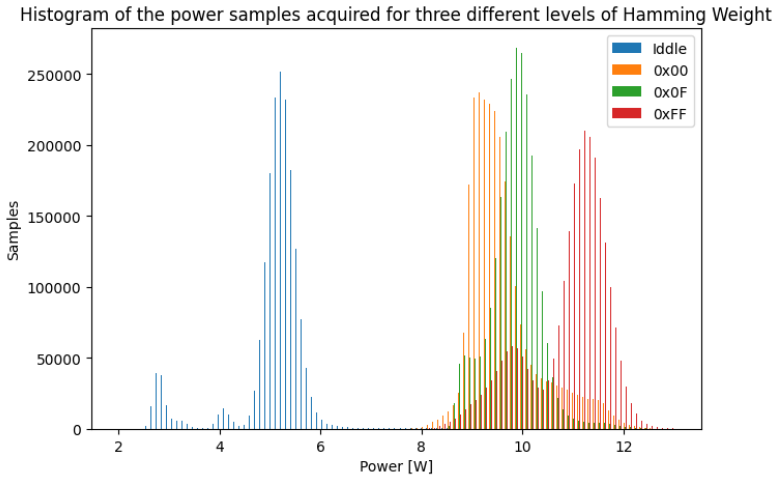

Figure 6.4: Power Histogram of idle computer behavior vs Loading 0x00s vs Loading 0x0F vs Loading 0xFFs.

per power trace, the best result was 94.1% accuracy with five layers. For the RNNs the best result was obtained using two layers with 700 samples per power trace and also with 900 samples per power trace, resulting in 79.2% and 61.1%, respectively. The achieved accuracy from the ConvLSTM models was 86.1% and 84.1%, the first one with 700 samples per power trace and two layers and the second on with 900 samples per power trace and three layers.

Table 6.3: Accuracy results for trained and tested CNN configurations for three levels of HW.

Hamming Weight Levels	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
3	700	90.3%	90.8%	**94.3%**	93.2%	70.6%
	900	93.6%	94.1%	93.6%	93.5%	71.0%

Table 6.4: Accuracy results for trained and tested RNN and ConvLSTM configurations for three levels of HW.

Hamming Weight Levels	Samples per Power Trace	RNN (layers)			ConvLSTM (layers)		
		2	3	4	1	2	3
3	700	79.2%	79.1%	79.0%	85.3%	**86.1%**	84.3%
	900	61.1%	58.2%	53.1%	82.7%	83.9%	84.1%

For the best model, the CNN architecture with 6 layers, the Confusion Matrix was calculated, and can be analyzed in Figure 6.5. It is possible to verify that the increased complexity has truly increased, since the best model ended up with an overall accuracy of 94.32%, more specifically:

- For the level zero of HW the model was able to correctly predict 96.18 % of the time. The wrong classifications were almost equally distributed between the levels four and eight of HW;
- For the level four of HW the model was able to correctly predict 94.97 % of the time, and when the model misclassified there is no special inclination for any of the other classes;

- For the level eight of HW the model was able to correctly predict 91.64 % of the time, making it the most misclassified level of this model. In this case, the model wrongly classified the power traces mostly as level four of HW.

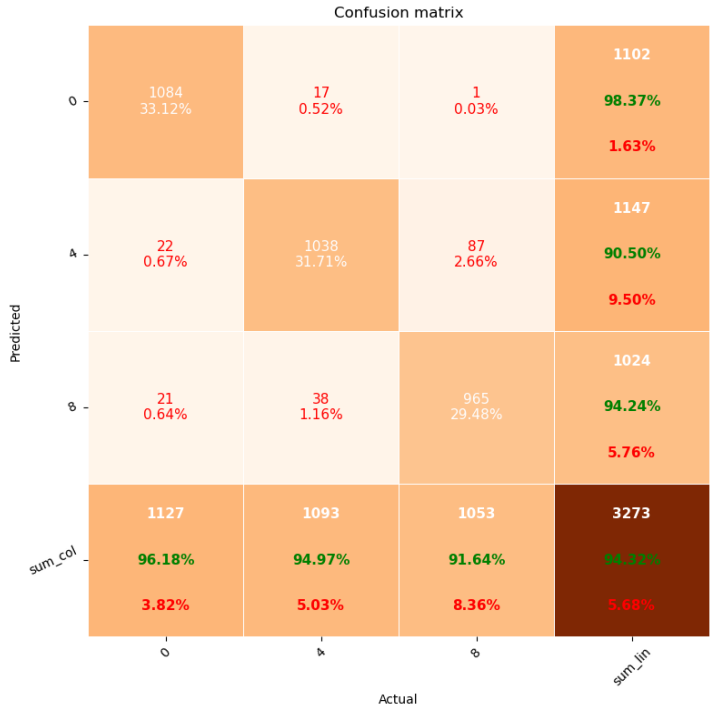

Figure 6.5: Confusion matrix obtained with the best model for levels 0, 4 and 8.

The biggest difficulty that our trained model had was to distinguish between the levels four and eight of HW, because for the level eight of HW the model misclassified 88 times, and 87 were wrongly associated with level four of HW, as well as for the level four of HW, where in the 55 times it was wrongly classified, 38 were associated with level eight of HW.

6.3 Five levels of Hamming weight

For our third experiment of increasing complexity, we tried to classify five different levels of HW, the previous three and two more. Using the same method as before, in order to get the output values of 0x00, 0x03, 0x0F, 0x3F and 0xFF, we have to obtain the values 0x52, 0xD5, 0xFB, 0x25 and 0x70 using the same method as in Section 5.2.1. The key remains with the same value of 0x50, and for the plaintext the value has to be changed such that $p \oplus$ 0x50 = 0x00, $p \oplus$ 0x50 = 0x03, $p \oplus$ 0x50 = 0x0F, $p \oplus$ 0x50 = 0x3F and $p \oplus$ 0x50 = 0xFF, which results in 0x02, 0x85, 0xAB, 0x75 and 0x2D, respectively, to produce those values.

6.3.1 Dataset

Following the explanation in 5.3, for this five levels experiment, we acquired 12736100 samples and also divided them in five major groups of power traces, one for each label. We created two different

datasets, one with with 700 samples and another with 900 samples per power trace.

6.3.2 Training and Results

In order to maintain an equal pattern between all the experiments, the applied models stayed exactly the same, except for the output layer, which reflects how many different levels there are, so that we could eventually see the differences in the results and if there is or not a clear pattern between them. Tables 6.5 and 6.6 show the accuracy obtained for all the trained models.

For the CNNs with 700 samples the most successful model was the one with seven layers resulting in 88.7% of accuracy, as for the 900 samples per power trace the best model was with six layers, reaching an accuracy of 87.3%. The best results acquired from the RNNs were 25.2% using 700 samples per power trace and 25.1% using 900 samples per power trace, both using two layers. The ConvLSTM models with the best performance were the two layers for 700 samples per power trace achieving 51.6% and the three layers for 900 samples per power trace achieving 48.2%.

Table 6.5: Accuracy results for trained and tested CNN configurations for five levels of HW.

Hamming Weight Levels	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
5	700	67.8%	85.2%	83.0%	**88.7%**	40.7%
	900	74.2%	74.0%	87.3%	86.5%	44.9%

Table 6.6: Accuracy results for trained and tested RNN and ConvLSTM configurations for five levels of HW.

Hamming Weight Levels	Samples per Power Trace	RNN (layers)			ConvLSTM (layers)		
		2	3	4	1	2	3
5	700	25.2%	31.2%	19.4%	19.8%	53.7%	54.5%
	900	25.1%	34.0%	19.9%	19.5%	19.5%	**56.0%**

Analyzing the results above, the best model was the CNN with 7 layers, trained with the 700 samples per power trace dataset. The Confusion Matrix of this model was calculated, and the result is shown in Figure 6.6, showing that as we add more HW levels the accuracy starts to decrease, ending up with an overall accuracy of 88.67%, more specifically:

- For the levels zero, two, and eight of HW the model was able to correctly predict 94.81%, 94.36%, and 94.64% of the times, respectively. The level zero and eight of HW are mostly mutually confused, when the prediction is wrong;
- For the level four of HW the model was able to correctly predict 76.86% of the time. When the model misclassifies level four of HW, it primarily confuses it with the immediate levels two and six of HW;
- For the level six of HW the model was able to correctly predict 86.55% of the time. The model predominantly confuses level six of HW as level four of HW.

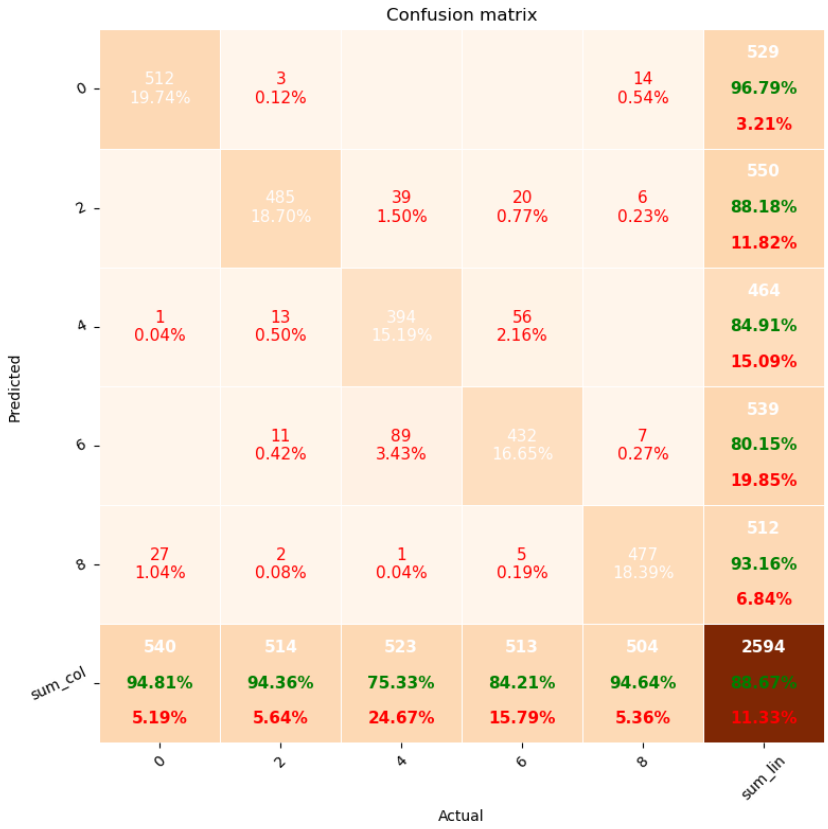

Figure 6.6: Confusion matrix for levels 0, 2, 4, 6 and 8.

By these results it is possible to tell that the majority of the errors occurred on the levels four and six of HW. Looking at level four of HW predictions, it is possible to understand the reason of the lower accuracy, once in 129 wrong predicted cases, 89 were classified as being level six of HW and looking at level six HW predictions in 121 wrong predicted cases, 56 were classified as being level four of HW. Which signifies that the model mostly confuses the levels four and six, and the remaining cases are mostly correctly classified.

6.4 Nine levels of Hamming weight

For our last experiment using the simplified AES, we tried to classify all the nine different classes, which translates to a large increase in complexity, and therefore difficulty. These nine classes of Hamming weight go from all the bits in a byte being zero, to all the bits being one.

Using the same method as before, it is possible to understand that in order to get the output values of 0x00, 0x01, 0x03, 0x07 0x0F, 0x1F, 0x3F, 0x7F and 0xFF, it is necessary to get the values 0x52, 0x09, 0xD5, 0x38, 0xFB, 0xCB, 0x25, 0x6B and 0x70 from the previous calculation. With the established value 0x50 for the key it is necessary to vary the value for the plaintext such that $p \oplus$ 0x50 = 0x00, $p \oplus$ 0x50 = 0x01, $p \oplus$ 0x50 = 0x03, $p \oplus$ 0x50 = 0x07, $p \oplus$ 0x50 = 0x0F, $p \oplus$ 0x50 = 0x1F, $p \oplus$ 0x50 = 0x3F, $p \oplus$ 0x50 = 0x7F and $p \oplus$ 0x50 = 0xFF, which results in 0x02, 0x59, 0x85, 0x68, 0xAB, 0x9B, 0x75, 0x3B and 0x2D respectively, to produce those values.

6.4.1 Dataset

With this experiment, we achieved the maximum complexity possible using the simplified AES, considering all possible cases of HW, nine levels, acquiring 11090750 samples and dividing them in nine major groups of power traces, one for each label, where again, each group of bigger power traces has smaller individual power traces having 700 and 900 samples each.

With this acquisition method it is still possible to create a reliable dataset, based on the power consumption, capable of serving really well the training of the NN in order to distinguish all the different operations.

6.4.2 Training and Results

In this last experiment using the simplified AES, the most difficult challenge for the NNs is achieved, the number of different levels of HW to classify plus the amount of information that is generated gets to a point where it could be extremely difficult to find distinguishing characteristics for each single level of HW. After training the CNNs, the RNNs, and the ConvLSTMs with both datasets, the results can be observed in Tables 6.7 and 6.8. The best results for 700 samples per power trace with the CNN were 72.3% with six and seven layers, and with 900 samples per power trace the best result was 81.1% for seven layers. For the RNNs with 700 samples the result was 11% of accuracy in the testing dataset, either with two or four layers, and with 900 samples the best obtained result was 22.8% with two layers. Finally, the applied ConvLSTMs accomplished an accuracy of 51.6% with two layers for 700 samples and 48.2% with three layers for 900 samples.

Table 6.7: Accuracy results for trained and tested CNN configurations for nine levels of HW.

Hamming Weight Levels	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
9	700	8.6%	71.5%	72.3%	72.3%	11.0%
	900	12.6%	59.9%	67.1%	**81.1%**	16.0%

Table 6.8: Accuracy results for trained and tested RNN and ConvLSTM configurations for nine levels of HW.

Hamming Weight Levels	Samples per Power Trace	RNN (layers)			ConvLSTM (layers)		
		2	3	4	1	2	3
9	700	11.0%	10.9%	11.0%	11.2%	**51.6%**	11.0%
	900	22.8%	17.6%	10.7%	10%	10%	48.2%

For further analysis, the confusion matrix of the best performing model, the CNN with 7 layers with an accuracy of 81.10$\%$ was calculated, and is displayed Figure 6.7. The confusion matrix demonstrates that this final increase in HW levels to incorporate all possible HW levels of a byte had an influence in the obtained overall accuracy. More specifically:

- For the level one of HW the model was able to correctly predict 98.10 $\%$ of the times, meaning that in 420 power traces the model correctly classified 412 of them, which was the class with the best

accuracy result.

- For the level two and three of HW the model was able to correctly predict 60.77 % and 64.29% of the times. It resulted in the worst obtained results. The main cause for the poor predictions on level two of HW was the improper classification of 82 power traces as level one of HW, and 65 values as level zero of HW. When analyzing the predictions of level three it is clear that it went wrong by misclassifying this level as all the HW levels lower than itself, with 164 bad predictions in a total of 170.

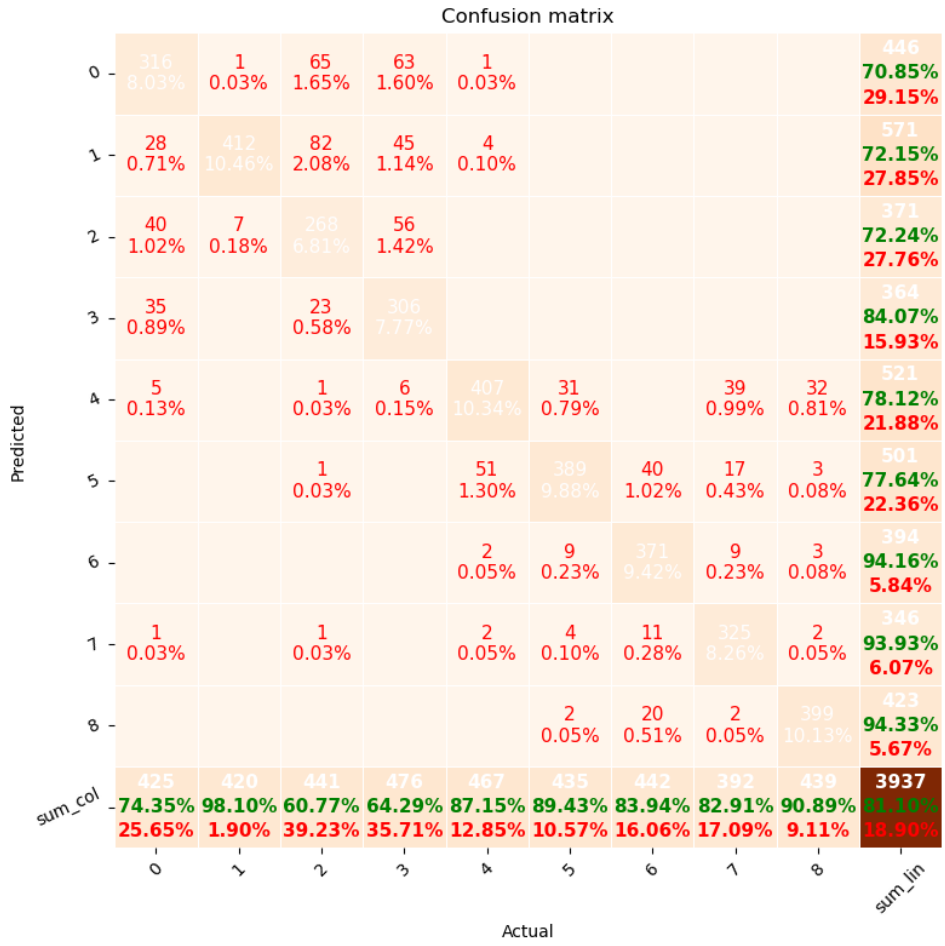

Figure 6.7: Confusion matrix for levels 0, 1, 2, 3, 4, 5, 6, 7 and 8 obtained with 900 samples per power trace.

Even though the obtained results are not as good as expected, the general division per classes made by the model was quite good. In Figure 6.7 it is possible to see that when the model is classifying power traces that have bigger levels of associated HW, it does not ever predict a value from the smaller levels of HW. Almost the same happens when predicting the smaller levels of the HW, the model only predicts three values corresponding to the bigger levels of HW. So, generally, the model mostly misclassifies power traces with HW values close to the real value.

6.5 Key Rank

As previously defined, the key rank Function is the last metric to be used in this work to determine the effectiveness of this Deep Learning approach to SCAs.

In this Chapter several models have been trained for different levels of available Hamming weights in a byte, with increasing complexity, but for this metric the model for nine levels of Hamming weight will be used, as this is the only one that allows the recovery of all possible 256 byte values. Therefore, the objective is to recover any byte of the secret key, since in the data collected, all bytes have the same value.

The rank function was implemented and tested with the best performing model of Section 6.4, the CNN model with seven layers trained with power traces of 900 samples each. Figure 6.8 shows the evolution of the mean rank using this model, given successive power traces for classification. As it can be observed, the model was able to achieve a mean rank equal to zero after only 6 traces, which means that with simply 5400 samples of the testing dataset this model was able to completely recover one byte of the secret key. Additionally, in this experiment the model only needed, on average, 3 power traces to recover the encryption byte.

This model can be used to discover any secret key byte through the power consumption of a CPU acquired in the same conditions as the training dataset.

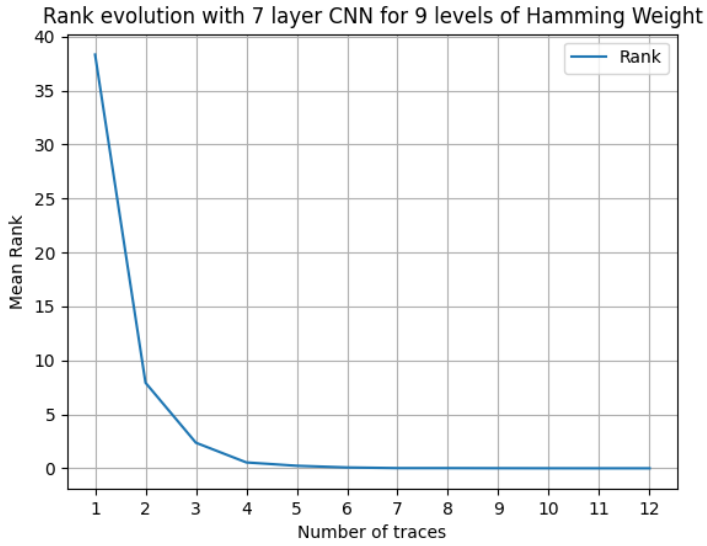

Figure 6.8: Evolution of the mean rank of the attack performed with the best CNN model trained in Section 6.4.

6.6 Results and Analysis

Supported by the tables below, Tables 6.9 and 6.10, we were able to understand if there were any patterns, similar results, extreme variations or even if it would be worthy to proceed with the attack itself. All the gathered information was gathered and displayed into two different tables:

- The CNN results (Table 6.9), where all the information regarding the accuracy of the trained CNN models is organized;
- The RNNs and ConvLSTM (Table 6.10), where all the information regarding the accuracy of the RNN and ConvLSTM trained models is organized.

The first models to be trained and tested were the CNN models. From the less complex dataset with only two different levels of HW that was tested, to the most complex with nine different levels of HW, all were submitted to several CNN models composed by four, five, six, seven, and eight layers. Maintaining what was explained before, the idea always was to have credible information to establish a comparison between all HW levels, to understand if there is a correlation in results.

The overall obtained results shown in Table 6.9 show good accuracies and did not evolve in a completely uniform way, but it is possible to see that as the complexity increased, the models with fewer number of layers started to have poor results. At the same time, when the complexity was lower and a model with more layers was used, the accuracy also suffered.

Having a closer look of the data, we see that the CNN with four layers had a great performance when training and evaluating the simpler dataset, achieving the best accuracy result for the two different levels of HW. We can also observe that the seven layers CNN was able to achieve the same result, with both results being reached using 700 samples per power trace. For the three different levels of HW the CNN with best accuracy was the one with six layers and 700 samples per power trace. For the five and nine different levels of HW the model with the best accuracy results was the seven layer CNN, using 700 samples for the five different levels and 900 samples for the nine different levels.

Table 6.9: Accuracy results for different trained and tested CNN configurations.

Hamming Weight Levels	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
2	700	**99.7%**	99.1%	99.5%	**99.7%**	96.4%
	900	87.5%	99.0%	99.2%	98.4%	96.0%
3	700	90.3%	90.8%	**94.3%**	93.2%	70.6%
	900	93.6%	94.1%	93.6%	93.5%	71.0%
5	700	67.8%	85.2%	83.0%	**88.7%**	40.7%
	900	74.2%	74.0%	87.3%	86.5%	44.9%
9	700	8.6%	71.5%	72.3%	72.3%	11.0%
	900	12.6%	59.9%	67.1%	**81.1%**	16.0%

One aspect that is immediately evident is that all the results using five and eight layers were never the best results. In fact, the majority of the calculated accuracies seem to improve from using four layers to five layers, and then all accuracies are worse with eight layers rather than seven layers. This shows that for the more complex datasets, CNNs with fewer layers cannot handle the complexity and the small differences that exist between power traces of different HW values. However, if the CNN complexity is increased too much (as in the eight layer CNN), the created model might be prone to overfitting to the training data, because of all the available training parameters. As such, the more complex models all show poor results in the testing datasets. All the best accuracy results are concentrated between six and seven layers, being the seven layer architecture the one with consistently better results. Overall, the

models with fewer layers cannot deal with the more elaborate datasets, and the model with more layers adapts too much to the training set, making it ineffective in the training stage, or the attack stage.

When talking about the number of samples per single power trace, the best results were achieved when using 700 samples, except for the ninth level for HW where 900 samples resulted in the best accuracy possible. For the difference in using 700 samples or 900 samples per power trace, it seems the best results are almost always with the 700 samples dataset, but looking at individual cases there is no clear pattern that allows us to conclude one dataset is much better than the other.

Moving on to the other type of NNs also implemented in this work, the chosen ones were based on the fact that they are generally better at processing time series because of their capability to remember previous data points in a sequence and make decisions regarding these previous states. Theoretically, it would make sense to implement them, once we are dealing with cyclic operations that time to time repeat themselves outputting the same values all the time, however they are oftentimes very difficult to train, due the their architecture not being strictly feed-forward which makes their signal movements extremely complex and makes it hard for the optimization algorithm to update the weights, which results in poor results. That is perfectly demonstrated in Table 6.10, where it is possible to see that no accuracy value obtained with an RNN was able to outperform any of the CNNs accuracies.

On the other hand the results obtained from the ConvLSTMs for the two different levels of HW was similar to the CNN's, but for the other datasets the accuracy results were quite lower than the ones obtained by the CNN models.

Table 6.10: Accuracy results for different trained and tested RNN and ConvLSTM configurations.

Hamming Weight Levels	Samples per Power Trace	RNN (layers)			ConvLSTM (layers)		
		2	3	4	1	2	3
2	700	95.2%	86.8%	73.7%	97.6%	97.2%	**98.0%**
	900	71.7%	81.1%	93.8%	97.8%	**98.0%**	97.5%
3	700	79.2%	79.1%	79.0%	85.3%	**86.1%**	84.3%
	900	61.1%	58.2%	53.1%	82.7%	83.9%	84.1%
5	700	25.2%	31.2%	19.4%	19.8%	53.7%	54.5%
	900	25.1%	34.0%	19.9%	19.5%	19.5%	**56.0%**
9	700	11.0%	10.9%	11.0%	11.2%	**51.6%**	11.0%
	900	22.8%	17.6%	10.7%	10%	10%	48.2%

With the model obtained for the nine different levels of HW, the performed SCA was remarkably successful, taking five power traces to discover the key byte, which in this case reveals the whole key, since all bytes are the same. This is in line with the desired result for this experiment, since the full byte was recovered.

Chapter 7

One Full Round of AES: Results and Analysis

The ultimate purpose of this work is to get closer to the real case scenario, where all the 10 rounds of the AES algorithm would be implemented for encryption, as such, in this Chapter a complete round of the AES algorithm will be used, therefore increasing the complexity of the analysis. Similarly to Chapter 6 we will start by implementing the AES algorithm for two, three, five and finally nine different levels of HW and from that point, design an attack to recover the secret key used for the AES encryption.

However, taking into consideration the results of that previous Chapter 6, instead of using the initial three predefined NNs, only the CNNs and ConvLSMTs will be used, due to the poor results acquired by the RNNs.

7.1 Two levels of Hamming weight

For just two different levels of HW, as shown in Figure 7.1, even with the implementation of one entire round of AES, it is possible to perfectly visually distinguish the values 0x00 and 0xFF coming from the S-box.

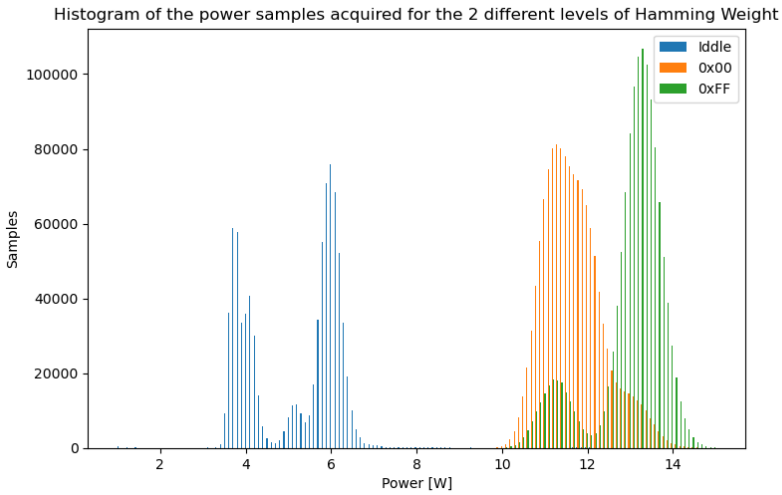

Figure 7.1: Power Histogram of Loading Idle vs 0x00s vs Loading 0xFFs

7.1.1 Training and Results

As already explained, the implemented NN models were the CNNs and ConvLSTMs maintaining the architecture of each one. The CNNs were able to achieve 88.6% using four layers with 700 samples per power trace and 88.1% using five layers with 900 samples, while the ConvLSTMs obtained 88.3% and 87.4% using two layers, the first one with 700 samples and the second one with 900 samples per power trace. Tables 7.1 and 7.2 show the accuracy obtained for all the trained models.

Table 7.1: Accuracy results for trained and tested CNN configurations for level two of HW.

Hamming Weight	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
2	700	**88.6%**	88.5%	88.3%	88.1%	87.7%
	900	87.5%	88.1%	87.5%	86.0%	87.1%

Table 7.2: Accuracy results for trained and tested ConvLSTM configurations for level two of HW.

Hamming Weight	Samples per Power Trace	ConvLSTM (layers)		
		1	2	3
2	700	87.6%	**88.3%**	73.7%
	900	87.3%	87.4%	87.0%

In this specific case, it is again possible to analyze the results in two different and complementing ways, the Confusion Matrix and the ROC curve, represented in Figure 7.2 and Figure 7.3. To do so, we selected the accuracy value with the bigger percentage because it is the one with more probability of success, which in this case belongs to the CNN models that uses four layers with 700 samples per power trace. Looking with more detail at Figure 7.2, it is possible to notice that:

- For the level zero of HW the model was able to correctly predict 92.53% of the time, only getting wrong the classification of 41 power traces of this level;

- For the level eight of HW was able to correctly predict 84.76% of the time. For this level, the model misclassified 87 examples, mora than for level zero of HW.

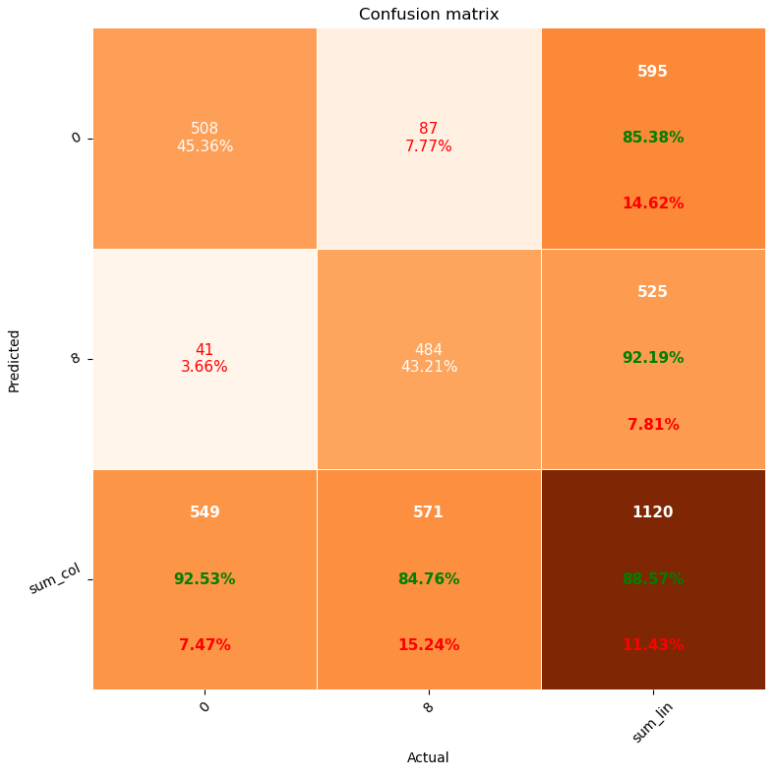

Figure 7.2: Confusion matrix for levels 0 and 8.

The aspect that is possible to retrieve from this Figure 7.2 is that the predictions for the eighth level of HW were wrong more than twice the times of the predictions of the level zero of HW. Comparatively to the results achieved in the first part of this work, the NN model accuracy was quite lower, with a difference of 11.2%.

Looking at Figure 7.3, the ROC curve reflects the problems caused by the increased complexity to the success rate of the NN model training, not only because the curve is now standing much more towards the center of the true positive rate side but also because of the area that previously was equal to 1 and now decreased to 0.886 meaning a good performance but not the best possible.

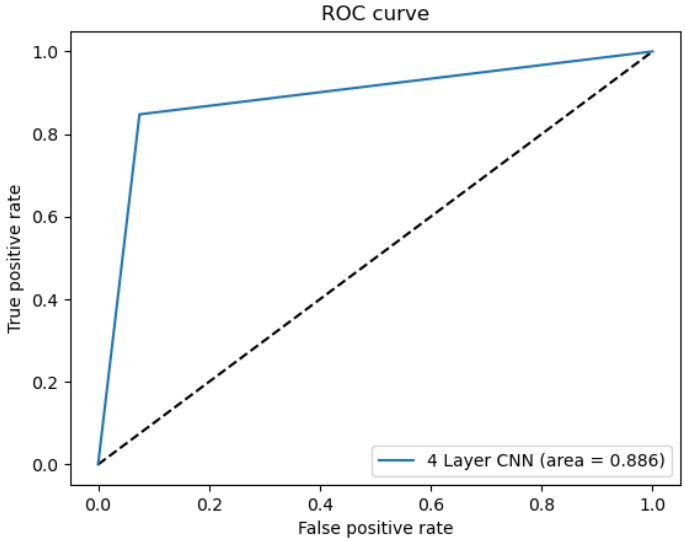

Figure 7.3: ROC curve for levels 0 and 8.

7.2 Three levels of Hamming weight

The best accuracy values achieved by the CNNs were 66.2% when using seven layers and 700 samples and 60.0% when using six layers and 900 samples, while for the ConvLSTMs the best accomplished results were 68.10% using two layers and 700 samples and 68.37% using three layers and 900 samples. Tables 7.3 and 7.4 show the accuracy obtained for all the trained models. After analyzing this results, the experiments will proceed with the accuracy value provided by the three layers ConvLSTM with 900 samples, in this case, 68.37%.

Table 7.3: Accuracy results for trained and tested CNN configurations for three levels of HW.

Hamming Weight	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
3	700	60.9%	56.7%	29.6%	**66.2%**	65.7%
	900	53.0%	54.0%	60.0%	57.0%	32.5%

Table 7.4: Accuracy results for trained and tested ConvLSTM configurations for three levels of HW.

Hamming Weight	Samples per Power Trace	ConvLSTM (layers)		
		1	2	3
3	700	32.5%	65.1%	68.1%
	900	33.2%	33.6%	**68.4%**

7.2.1 Training and Results

By analyzing the Figure 7.4 reflecting the three layers ConvLSTM with 900 samples, the first aspect that is impossible to ignore is the decay of accuracy comparing to the level two of HW, more specifically:

- For the level zero of HW the model was able to correctly predict 42.01% of the time. However, in 738 power traces the model classified 310 as level zero and 370 as level four, so the model is particularly poor at classifying power traces with level zero of HW, assuming more times that they belong to class four of HW;

- For the level four of HW the model was able to correctly predict 80.86% of the time. The class that the model most confuses as class four is class zero, as in 742 power traces of HW four, the model classified 133 of them as HW zero, and only 9 of them as HW eight;

- For the level eight of HW the model was able to correctly predict 82.63% of the time. In 714 power traces of HW eight, the model classifies 47 of them with HW zero and 77 of them with HW four, so this model almost equally confuses power traces of HW eight with either of the other classes, when they are misclassified.

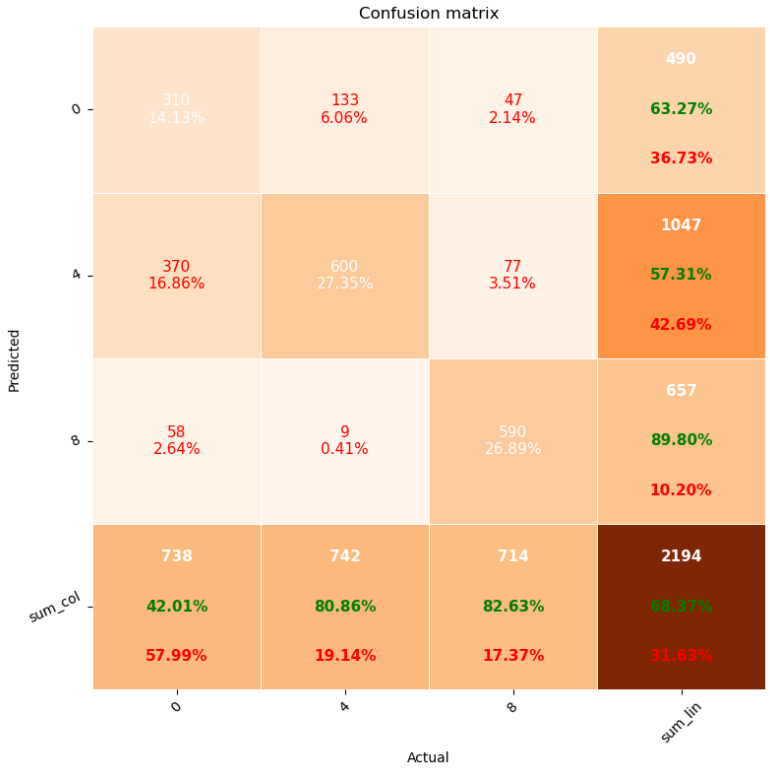

Figure 7.4: Confusion matrix for levels 0, 4 and 8.

In this case, the main adversity that our model found was the lower capability of distinguish the values of the level zero of HW with the ones from level four of HW. This model mostly confuses power traces of HW zero, with power traces of HW four, and is mostly good at classifying power traces of HW four and eight.

7.3 Five levels of Hamming weight

The most worthy accuracy values that the CNN models were capable of acquiring were 27.2% applying four layers to 700 samples and 31.4% applying seven layers to 900 samples. The ConvLSTMs, obtained 19.1% of accuracy using three layers for 700 samples and 20.4% using two layers for 900 samples. The tables 7.5 and 7.6 contain the accuracy values obtained for all the trained models. Based on this information, the accuracy value coming from the CNN model with seven layers and 900 samples was the one chosen for the rest of the experiment.

Table 7.5: Accuracy results for trained and tested CNN configurations for five levels of HW.

Hamming Weight	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
5	700	27.2%	18.4%	19.4%	24.2%	24.6%
	900	19.2%	19.0%	26.6%	**30.7%**	30.1%

Table 7.6: Accuracy results for trained and tested ConvLSTM configurations for five levels of HW.

Hamming Weight	Samples per Power Trace	ConvLSTM (layers)		
		1	2	3
5	700	19.0%	19.0%	19.1%
	900	19.2%	**20.4%**	19.2%

7.3.1 Training and Results

The Confusion Matrix of this model, represented in the Figure 7.5, is showing that as we add more HW levels the accuracy keeps decreasing, more specifically:

- For the levels four and six of HW the model was able to correctly predict 63.65% and 43.15% of the times, respectively. Both of these levels are mostly mutually confused by the model;
- For the levels zero and eight of HW, the model was able to correctly predict 18.31% and 21.96% of the times. Both of these levels are mostly confused as being level four or six of HW;
- For the level two of HW the model was only able to correctly predict 1.18% of the time, which is the class with worst accuracy for this model. It can be observed that the model rarely classifies any power trace as level two of HW, and when it does it is usually wrong, thus this class must not carry characteristics that are clear enough for the model to pick up on.

Figure 7.5: Confusion matrix for levels 0, 2, 4, 6 and 8.

In general, the accuracy this model achieved is close to random, and it is specifically extremely poor for all levels, except for the power traces with HW four. Most levels of HW are mostly misclassified as being level four or six of HW, and the model almost never classifies a power trace as being level two but when it does it is mostly wrong, and almost never classifies a power trace as being level eight, and when it does it is generally correct.

Even though the power traces with HW four are the most correctly classified ones, this stems from the fact that the model classifies most power traces as this level, and not because the model is being good at classification, as can be observed by the 72.02% of cases that the model classified as level four that were wrong.

7.4 Nine levels of Hamming weight

The best accuracy result that was possible to obtained from the all the executed tests with the CNNs were 27.7% using seven layers for 700 power traces and 25.6% using six layers for 900 samples. For the ConvLSTMs the best accuracy values were 11.0% applying three layers for 700 samples and 38.7% also applying three layers for 900 samples. The tables 7.7 and 7.8 are holding the information about the global obtained accuracy for all the trained models. For this last level of HW that is being studied, the best obtained value came from a ConvLSTM with three layers that achieved 38.7% of accuracy, which

was the chosen model to continue the study.

Table 7.7: Accuracy results for trained and tested CNN configurations for nine levels of HW.

Hamming Weight	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
9	700	25.0%	25.8%	23.1%	**27.7%**	13.2%
	900	19.5%	16.2%	25.6%	18.1%	24.2%

Table 7.8: Accuracy results for trained and tested ConvLSTM configurations for nine levels of HW.

Hamming Weight	Samples per Power Trace	ConvLSTM (layers)		
		1	2	3
9	700	10.9%	10.3%	11.0%
	900	10.9%	10.3%	**38.7%**

7.4.1 Training and Results

The last experiment is displayed in Figure 7.6, where is visible that the final increase in HW levels to incorporate all possible HW levels of a byte had an influence in the obtained overall accuracy just like it had when we were only using the simplified AES. More specifically:

- For the levels two and seven of HW, the NN model was able to correctly predict 86.50% and 93.74% of the times, respectively;
- For the level five of HW, the NN was able to correctly predict only 6.05% of the time, which was the class with the worst accuracy result. In this case, the main contribution for this low accuracy was that the power traces with level five of HW were more times individually classified as any other level of HW (except level seven), rather than as the level five itself.

Figure 7.6: Confusion matrix for levels 0,1, 2, 3, 4, 5, 6, 7 and 8.

Unlike in the equivalent experiment in Section 6.4, this model does not show such a good separation of the power traces associated with bigger HW values and the ones associated with lower HW values. This is expected, as the full first round of the AES algorithm conceals the association between the power trace and the HW value it represents.

7.5 Key Rank

The key rank Function is the only metric left to be used in this work in order to establish the potential of this deep learning approach to SCAs with one full round of the AES algorithm.

In this Chapter several models were trained, with increasing complexity, for diverse levels of available Hamming weights in a byte. It is assumed that the model with the best accuracy value among the different models trained to classify nine different HW levels will be the one with best performance for the attack. The objective of the DL based SCA is to recover the byte of the secret key, since in the data collected, all bytes have the same value.

The rank function was implemented and tested with the best performing model of Section 7.4, the ConvLSTM model with three layers trained with power traces of 900 samples each. Figure 7.7 shows the evolution of the mean rank with each processed power trace, using this model. As it can be observed, the model was able to achieve a mean rank of zero in 15 traces, which means that with 13500 samples of the testing dataset this model was able to completely recover one byte of the secret key, for all cases. Moreover with this model the SCA only needs, on average, 8 power traces to recover the key byte.

This model can be used to discover the secret key byte through the power consumption of a CPU acquired in the exact same conditions as the training dataset.

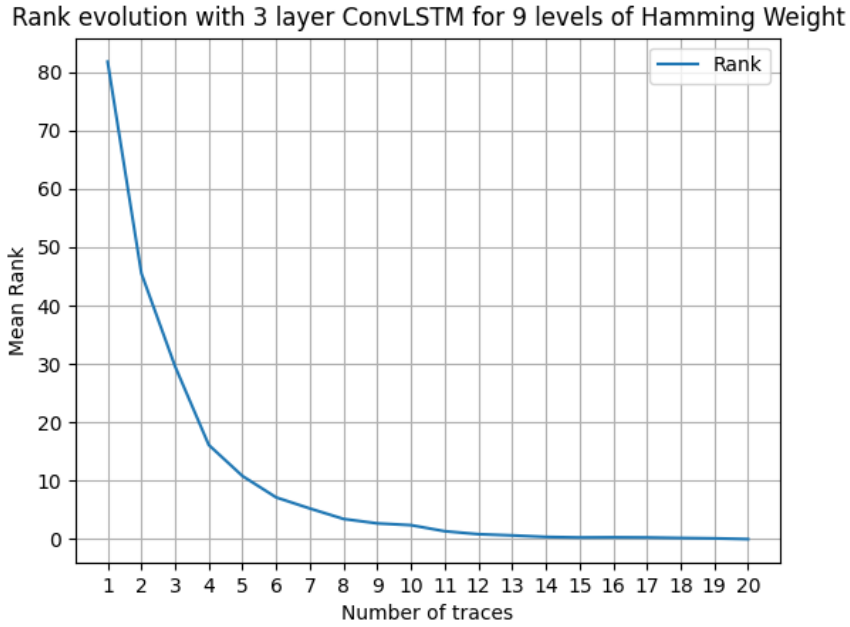

Figure 7.7: Evolution of the mean rank of the attack performed with the best ConvLSTM model trained in Section 7.4.

7.6 Results and Analysis

As expected, with the approximation towards a real world scenario, the accuracy of the trained models has decreased, because the extra operations obscure the connection between the power trace and the HW of the byte at the output of the S-Box.

After all the experiments and results presented in this Chapter, the first thing that can be noticed by looking at Tables 7.9 and 7.10 is that the global values of the obtained accuracy are lower than the previous ones obtained when applying the simplified AES during the encryption.

With a quick analysis of the CNN models results of this Chapter, and comparing them with the ones represented in Table 6.9, a variation of about 11% in the accuracy of the dataset with two different levels of HW, a variation of about 28% in the dataset with three different levels of HW, a variation of about 56% in the dataset with five different levels of HW and a decay of about 53% in the dataset with the full range of HW levels, are enough to demonstrate the difference in complexity between these two different

Table 7.9: Accuracy results for different trained and tested CNN configurations.

Hamming Weight	Samples per Power Trace	CNN (layers)				
		4	5	6	7	8
2	700	**88.6%**	88.5%	88.3%	88.1%	87.7%
	900	87.5%	88.1%	87.5%	86.0%	87.1%
3	700	60.9%	56.7%	29.6%	**66.2%**	65.7%
	900	53.0%	54.0%	60.0%	57.0%	32.5%
5	700	27.2%	18.4%	19.4%	24.2%	24.6%
	900	19.2%	19.0%	26.6%	**30.7%**	30.1%
9	700	25.0%	25.8%	23.1%	**27.7%**	13.2%
	900	19.5%	16.2%	25.6%	18.1%	24.2%

experiments.

It can also be observed, with the exception of the two different levels of HW, which also obtained the best accuracy value using the CNN with four layers and 700 samples per power trace, that the best results for the CNNs were all achieved when applying seven layers either using 700 or 900 samples per power trace. This shows that the increase in complexity with the full round of AES forces us to generally consider the CNN with seven layer more capable of performing this classification task, and that the simpler architectures do not have enough trainable parameters to capture all the needed information to correctly classify the test power traces. But there is still a decrease in accuracy when training CNN with eight layers, which shows that it is still possible to create models that are too complex for the data they are classifying.

What is novel in the results of this Chapter, as compared to Chapter 6, is the good results obtained by the ConvLSTM networks when compared to the CNNs. For all different levels of HW, except five, the ConvLSTM has performed comparably to the CNN, or even better in some cases. As such, the best accuracy for the full range of HW values with one full round of AES was with the ConvLSTM architecture, using three layers with the 900 samples per power trace dataset. Although this accuracy is much worse than the one obtained with the equivalent dataset in Chapter 6, it is still far from arbitrary classification by the model. It can be concluded that the ConvLSTM architecture does not outperform the CNN for simpler datasets, but that for more complex datasets, the time advantage that this type of network brings is relevant for this problem.

For the ConvLSTM, it can also be observed that it mostly benefits from processing a larger time series, that is, processing power traces with 900 instead of 700 samples, which can be explained by its capabilities of dealing with time related data.

Table 7.10: Accuracy results for trained and tested ConvLSTM configurations.

Hamming Weight	Samples per Power Trace	ConvLSTM (layers)		
		1	2	3
2	700	87.6%	**88.3%**	73.7%
	900	87.3%	87.4%	87.0%
3	700	32.5%	65.1%	68.1%
	900	33.2%	33.6%	**68.4%**
5	700	19.0%	19.0%	19.1%
	900	19.2%	**20.4%**	19.2%
9	700	10.9%	10.3%	11.0%
	900	10.9%	10.3%	**38.7%**

Finally, the key rank computed in this Chapter also shows very good results, it is only needed on average seven power traces for a correct key recovery using the best performing model, the ConvLSTM with 3 layers trained with the 900 samples per power trace. As expected, the key rank does not perform as well as in Chapter 6, but it still achieves a very satisfactory result.

Chapter 8

Conclusions and Future Work

Nowadays, with the continuous growth of technology, especially at economic and social levels, the majority of the information is digitized, and all the associated processes that occur also use this technology.

As expected, this non stopping evolution cannot be separated from the less desirable characteristics of technology that create vulnerabilities and new opportunities to be explored. Like so, a very careful approach should be considered when using devices daily, such as a computer, which was the focus of this work. Inside computers, there are several components, such as the processor, that could easily be releasing information that can be exploited and linked to secret information by someone with knowledge and not very sophisticated tools, just a computer. It must be provided extra attention to attacks like SCAs, since they are hard to detect and tough prevent, in conjunction with their dangerous potential, especially with the help of deep learning.

By doing this work, we aim to continue the study of software-based energy measuring SCAs against AES, in an attempt to completely recover the secret key from the power consumption of an Intel CPU. The experiment started by implementing a simplified scenario based on a simplified AES algorithm, evolving from this basis towards a real case scenario with the full AES, which is the most used symmetrical encryption algorithm. In order to do this, it was utilized a framework that reads the values directly from the MSRdrv registers with the help of the RAPL tool, minimizing the waste caused by other frameworks, due their slow acquisition time. By analyzing the obtained preliminary results coming from the power traces of encryptions under these conditions, which represents the link between the processed data and the power consumption detected with RAPL, we were able to conclude that a leakage exists and is possible to detect, and that could possibly be exploited in a manner that might put at risk the system security.

To accomplish the objective of the work, several steps were taken. A laptop was used so the power values resulting from the AES algorithm could be acquired, and these power values were used to create a valid dataset. At the same time several NN models were designed to be trained by this dataset, allowing the HW value classification of the byte loaded from the first S-box operation during the encryptions. Looking at the obtained accuracies with various different models, the models which had the best perfor-

mance on the testing set for both nine different levels of HW studied in this work (for simplified AES and for one round of full AES) were used to perform the key rank Function, thus completing the proposed SCA.

This work can be divided into two main parts, following the steps that were just mentioned. The first part, where it was proven that a leakage can be detected through software based acquisition, and a key byte can be discovered from a simplified encryption implementation, the simplified AES, which we were able to do with great success.

The second part of this work was focused on increasing the complexity by implementing the full AES algorithm with only one round, maintaining the key and the plaintexts as they were. All the other processes were exactly the same as before, including the NN models used with the exception for the RNNs that, as shown before, were not suited for this specific task. As expected, the results obtained for this experiment were not as impressive as the first ones, specially for the higher levels of HW, where the accuracy values were much lower. This reduced capability for good predictions, was certainly due to the considerable amount of extra operations introduced by the full round of AES encryptions, that masked correlation between the power consumption and the value of the byte under attack.

Despite this, the experiment showed that under these conditions, it is possible to detect a leakage and use it to discover a key byte from a full 16 byte AES key with seven power traces, on average, proving that it is possible to use software based tools for measuring the CPU power and also use NN models to correlate, evaluate and formalize the attack, in a simplified scenario.

In our work, some processes could have been done differently, such as applying more pre-processing to the data before creating the datasets, namely a z-score normalization which is commonly done in this type of work, and varying the trained NN hyper parameters, specially the loss function, the optimizer, the learning rate, and if the used GPU had more available memory, the batch size could be incremented speeding up all the processes. Although, given the complexity of the work done and the achieved results, those processes turned out not to be necessary to implement.

Taking into account the available time and the objectives of this thesis, there are more steps that can be taken towards the achievement of a complete full AES SCA that more closely relates to a real world scenario, that were not implemented in this work. These steps would be:

- Performing the AES algorithm in its totality, with the ten complete rounds (even though this objective may also be divided in smaller incremental steps);
- Randomizing all the 16 bytes of the plaintext or the key;
- Randomizing all the 16 bytes of both the plaintext and the key.

One of the reasons for not being able to follow all these steps was that not much studies have been done involving SCAs using DL over Intel's power leakage captured by software.

In summary, the implementation of an SCA using software acquisition and DL-based processing is still a new endeavor, but the preliminary results shown in this work with a simplified SCA scenario have shown promising results. With further research, this type of SCA may come to provide fruitful results, as DL methods are always evolving and delivering better results.

Bibliography

[1] A. J. Menezes, P. C. Van Oorschot, and S. A. Vanstone. Applied cryptography. *CRC, Boca Raton*, 1996.

[2] Y. Zhou and D. Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. *IACR Cryptol. ePrint Arch.*, 2005:388, 2005.

[3] T. Popp, S. Mangard, and E. Oswald. Power analysis attacks and countermeasures. *IEEE Design & test of Computers*, 24(6):535–543, 2007.

[4] H. Gamaarachchi and H. Ganegoda. Power analysis based side channel attack. *arXiv preprint arXiv:1801.00932*, 2018.

[5] S. B. Kotsiantis, I. Zaharakis, P. Pintelas, et al. Supervised machine learning: A review of classification techniques. *Emerging artificial intelligence applications in computer engineering*, 160(1):3–24, 2007.

[6] D. Michie, D. J. Spiegelhalter, and C. C. Taylor. *Machine learning, neural and statistical classification*. Paramount Publishing International, 1994.

[7] Omdena. Top 10 machine learning examples in real life (which make the world a better place). `https://omdena.com/blog/machine-learning-examples/`. Accessed: 2022-09-30.

[8] A. Kumar. Classification problems real-life examples. `https://vitalflux.com/classification-problems-real-world-examples/`. Accessed: 2022-08-17.

[9] M. Hüsken and P. Stagge. Recurrent neural networks for time series classification. *Neurocomputing*, 50:223–235, 2003.

[10] F. Karim, S. Majumdar, H. Darabi, and S. Chen. Lstm fully convolutional networks for time series classification. *IEEE access*, 6:1662–1669, 2017.

[11] D. Smirnov and E. M. Nguifo. Time series classification with recurrent neural networks. *Advanced analytics and learning on temporal data*, 8, 2018.

[12] G. Hospodar, B. Gierlichs, E. De Mulder, I. Verbauwhede, and J. Vandewalle. Machine learning in side-channel analysis: a first study. *Journal of Cryptographic Engineering*, 1(4):293, 2011.

[13] R. Benadjila, E. Prouff, R. Strullu, E. Cagli, and C. Dumas. Deep learning for side-channel analysis and introduction to ascad database. *Journal of Cryptographic Engineering*, 10(2):163–188, 2020.

[14] N. Sklavos and I. D. Zaharakis. Cryptography and security in internet of things (iots): Models, schemes, and implementations. In *2016 8th IFIP International Conference on New Technologies, Mobility and Security (NTMS)*, pages 1–2. IEEE, 2016.

[15] What is key management? how does key management work? https://www.encryptionconsulting.com/education-center/what-is-key-management/. Accessed: 2022-01-03.

[16] J. Bhatia. Comparison of white box, black box and gray box cryptography. *International Journal of Innovations in Engineering and Technology*, pages 217–221, 2017.

[17] P. C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Annual International Cryptology Conference*, pages 104–113. Springer, 1996.

[18] P. Kocher, J. Jaffe, and B. Jun. Differential power analysis. In *Annual international cryptology conference*, pages 388–397. Springer, 1999.

[19] A. Efe and K. M. Alashik. Side channel attack. *gazi university journal of science*, 6:61–73, 2019.

[20] R. Paccagnella, L. Luo, and C. W. Fletcher. Lord of the ring (s): Side channel attacks on the {CPU} on-chip ring interconnect are practical. In *30th {USENIX} Security Symposium ({USENIX} Security 21)*, 2021.

[21] What is a cpu? definition and working. https://www.it4nextgen.com/what-is-a-cpu-central-processing-unit/. Accessed: 2022-01-03.

[22] A. L. N. da Silva. Hacking the systems from within: Using rapl for power analysis. Master's thesis, Instituto Superior Técnico, January 2020.

[23] rapl. https://www.intel.com/content/www/us/en/developer/articles/technical/software-security-guidance/advisory-guidance/running-average-power-limit-energy-reporting.html. Accessed: 2022-01-05.

[24] M. Lipp, A. Kogler, D. Oswald, M. Schwarz, C. Easdon, C. Canella, and D. Gruss. Platypus: Software-based power side-channel attacks on x86. In *IEEE Symposium on Security and Privacy (SP)*, 2021.

[25] Skylake soc block diagram. https://en.wikichip.org/wiki/File:skylake_soc_block_diagram.svg. Accessed: 2022-01-05.

[26] Building a simple neural network from scratch. https://towardsdatascience.com/building-a-simple-neural-network-from-scratch-a5c6b2eb0c34. Accessed: 2022-07-05.

[27] Understanding neural network in machine learning. https://techiestalk.in/understanding-neural-network-in-machine-learning/. Accessed: 2022-07-05.

[28] Illustration of max pooling and average pooling. https://www.researchgate.net/figure/Illustration-of-Max-Pooling-and-Average-Pooling-Figure-2-above-shows-an-example-of-max_fig2_333593451. Accessed: 2022-07-05.

[29] Building a simple neural network from scratch. https://towardsdatascience.com/building-a-simple-neural-network-from-scratch-a5c6b2eb0c34. Accessed: 2022-07-05.

[30] 20 questions to test your skills on cnn. https://www.analyticsvidhya.com/blog/2021/05/20-questions-to-test-your-skills-on-cnn-convolutional-neural-networks/. Accessed: 2022-07-05.

[31] Recurrent neural network(rnn). https://www.researchgate.net/figure/Recurrent-neural-networkRNN-or-Long-Short-Term-MemoryLSTM-5616_fig2_324883736. Accessed: 2022-07-05.

[32] M. Randolph and W. Diehl. Power side-channel attack analysis: A review of 20 years of study for the layman. *Cryptography*, 4(2):15, 2020.

[33] H. Mestiri, N. Benhadjyoussef, M. Machhout, and R. Tourki. A comparative study of power consumption models for cpa attack. *International Journal of Computer Network and Information Security*, 5(3):25, 2013.

[34] E. Brier, C. Clavier, and F. Olivier. Correlation power analysis with a leakage model. In *International workshop on cryptographic hardware and embedded systems*, pages 16–29. Springer, 2004.

[35] D. Agrawal, B. Archambeault, J. R. Rao, and P. Rohatgi. The em side—channel (s). In *International workshop on cryptographic hardware and embedded systems*, pages 29–45. Springer, 2002.

[36] B. S. Kaliski, Ç. K. Koç, C. Paar, et al. *Cryptographic hardware and embedded systems–CHES 2002: 4th international workshop, Redwood Shores, CA, USA, August 13-15, 2002: revised papers*. Springer, 2002.

[37] N. Hanley, M. Tunstall, and W. P. Marnane. Unknown plaintext template attacks. In *International Workshop on Information Security Applications*, pages 148–162. Springer, 2009.

[38] C. O'Flynn and A. Dewar. On-device power analysis across hardware security domains. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pages 126–153, 2019.

[39] H. Mantel, J. Schickel, A. Weber, and F. Weber. How secure is green it? the case of software-based energy side channels. In *European Symposium on Research in Computer Security*, pages 218–239. Springer, 2018.

[40] Papi. https://icl.utk.edu/papi/. Accessed: 2022-01-05.

[41] Powercap. https://www.kernel.org/doc/html/latest/power/powercap/powercap.html. Accessed: 2022-01-05.

[42] Ippet. https://community.intel.com/t5/Software-Tuning-Performance/Intel-Platform-Power-Estimation-Tool-IPPET-available/td-p/1002643. Accessed: 2022-01-05.

[43] S. Ghandali, S. Ghandali, and S. Tehranipoor. Deep k-tsvm: A novel profiled power side-channel attack on aes-128. *IEEE Access*, 9:136448–136458, 2021.

[44] A. Heuser and M. Zohner. Intelligent machine homicide. In *International Workshop on Constructive Side-Channel Analysis and Secure Design*, pages 249–264. Springer, 2012.

[45] S. Picek, A. Heuser, A. Jovic, and L. Batina. A systematic evaluation of profiling through focused feature selection. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 27(12): 2802–2815, 2019.

[46] J.-W. Chou, M.-H. Chu, Y.-L. Tsai, Y. Jin, C.-M. Cheng, and S.-D. Lin. An unsupervised learning model to perform side channel attack. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 414–425. Springer, 2013.

[47] S. Ding, J. Yu, B. Qi, and H. Huang. An overview on twin support vector machines. *Artificial Intelligence Review*, 42(2):245–252, 2014.

[48] S. Ding, X. Zhang, Y. An, and Y. Xue. Weighted linear loss multiple birth support vector machine based on information granulation for multi-class classification. *Pattern Recognition*, 67:32–46, 2017.

[49] Z. Martinasek, P. Dzurenda, and L. Malina. Profiling power analysis attack based on mlp in dpa contest v4. 2. In *2016 39th International Conference on Telecommunications and Signal Processing (TSP)*, pages 223–226. IEEE, 2016.

[50] Z. Martinasek, J. Hajny, and L. Malina. Optimization of power analysis using neural network. In *International Conference on Smart Card Research and Advanced Applications*, pages 94–107. Springer, 2013.

[51] W. Fischer and N. Homma. *Cryptographic Hardware and Embedded Systems–CHES 2017: 19th International Conference, Taipei, Taiwan, September 25-28, 2017, Proceedings*, volume 10529. Springer, 2017.

[52] B. S. Kaliski, Ç. K. Koç, C. Paar, et al. *Cryptographic hardware and embedded systems–CHES 2002: 4th international workshop, Redwood Shores, CA, USA, August 13-15, 2002: revised papers*. Springer, 2002.

[53] Hdf5 for python. https://docs.h5py.org/en/stable/. Accessed: 2022-07-05.

[54] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard, et al. Comparison of classifier methods: a case study in handwritten digit recognition. In *Proceedings of the 12th IAPR International Conference on Pattern Recognition, Vol. 3-Conference C: Signal Processing (Cat. No. 94CH3440-5)*, volume 2, pages 77–82. IEEE, 1994.

[55] S. Picek, A. Heuser, G. Perin, and S. Guilley. Profiling side-channel analysis in the efficient attacker framework. *Cryptology ePrint Archive*, 2019.

[56] P. Ghanty, S. Paul, and N. R. Pal. Neurosvm: An architecture to reduce the effect of the choice of kernel on the performance of svm. *Journal of Machine Learning Research*, 10(3), 2009.

[57] Advanced encryption standard. `https://www.tutorialspoint.com/cryptography/advanced_encryption_standard.htm`. Accessed: 2022-04-07.