UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona

Analyzing the Limits of Deep Learning Applied to Side Channel Attacks

Degree Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya
by

CRISTIAN FERNÁNDEZ ORTIZ

In partial fulfillment
of the requirements for the degree in
Telecommunications Technologies and Services **ENGINEERING**

Company Advisor: DAVID HERNÁNDEZ
University Advisor: SERGIO BERMEJO
Barcelona, June 2022

Contents

List of Figures

List of Tables

Abstract

Society is advancing by leaps and bounds in terms of technology in recent decades. These advances come with new products and services, which are generally designed within a few years, and potentially without undergoing tests to verify whether they are susceptible to physical or logical attacks. In an increasingly connected world, it is necessary to highlight the importance of cybersecurity. Within cybersecurity there is the field of hardware, where products can also have vulnerabilities. For instance, the information that cryptographic algorithms manage could be exploited by an attacker.

This thesis is based on one of the most innovative techniques for analysing side-channel attacks: deep learning. In particular, the limits that may exist in the world of side-channel analysis techniques applying deep learning are explored, introducing the readers to the exciting world of hardware attacks. In addition, this thesis provides an introduction to neural computation.

After gaining a detailed understanding of the functioning of ANN applied to SCA through the experiments carried out, previous results based on the ASCAD database have been improved using a better optimization of the models parameters.

Resumen

La sociedad avanza a pasos agigantados en materia de tecnología en las últimas décadas. Estos avances vienen acompañados de nuevos productos y servicios, que generalmente se diseñan en pocos años, y potencialmente sin someterse a pruebas para verificar si son susceptibles de ataques físicos o lógicos. En un mundo cada vez más conectado, es necesario destacar la importancia de la ciberseguridad. Dentro de la ciberseguridad está el campo del *hardware*, donde los productos también pueden tener vulnerabilidades. Por ejemplo, la información que manejan los algoritmos criptográficos podría ser explotada por un atacante.

Esta tesis se basa en una de las técnicas más innovadoras para analizar los ataques de *side-channel*: *deep learning*. En particular, se exploran los límites que pueden existir en el mundo de las técnicas de análisis *side-channel* aplicando aprendizaje profundo, introduciendo a los lectores en el apasionante mundo de los ataques por *hardware*. Además, esta tesis ofrece una introducción a la computación neuronal.

Tras conocer en detalle el funcionamiento de las ANN aplicadas a SCA a través de los experimentos realizados, se han mejorado los resultados anteriores basados en la base de datos de ASCAD mediante una mejor optimización de los parámetros de los modelos.

Resum

La societat avança amb passes de gegant en matèria de tecnologia en les últimes dècades. Aquests avanços venen acompanyats de nous productes i serveis, que generalment es dissenyen en pocs anys, i potencialment sense sotmetre's a proves per a verificar si són susceptibles d'atacs físics o lògics. En un món cada vegada més connectat, és necessari destacar la importància de la ciberseguretat. Dins de la ciberseguretat està el camp del *hardware*, on els productes també poden tenir vulnerabilitats. Per exemple, la informació que manegen els algorismes criptogràfics podria ser explotada per un atacant.

Aquesta tesi es basa en una de les tècniques més innovadores per a analitzar els atacs de *side-channel*: *deep learning*. En particular, s'exploren els límits que poden existir en el món de les tècniques d'anàlisis de *side-channel* aplicant l'aprenentatge profund, introduint als lectors en l'apassionant món dels atacs *hardware*. A més, aquesta tesi ofereix una introducció a la computació neuronal.

Després de conèixer detalladament el funcionament de les ANN aplicades a SCA a través dels experiments realitzats, s'han millorat els resultats anteriors basats en la base de dades de ASCAD mitjançant una millor optimització dels paràmetres dels models.

A mi familia, por todo el apoyo durante estos años

Acknowledgements

I would like to express my deepest gratitude to all my colleagues at Applus+ Laboratories during the development of this project, especially to David Hernández PhD for his time and all the help he has given me for the development of the work as a tutor in the company, as well as to Victor Mico for his lessons on deep learning and to Natalia Mendo for sharing with me her knowledge on side-channel techniques.

In addition, I would like to thank Sergio Bermejo for his time as tutor for the follow-up of the work.

I would also like to thank my family and especially my mother and my grandmother for their unconditional support during these last hard years of my studies.

Finally, I would like to express my gratitude to my friends and colleagues at the university, and above all to my partner for all the encouragement I have received to keep going.

Revision history and approval record

Revision	Date	Purpose
0	26/02/2022	Document creation
1	02/03/2022	Document modification
2	25/04/2022	First document revision
3	13/06/2022	Second document revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
CRISTIAN FERNÁNDEZ ORTIZ	cristian.fernandez.ortiz@estudiantat.upc.edu
DAVID HERNÁNDEZ	david.hernandez.g@applus.com
SERGIO BERMEJO	sergio.bermejo@upc.edu

Written by:		Reviewed and approved by:	
Date	26/02/2022	Date	20/06/2022
Name	Cristian Fernández	Name	Sergio Bermejo
Position	Project Author	Position	Project Supervisor

1 Project plan

1.1 Introduction

Applus+ Laboratories, an **Applus+** division dedicated to Testing, Inspection and Certification projects, leads this project contextualized in the field of hardware cybersecurity. This division is responsible for validating the security of devices that manufacturers seek to certify before releasing them to the market.

Generally speaking, cybersecurity is strongly supported by cryptology. Cryptology is a science dedicated to the study of secret writing, which is divided into cryptography and cryptanalysis. Cryptography is of great importance and has the function of protecting the confidentiality and integrity, among others, of secrets and sensitive information. On the other hand, cryptanalysis is the group of techniques meant for breaking cryptography. Therefore, security could be compromised if adequate measures are not taken into account in our increasingly connected world, which is the reason for being of cryptanalysis, with the aim of finding weaknesses in cryptographic techniques.

Within the world of cybersecurity, we can find different targets for attacks (hardware, software, networks...). This project targets attacks and vulnerabilities associated with hardware security.

In the field of cybersecurity, we must also consider technologies related to artificial intelligence, which can allow us to verify that a product or service meets the requirements to be certified as 'secure' more quickly and easily, and even finding vulnerabilities in devices that classical techniques are not capable of.

Thus, the main objective of the project, or what this project intends to cover, is to improve the current artificial intelligence networks used by the **Applus+ Cybersecurity Laboratory** to take a step forward in current limitations of hardware attacks.

These hardware attacks, where artificial intelligence is used, seek to locate and exploit vulnerabilities in the hardware through the power it consumes and the electromagnetic (EM) fields it radiates while operating, in order to obtain the keys of the different encryption algorithms, among other purposes detailed in the next section. That is the objective of Side Channel Attacks.

Obtaining this type of sensitive information, such as keys of the different encryption algorithms, is not trivial. Normally, the artificial neural network trained for one device is not capable to obtain information from another device in the same way and it is necessary to train again a new model. In general, training deep-learning models are expensive in terms of time and computing resources and usually the evaluators do not have enough time or resources to train new models. This amount of time and computing resources can be considered two limits, but not the only ones, when applying deep-learning models in Side-Channel analysis techniques. This project looks forward to analyze these deep-learning limits applied in Side-Channel attacks.

For this purpose, different deep-learning models will be trained to study the effect of the different configurations, modifying deep-learning parameters, topologies, etc. Different metrics will be evaluated to check the models results using a set of public traces obtained using side-channel techniques, available for researchers.

The starting point of this project is the paper *"Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database"* [1] published by Emmanuel Prouff et al. in 2018. The authors evaluated the results of different configurations of deep-learning models using their own dataset (or traceset) which contains traces obtained with Side-Channel techniques over an AES-128 implementation in a 8-bit microcontroller. Advanced Encryption Standard (AES) is a symmetric cryptography algorithm that uses the same key to cipher and decipher. It is large implemented nowadays. See Appendix A for AES explanation step by step to understand the algorithm. This dataset is used during all this project to compare the authors results and analyze the limits of deep-learning applied in Side-Channel.

In addition, with the objective to improve deep-learning models, two recent papers have been choosen to implement the two proposed deep-learning techniques and analyzing the results in terms of best results and perfomance. These recent papers are:

- *"Ranking Loss: Maximizing the Success Rate in Deep Learning Side-Channel Analysis"* [2] published by Zaid et al. in 2020. This paper introduce the Ranking Loss function, proposing use this new loss function for training deep-learning models.
- *"The Need for Speed: A Fast Guessing Entropy Calculation for Deep Learning-based SCA"* [3] published by Guilherme Perin et al. in 2021. This paper introduce the Fast Guess Entropy as an alternative metric in deep-learning models.

Both techniques and the experimental procedure are explained in more detail in section 2.2 and 3, respectively.

1.1.1 Introduction to Side-Channel Analysis (SCA)

The world of cybersecurity is quite a wide world. In this area, we can find network security, hardware security and software security. Within each area, we find different threats, and specialized attack topologies for each of these cybersecurity threats. This project will address hardware security, concretely the attack method of Side-Channel Analysis (SCA), utilizing artificial intelligence for that.

These SCA techniques seek to obtain sensitive information (such as the secret key of an encryption algorithm: AES, DES...) by measuring the power consumption or the electromagnetic radiation of the device while performing the cryptographic operations. This power consumption varies depending on the functions that the device is being carried out. These variances between the power consumption or electromagnetic fields that are generated by the current that circulate in the device, allows to analyze the different steps of the algorithms that the device is being carried out. Once the signals are measured and the interest points of the algorithm has been located in function of the power consumption, they are analyzed by these artificial intelligence algorithms to retrieve the sensitive data, such as byte values of a sensitive variable utilized by the algorithm that can be related to the secret key.

In general terms, there are three analysis methods to obtain leakages from devices:

- *Electromagnetic fields analysis*: monitoring the radiation that devices emanate when they are processing, for instance, cryptographic algorithms. Is possible concentrate

the probe tool in one hardware module to obtain an 'unique radiation' and depreciate the radiation of the other modules.

- *Power consumption analysis*: during cryptographic algorithms the hardware consumes a characteristic power consumption that contains sensitive information such as the output of one S-BOX, if device applies AES or other symmetric key encryption method.
- *Timing analysis*: such as the other analysis, timing analysis consist of measuring and analyzing how much time is needed to do mathematical operations or others crucial operations that contain sensitive information.

In particular, this type of attacks will be applied to Smartcards. A Smartcard is a card containing a small chip in charge of processing sensitive operations such as performing payments in a payment terminal or to identify a user e.g. the Spanish DNI. The information these devices contain is sensitive information and therefore it is necessary to verify that it complies with the security requirements/standards defined by the organizations in charge of managing and controlling the payment network, the census of a country, or the organizations issuing e-Passports, among others. The figure below shows a general SCA setup using EM techniques.

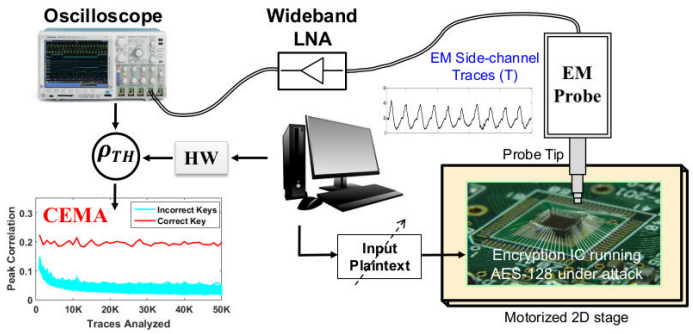

Figure 1: Side-Channel Analysis setup using EM techniques. Adapted from [4]

The hardware needs more power when try to perform more complex and costly operations or functions due to many logic gates and transistors are activated, manipulate data, change state, etc. This principle of SCA allows to identify differences in power consumption.

The traces obtained will show an increase in power consumption when the device works with more complex functions or algorithms. Even so, it is not easy to observe these differences and to locate the points of interest during the algorithms of interest. These traces may contain noise that masks, unintentionally or intentionally as a countermeasure created by the hardware developer, the points of interest or the real power consumption of the device.

These traces obtained are analog traces, which after processing to digitize them are composed of samples. The set of traces is called traceset. The number of traces is consequently high because of the large quantity of traces needed to visualize or identify patterns due to the noise added during the power consumption or EM fields acquisition. Once these patterns are identified, it is possible to find the 'critical execution time', e.g. when the

hardware is doing the encryption algorithm and consequently the device is working with private keys.

1.1.2 Introduction to Artificial Neural Networks (ANN)

Artificial Neural Networks (ANN) are used to obtain and identify patterns or useful information within these traces obtained with SCA techniques. The next image represents a standard configuration of ANN.

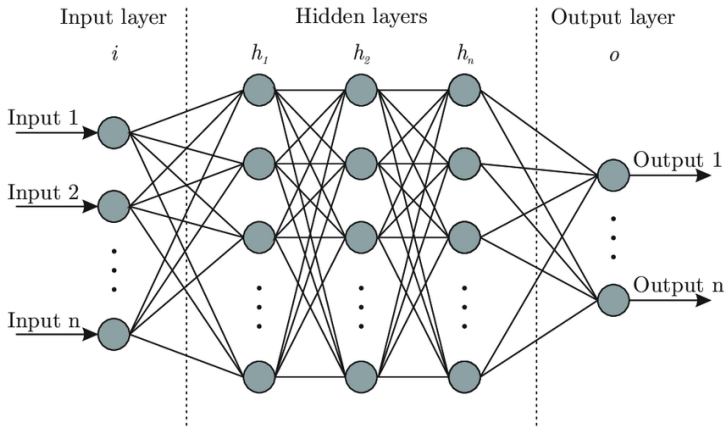

Figure 2: Artificial neural network general schematic. Adapted from [5]

Warren M. and Walter P. introduced the idea of ANN in 1943. They created the first computational model for ANN. During the 40s decade, Hebb introduced a learning hypothesis based on the behavior of neurons. This was known as Hebbian learning, nowadays known as unsupervised learning. Approximately twenty years later, the research of neural networks stopped due to Minsky and Papert discovered that perceptrons were incapable of processing the exclusive-or circuit and computers had not sufficient power to process useful neural networks. The major limitation for ANN was the poor computational power that existed in these years. It was not until around 1995, when the ANN were implemented in stock market and self-driving cars models and then started a revolution in artificial intelligence (IA). For more information about the history of ANN see the chapter History in [6].

The first element that can be found in ANN are perceptrons, or better known as neurons, introduced by Frank Rosenblatt in [7], published in 1958.

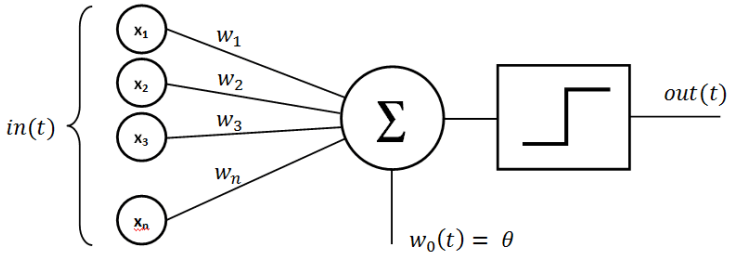

Figure 3: General perceptron/neuron scheme. Adapted from [8]

Neurons are characterized by having an activation function, similar to the human neurons behaviour, which can vary and have different shapes, such as:

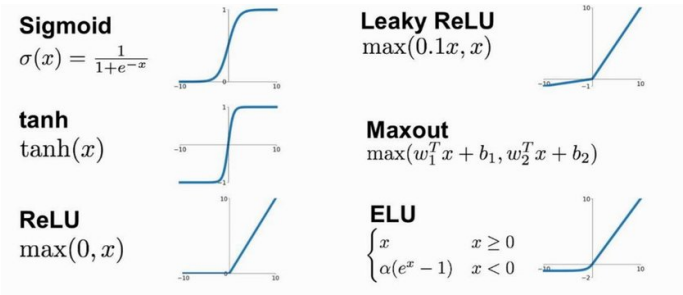

Figure 4: Typical activation functions. Adapted from [9]

With one of these neurons it is possible to distinguish between two colors, for instance red and blue. Should the ANN have to identify more colors, more than one neuron will have to be used.

The purpose of these ANN models is the correct performance in solving data classification or regression problems to make possible predictions.

There are different configurations of ANN. This paper will focus on the development and performance comparison of the architectures:

- Convolutional Neural Networks (CNN): used by Kevin J. et al. in [10] and based on backpropagation techniques and the gradient descent optimizer, which are still used nowadays. The backpropagation technique was introduced by Seppo L. in [11]. This technique allows backward steps for each of the neurons from the output to check whether the network is learning correctly and to correct the behavior of the ANN. The optimizers, such as gradient descent optimizer, are mathematical methods that, when used in neural networks, it is possible to achieve maximum or minimum points in order to improve learning.

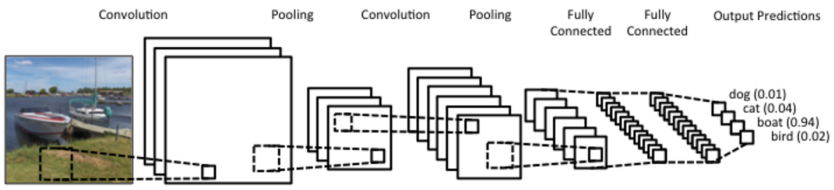

Figure 5: Example of CNN model. Adapted from [12]

The main feature of CNN is that they are widely used in image classification, as they are able to obtain features in images that allow distinguishing between objects or recognizing, for instance, people or gestures. Filters are applied to reduce the search size of these intrinsic image features. An instance of the use of filters can be seen in Figure 11, where Pooling Layers are used to search for these patterns in the initial image. At the end, these features are added as input in MLP containing a set of fully connected neurons to perform the classification.

- Multi-layer Perceptron Networks (MLP): explained in detail in [13] by L. Noriega, MLP consist of several hidden layers with a distribution of neurons in each of them. In this way all neurons of the hidden layer can be connected with the all neurons of the next hidden layer.

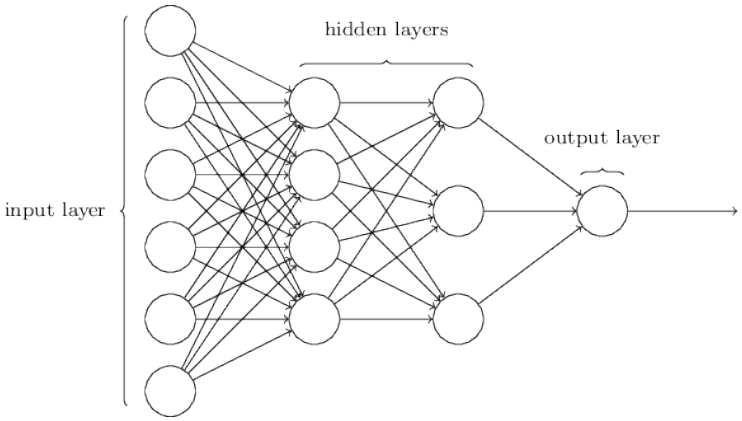

Figure 6: Example of MLP model. Adapted from [14]

MLP can also rely on the Backpropagation technique to adjust each neuron to check the correct learning of the network and use the gradient descent optimizer or another optimizer.

Once these two classes of ANN have been introduced, it is necessary to introduce the way ANN work. By introducing a large dataset, the ANN will perform three phases: training, validation, and testing. The data from the dataset is approximately divided in the following way: 70% of the data for training the network, 20% for validating the training, and finally 10% for testing the trained ANN.

The validation phase is used to check the correct behavior of the ANN, whether overfitting or underfitting occurs. Overfitting and underfitting refer to the concept that ANN learn incorrectly and fail to solve classification or regression problems as they should. In the case of overfitting, the trained ANN is not capable to generalize, i.e. to adapt to different scenarios other than the one it has learned, and will always try to solve it in the same way. As for underfitting, the trained ANN is not capable to identify patterns and thus, solve the initial problem and the other possible scenarios.

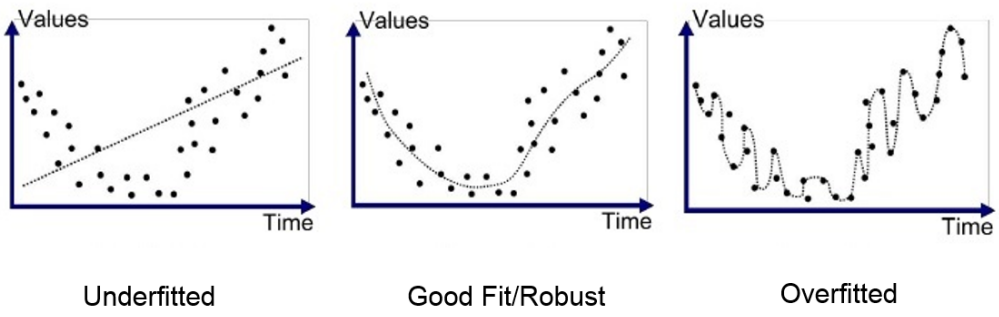

Figure 7: Overfitting, underfitting and correct behaviour. Adapted from [15]

All perceptrons (neurons) begin training with random or assigned initial weights. During training, these weights change based on the loss function and the optimizer working together.

The ANN classes explained before are applied in SCA techniques to facilitate the work of evaluators. Increasingly, countermeasures are implemented to prevent information leaks and sensitive information from being obtained through SCA techniques.

1.2 Requirements and Specifications

Within this point there are defined the requirements or functionalities that will be achieved with the implementation of different deep learning techniques applied to SCA. Project requirements and specifications:

- ASCAD dataset contains traces to train the models and validate the training of deep-learning models. This free dataset published by ANSSI (*Agence nationale de la sécurité des systèmes d'information*) is available online for researchers for testing ANN in SCA. In addition, there are available Python Scripts for training and testing models using this dataset. See 3.1 for more detail of ASCAD dataset and scripts.
- Python programming environment to modify Python Scripts (published by the ASCAD paper author's) and implement others techniques.
  - Python V3.7.8
  - Tensorflow V2.4.0
  - Keras V2.4.3
  - Numpy V1.21.6
  - H5PY V3.6.0
  - Matplotlib V3.5.1

1.3 Methods and procedures

The project starts from scratch, as it is not a continuation of any other project. The main objective is to improve performance and obtaining results by including two deep-learning techniques proposed by the community/researchers. In this way, training, validation and testing phases will be carried out using these techniques to check if the improvements in the projects associated with deep learning and the use of ANNs are relevant.

By implementing these techniques, we intend to achieve a number of objectives:

- Improving the performance and training speed of ANNs applied to SCA.
- Techniques and/or improvements to avoid or prevent ANN overfitting.
- Improving or facilitating the selection of hyper-parameters of ANNs.

1.4 Work plan

The methodology to organize the project consists of a breakdown structure where we find the different work packages. In each of the work packages the tasks to be performed are specified, with their corresponding deliverables and delivery times. In addition, a Gantt Diagram is also included for the organization of time and tasks to be performed.

1.4.1 Work Breakdown Structure

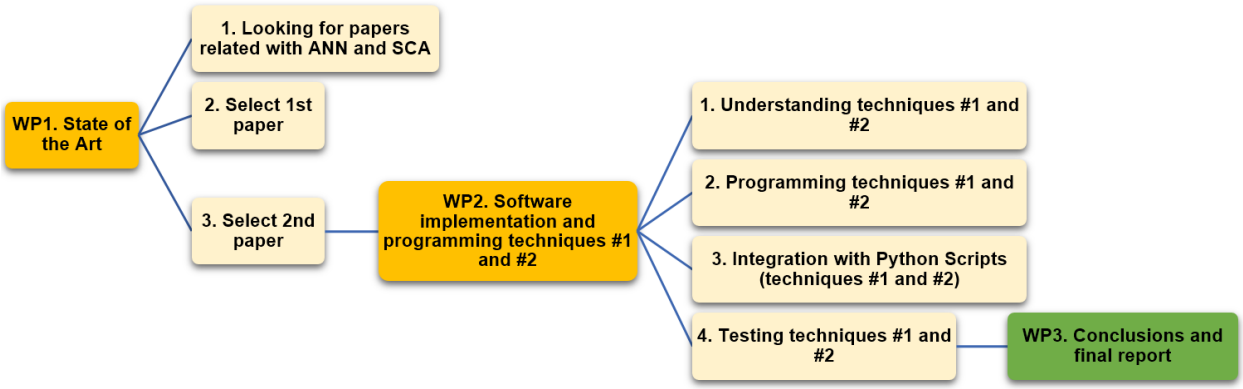

Figure 8: Work Breakdown Structure

1.4.2 Work Packages

Project: Analyzing the Limits of Deep Learning Applied to Side Channel Attacks	WP ref: (WP1)	
Major constituent: Previous analysis of State-of-the-Art (recent papers)	Sheet 1 of 4	
Short description: Search for scientific articles related to Side Channel Attacks and deep learning. Once the papers with the techniques to be applied have been located, the following work packages will be redefined, if it would be necessary.	Planned start date:14/02 Planned end date: 08/03 Start event: 14/02 End event: -	
Internal task T1: Looking for papers that implement deep learning techniques for avoiding overfitting, improving performance, reducing the error… related to Side Channel Attacks. Internal task T2: Select 1st paper based on ANN and SCA techniques. Internal task T3: Select 2nd paper based on ANN and SCA techniques.	Deliverables: -	Dates: -

Project: Analyzing the Limits of Deep Learning Applied to Side Channel Attacks	WP ref: (WP2)	
Major constituent: Software implementation of the technique proposed in the paper: The Need for Speed A Fast Guessing Entropy Calculation for Deep Learning-based SCA. Fast Guess Entropy is proposed as a metric to evaluate the trained deep-learning models.	Sheet 2 of 4	
Short description: Hyper-parameter tuning is an exhausting work when, in the training phase, deep-learning models are running due to the large size of the datasets (traces). The purpose of this technique is to reduce the time of execution and to improve the performance of the deep learning models in the training phase, such as early-stopping techniques in deep-learning or use it to contrast with the most used metric in ANN applied in SCA.	Planned start date: 20/03 Planned end date: 10/06 Start event: TBD End event: -	
Internal task T1: Understanding the technique to be implemented or tested. Internal task T2: Programming the technique (if there is not any code published) or copy the code, if it exists. Internal task T3: Integration with Python Scripts. Internal task T4: Testing the technique and contrasting the author's opinion or expectation.	Deliverables: Software implemented	Dates: -

Project: Analyzing the Limits of Deep Learning Applied to Side Channel Attacks	WP ref: (WP2)	
Major constituent: Software implementation of the technique proposed in the paper: Ranking Loss Maximizing the Success Rate in Deep Learning Side-Channel Analysis. Ranking Loss is proposed as loss function applied in deep-learning during the training phase.	Sheet 3 of 4	
Short description: With the goal of improving the deep learning models performance applied in SCA, this technique seeks to change the loss function and replace the typical loss functions applied in standard deep learning problems.	Planned start date: 20/03 Planned end date: 10/06	
	Start event: TBD End event: -	
Internal task T1: Understanding the technique to be implemented or tested. Internal task T2: Programming the technique (if there is not any code published) or copy the code, if it exists. Internal task T3: Integration with Python Scripts. Internal task T4: Testing the technique and contrasting the author's opinion or expectation.	Deliverables: Software implemented	Dates: -

Project: Analyzing the Limits of Deep Learning Applied to Side Channel Attacks	WP ref: (WP3)	
Major constituent: Final conclusions and final report	Sheet 4 of 4	
Short description: This document includes all the documentation, tests performed and conclusions obtained from the previous work packages, in this case, associated with the experiments and the definition of the work. This WP include de final presentation defense of the project.	Planned start date: 02/05 Planned end date: 21/06	
	Start event: TBD End event: -	
Internal task T1: Define all procedures and conclusions about WP2. Internal task T2: Software implementations conclusions Internal task T3: General work/project conclusions. Internal task T4: Presentation and thesis defense.	Deliverables: Final report document	Dates: 10/06 internal and 21/06 to ETSETB.

1.5 Gantt Diagram

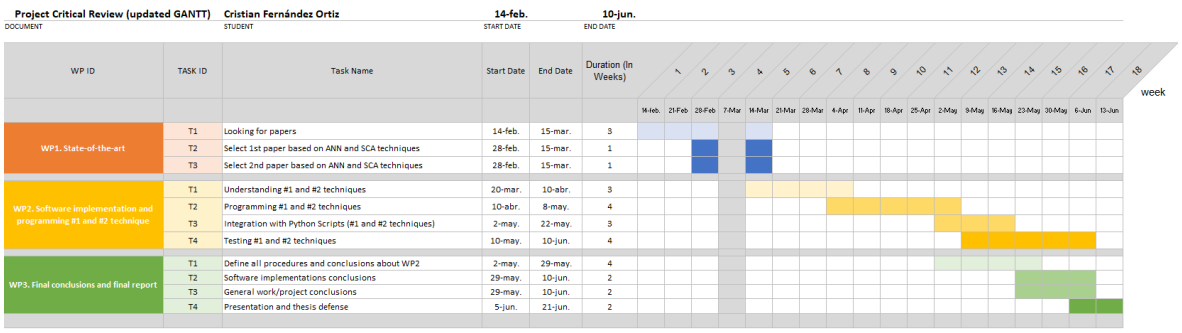

Figure 9: Gantt Diagram Time Plan

1.6 Deviation from the original plan and incidences

As a consequence of contracting Covid-19, Cristian Fernandez stopped the progress of the project from March 7 to March 14 as he was unable to attend Applus+ Laboratories where he was carrying out the project. Therefore, the planning of the project had been modified before to deliver Critical Review document.

The first Gantt Diagram Time Plan was modified. The week from March 7 to March 14 had been removed as working week. All work weeks had been delayed by one week. Furthermore, some tasks changed:

- T1 – Understanding techniques #1 and #2: added one working week
- T2 – Programming techniques #1 and #2: added one working week
- T3 – Integration with ApShadow tool (#1, #2): removed one working week
- T4 – Testing techniques #1 and #2: removed one working week

2 State of the art of the technology used or applied in this thesis

Side Channel Analysis techniques have been improved over the years, as have the technologies and/or techniques used in the world of deep learning and neural networks. This section will focus on the current situation regarding deep learning techniques applied to SCA. An overview will be made and it will be explained what the thesis will be based on or at least what it is intended to cover.

2.1 State of the Art of Side Channel Analysis

The Side Channel Analysis (SCA) techniques were first introduced by P. Kocher in the paper [17] published during the CRYPTO'96 conference in 1996. P. Kocher analyzed how to extract the secret key from RSA [18], Diffie-Hellman [19] and other asymmetric cryptographic algorithms and protocols by analyzing the timing of operations conducted by the device during the modular exponentiation of these algorithms. Kocher also defined the countermeasures that should be used to avoid these side channel techniques on these asymetric cryptographic algorithms.

A few years laters, two of the most important techniques of side channel analysis were described in [22] by P. Kocher. The Differential Power Analysis and Simple Power Analysis, known as DPA and SPA respectively, which consist of analyzing a group of traces to find the critical points where the sensitive information are processed by cryptographic algorithm. SPA consists of obtaining a reduced number of traces to contrast them to find the critical points where the sensitive information are processed. Unlike SPA, DPA is capable, through statistical methods and exploiting the power consumption leakage, to identify the interest byte value in these critical points.

Later on, in [23] published by E. Brier et al., the Correlation Power Analysis technique was introduced. This technique is similar to DPA but this one resorts to pearson coefficients to obtain the leakage information in the consumption traces obtained. This technique also allows to identify the interest byte value of the key with more precision than DPA.

When the techniques of electromagnetic analysis were introduced in [24] by Dakshi A. the way to obtain the power traces were based on using a electromagnetic probe, such as micro-antennas, and pre-processing the traces with low-pass filters to reduce the noise and finally obtain the desired traces.

Other technique presented by S. Chari et al. in [25] are the Template Attacks. Basically, this technique consists of characterise one 'open' device to obtain certain patterns to identify the critical points where the sensitive information are processed, to exploit information in other 'closed' device (with the equal behaviour and features). At this point, it is necessary to introduce the difference between profiled and non-profiled attacks.

In [26], B. Timon explained the two different classes of Side-Channel Analysis techniques that involve these analysis methods, profiled and non-profiled attacks:

- *Profiled Attacks*: consist of an attack in which the attacker needs to have access to two identical devices, known as the 'target' device and the 'profiling' device. The attacker knows and has full control over the 'profiling' device. This 'profiling' device

allows to recognize the behavior of the 'target' device in terms of power consumption, cryptographic algorithm execution time, etc. The attacker will use the information obtained in the 'profiling' device to exploit and find in sensitive information on the 'target' device.

Examples of Profiled Attacks: Template Attacks (explained above), Stochastic Attacks introduced in [27] by W. Schindler and Machine-learning-based Attacks introduced in [28] by G. Hospodar. These attacks are very similar and pretend to characterize one controlled device to exploit information in another closed device with the same features.

- *Non-Profiled Attacks*: unlike Profiled Attacks, Non-Profiled Attacks consist of an attack in which the attacker does not need two devices, but is able to collect several side-channel traces from one device, knowing the inputs or outputs and not knowing the value of the fixed key (for cryptographic algorithms). The attack relies on statistical methods such as Pearson Correlation or Mutual Information to infer, for instance, the key value of the cryptographic algorithm.

  Examples of Non-Profiled Attacks: Differential Power Analysis, Correlation Power Analysis and Mutual Information Analysis (MIA) introduced in [29] by B. Gierlichs (these non-profiled attacks are not considered in this thesis).

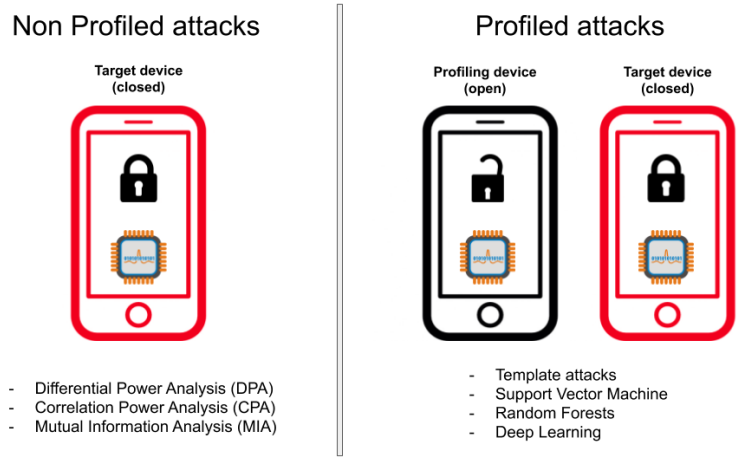

Figure 10: Non-Profiled and Profiled Attacks. Adapted from [30]

This thesis focuses on Profiled Attacks. The objective is to try to exploit sensitive information and try to discover the keys used in the cryptographic algorithms implemented, in this case, AES-128 bits. The type of data and the methodology of work carried out in this thesis is based on these attacks, whose procedure is defined in the book [31] published by François-Xavier Standaert and François Koeuneis:

1. The attacker has full control over the profiling device, from which the characteristic traces of the critical operations, such as AES-128 algorithm, can be obtained. Implicitly in the power or EM radiation traces, the sensitive information to be exploited is found.

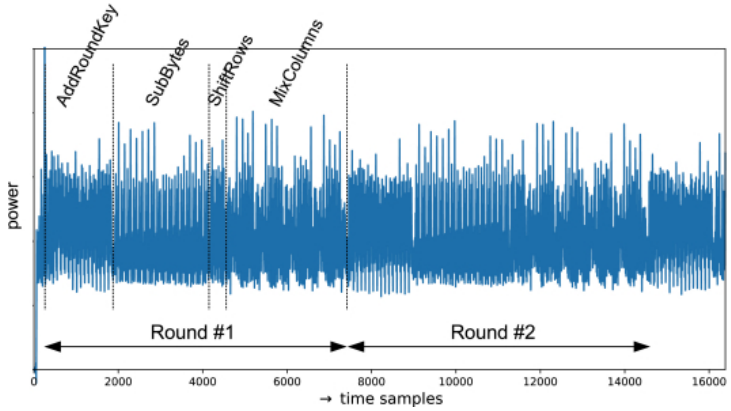

Figure 11: Example of a trace with AES algorithm execution. Adapted from [32]

2. By utilizing a reduced number of traces, the attacker can compare them to find the points of interest, based on the critical operations (e.g., AES-128 first SubBytes output) to exploit the critical operations using the SPA and DPA techniques. This technique is based on comparing, overlapping, or matching similarities between the traces to identify these critical operations.

   By having full access to the device, the attacker can check if the hypothetical point of interest is in fact the point of interest to be exploited (e.g., checking the software implementation). In Non-Profiled Attacks it is not possible because the attacker does not have access to the software or firmware of the device.

3. Once the traces are contrasted with the device information, it is possible to model, create templates, or characterize the points of interest where changes occur each time the attacker executes the algorithm on the device. These points of interest occur during the execution of the AES-128 bit encryption algorithm, as explained before.

Before proceeding with the next steps, it is necessary to define which are the points of interest and which characteristics help the attacker to identify them. SCA techniques are based on the analysis of information leakage, and this leakage is especially important in these points of interest.

When implementing a cryptographic algorithm such as AES-128, the points of interest are the outputs of the SubByte or MixColumn transformations of the first or last rounds of this algorithm, due to the non-linearity and the relationship with the SubKeys used. The purpose is to be able to generate or deduce part of the key used in the algorithm by being XORed in these SubByte or MixColumn transformations immediately after the AddRoundKey transformation.

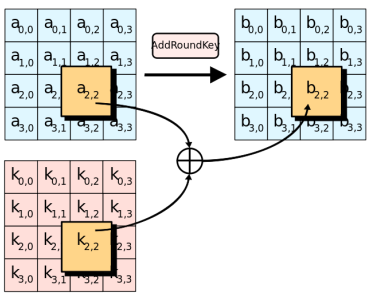

Figure 12: AES SubByte transformation with AddRoundKey XORed. Adapted from [33]

The outputs resulting from the transformations always have an associated leakage of consumption and it can be translated into a leakage of information. One of the ways to consider this leakage is based on the Hamming Weight, as explained in [34] by M. Akkar. According to Hamming Weight it is possible quantify when one bit changes from '0' to '1' or viceversa, as can you see in Figure 16. These bits form the byte value that is attempted to be identified.

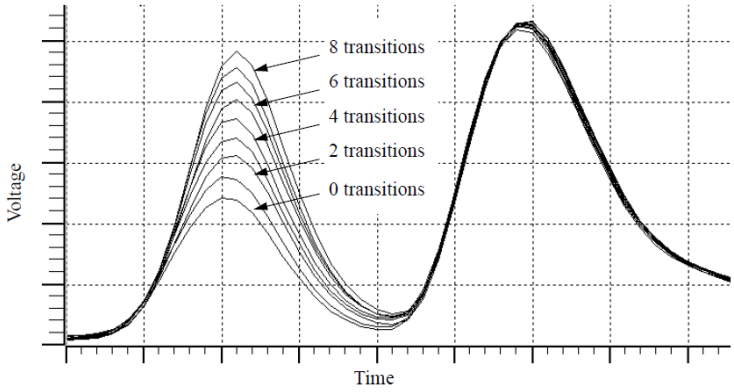

Figure 13: Generic example of Hamming Weight for one byte. Adapted from [35]

By means of these variations of consumption, it is possible to characterize the points of interest and identify each byte value according to the leakage generated.

4. Based on the information extracted from the profiled device (e.g., classifying the different forms of leakage at the point of interest), the attacker creates templates to characterize this information.

5. Then, the attacker checks whether the templates and the target device information match.

The objective is to try to find the points of interest from the profiled device on the target device, over which the attacker has no control. Once located, the attacker uses the generated templates to check the leakage values (value of the trace at the points of interest) so that the attacker can deduce the byte being used in the transformations and thus obtain the possible value of the SubKey used. This can be repeated for each of the bytes in order to find the key used in the AES-128 algorithm.

These templates can be generated either with the Template Attacks technique (manual method) or through techniques based on deep learning (autonomous method), where artificial neural networks are responsible for identifying and characterizing the points of interest in order to determine these byte values. The latter can reduce the time required to perform Profiled Attacks.

2.2 State of the Art of Artificial Neural Networks

In [36], published by I. Goodfellow et al., it is possible to find a definition of the mathematics behind deep learning. From the definition of the different topologies of artificial neural networks to defining the behaviours they can take on when trained: overfitting or underfitting, and much more. This book is a good approach to introduce and understand in detail the basics of deep learning based models.

Some basic concepts on Neural Networks and model assessment are reviewed by T. Hastie et al. in [37]. Specially the chapters: the seven chapter where the introduction to bias, variance and model complexity in relation to model assessment and selection of the best model are explained and the eleven chapter where neural networks in terms of topologies, overfitting and underfitting are explained.

Once the previous books have been introduced that allow to understand in more detail how deep learning based models work, the most relevant articles in relation to deep learning applied to side channel analysis techniques will be cited.

Hetweer et al. made a complete review of the state of the art of ANN aplied to SCA in the article [38]. The authors introduced the relation between classical attacks in SCA and deep learning-based attacks.

Bendadjila et al. completed the work of Hetweer et al. in [39]. They conducted a comprehensive study of deep learning techniques applied in SCA techniques. They chose the ANN class of MLP and CNN to apply deep learning-based alternatives to typical attacks in SCA and tested with different hyper-parameters to obtain a better accuracy/results. In addition, they published an open dataset called ASCAD, containing AES-128 (fixed and variable key) traces to exploit with trained deep learning models.

After the Bendadjila et al. publication, deep learning applied in SCA researchers and investigators have focused on working with the ASCAD dataset.

Two useful examples due to the possible implementation with the scripts and the database provided by the authors of the ASCAD, are *"Ranking Loss: Maximizing the Success Rate in Deep Learning Side-Channel Analysis"* [2] published by Zaid et al. in 2020 and the last year publication by G. Perin et al. called *"The Need for Speed: A Fast Guessing Entropy Calculation for Deep Learning-based SCA"* [3].

In [2], the authors' aim is to adjust deep learning models to the SCA context. The technique they propose is to replace the cross-entropy cost function, based on calculating the entropy based on the probabilities that the network estimates and penalising if this is not correct (with the aim of reducing uncertainty and allowing the network to get it right) with the Ranking Loss function, which is based on learning to classify the probabilities by rankings and penalising the network when the ranking is not as expected, depending on the final position of the expected data in the first position.

The authors worked with CNN topology and three different datasets: the ASCAD, the AES_HD published by S. Picek in [40] and the Chipwhisperer (only ASCAD database is analyzed in this project).

In [3], the author's propose a new technique based on a new metric called Fast Guess Entropy, which consists of the calculate of the guess entropy but using a few traces

for validation phase during the model training to improve the training phase of deep learning models. In addition, they suggest using this FGE technique to monitoring the early stopping technique.

Before continue with the FGE explanation, it is necessary to understand the early stopping technique used in deep learning.

Explained in [41] by L. Prechelt, the objective of this powerful technique is stopping the training phase in an specific epoch when the metric, previously choosen by the evaluator to analyze the learning path of the model, falls into divergence and fails to converge, indicating that the model is generating overfitting or underfitting behaviour, following the next figure:

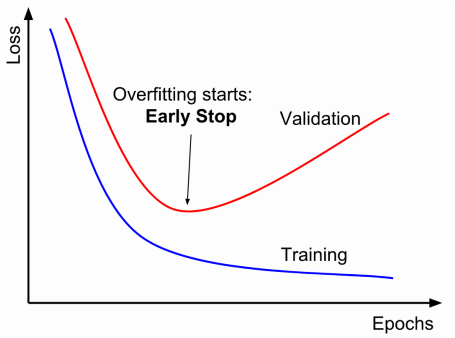

Figure 14: Early-stopping concept. Adapted from [42]

Typically, the metric used in the validation phase in parallel with the training phase are selected as monitor to early stopping. Well, the FGE author's propose using the validation FGE such as monitor to early-stopping, to select and save the network weights for the best epoch where the model obtain the best behaviour. According to the authors, training a model a small number of epochs can improve the generalisation of the network to new data sets.

These two papers seek to improve the performance of ANN applied to SCA by modifying the loss function of the models and using another valid metric for validating results, respectively. More details will be provided in the experimental part (see 3).

3 Methodology/Project Development

This section will explain the procedures performed to find the best model of MLP ANN applied in SCA, modifying different parameters such as the number of layers, of neurons, the optimizer used, etc. In addition, analyzing the Cross-Entropy loss function replacement by the *Ranking Loss Function* (RL) proposed in the paper [2] and checking the results of training with the *Fast Guess Entropy* (FGE) metric proposed in the paper [3]. The loss function will be contrasted with the most standardized cross-entropy loss function and the *Fast Guess Entropy* metric will be compared with the Mean Rank, explained in detail in section 3.2. In addition, MLP class is used to test the behavior of these newly proposed techniques.

EXPERIMENT PROCEDURE: In the First Stage, obtaining a good model for the MLP class. The hyper-parameters tuning is based on the paper [1]. Emmanuel Prouff et al. chose as the starting point a MLP_base model. After a lot of tests with different parameters, they obtained a MLP_optimized model. The objective is to contrast the results of these authors with the results of this thesis.

In the Second Stage, implementing the FGE metric and the RL function and testing the results for a pre-adjusted ANN model using the public ASCAD dataset as input data. It is important to note that the hyperparameters of the ANN model have been pre-adjusted, i.e., they have been previously tuned to obtain accurate results for an MLP_optimized model through the modified Python scripts. The scripts are explained in 3.1.

In the Final Stage, comparing results of the First Stage and the Second Stage using the FGE metric and the RL function.

EXPERIMENT RESOURCES: The dataset ASCAD, used in the project, is public. All related information can be found in `https://github.com/ANSSI-FR/ASCAD`. The ASCAD database consists of three groups of 60,000 traces originated from 8 bit AVR microcontroller ATMEGA8515, with an AES, fixed key and acquired with EM techniques. For each group, 50,000 of the traces are destined to the training and validation phases, and finally 10,000 for the attack phase (which corresponds to the testing phase). These three groups of 60,000 traces include synchronized traces, traces with a desynchronization of 50 time units delay, and traces with a desynchronization of 100 time units delay. The leakage model is found in the first round of the AES-128 S-BOX (in Byte substitution Transformation) and exploits the third byte of the subkey combined with the plaintext. The following equation expresses the S-BOX output:

$$Y^i(k) = SBOX[P_3^i \oplus k] \tag{1}$$

i is the number of round, P is the byte corresponding to the plaintext XORED with the corresponding byte of the subkey, represented by k.

3.1 Zero Stage

The main objective of this section is to introduce the experimental part procedure. In this Zero Stage, ASCAD Dataset and ASCAD Python scripts are introduced and explained.

3.1.1 ASCAD Dataset

This dataset, published by ANSSI, is avalaible in `https://github.com/ANSSI-FR/ASCAD`. The main purposes of this dataset are executing proves with SCA traces and training/testing ANN applied in SCA.

The file type of ASCAD Dataset is H5. In order to review the H5 file, which contains the information of the ASCAD digitalized traces and the byte values corresponding to each of these traces, the open source tool of HDF Group has been used. This HDFView V3.1.3 tool allows to visualize the digitalized values of each of the traces and the byte value. By using the h5 Python package, it is possible to modify the data contained in this type of files. The hierarchical structure of ASCAD Database is in the right image.

Figure 15: ASCAD hierarchical structure viewed on HDFview tool

The form of ten digital traces in *Profiling_traces* is the following:

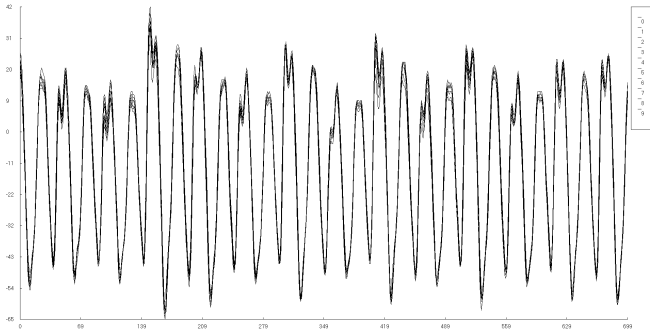

Figure 16: ASCAD traces plotted viewed on HDFview tool

This tool allows to visualize the digitized traces. Each trace contains 700 integer value samples. The *Profiling_traces* also includes the labels section. This section includes, in turn, the byte values of the third output byte of AES S-Box at round one that correspond to each trace.

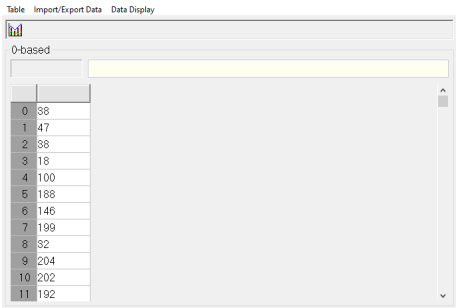

Figure 17: ASCAD traces byte values viewed on HDFview tool

Finally, in the metadata section there are defined the following: the plaintexts, the fixed key, the random masking of S-BOX output bytes, the text ciphered, and the desynchro-

nization applied, all used in the implementation of the AES-128 algorithm.

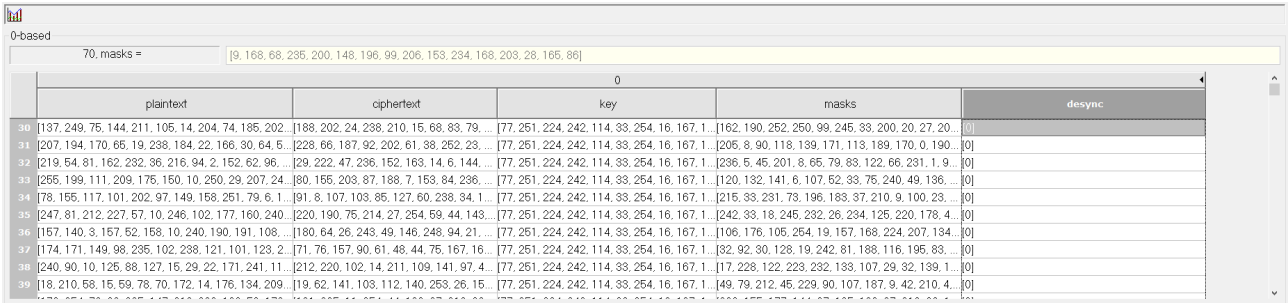

Figure 18: ASCAD traces metadata viewed on HDFview tool

The three groups of traces to be analysed are: synchronised, desynchronised 50 random time delays and desynchronised 100 random time delays. The structure of these traces are as follows:

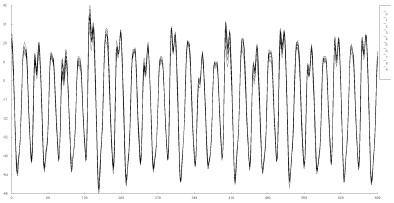

Figure 19: Synchronised traces

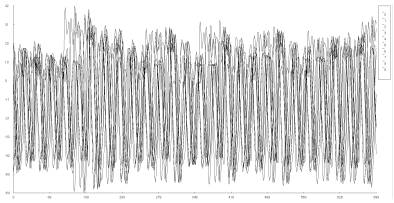

Figure 20: Desynchronised traces (50 time units)

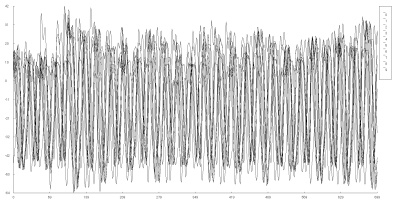

Figure 21: Desynchronised traces (100 time units)

3.1.2 ASCAD Python scripts

Before testing the new techniques, it is necessary to modify ASCAD's public Python scripts to change the execution flow and make them more user-friendly, facilitating the configuration of the models to be trained through keyboard inputs or improving the interpretability of the results through plots. The Python scripts that can be found in `https://github.com/ANSSI-FR/ASCAD` are:

- *ASCAD_generate.py*: allows an evaluator to generate datasets with the equal form of the ASCAD dataset published by the ASCAD authors.
- *ASCAD_train_models.py*: allows an evaluator to train the ASCAD datasets or their own generated datasets. This script is used in this thesis and it will be modified (the code can be found in Annex C.1).
- *ASCAD_test_models.py*: allows an evaluator to test the ASCAD datasets or their own generated datasets. This script is used in this thesis and it will be modified (the code can be found in Annex C.2).

For *ASCAD_train_models.py*, renamed as *ASCAD_training.py*, the configuration of the parameters and the network type has been modified to be more user-friendly. The parameters to be modified can be entered quickly and easily through the terminal. The users can

create MLP and CNN topologies, defining the typical parameters for one of the topologies. In addition, FGE metric can be used such as monitor metric for early-stopping technique, also available as option, and Ranking Loss function also has been implemented. Once the training is finished, the model is saved in a .h5 file to be able to perform different phases of testing a posteriori and study the performance of each of the trained models.

For *ASCAD_test_models.py*, renamed as *ASCAD_testing.py*, the Mean Rank metric implemented in previous Python script has been re-used to check the results of the trained models. The code has been modified to be able to run more than one test phase for different trained models in the same script execution. In addition, it is possible to compare, through graphs of the metrics, the results for different trained models. Other functionality added is the calculus of the Mean Rank averaged calculated when all attack traces (in testing phase) has been processed. Fundamentally, this Mean Rank averaged is a unique value to obtain a better approach in terms of an average ANN learning behaviour. These metrics, Mean Rank and Mean Rank averaged metric, have been added for all models tested in parallel. These metrics allow an evaluator, when ranking a better model, to review this last Mean Rank averaged value and Mean Rank during all the testing phase to have an orientation of the ranking capacity that the trained model may have. See 3.2 for more details about the Mean Rank and accuracy metrics.

Additionally, *main.py* script (Annex C.4) has been created to execute both scripts, *ASCAD_testing.py* and *ASCAD_training.py* in the same execution. *Metrics.py* script (Annex C.3) has also been created to manage the results plots.

3.2 First Stage

In this First Stage, the search of a better model for MLP topology is defined. In addition, the weights of the different hyperparamentes, such as the number of layers, number of neurons, etc. are analyzed in order to understand the importance of these hyperparameters using different SCA traces, synchronised and desynchronised.

Before starting to searching the best model, it is important understand the difference between Mean Rank and accuracy metrics and why is useful use Mean Rank metric.

Mean Rank is a very important metric in ANN applied to SCA, as it allows users to estimate how close they have been to the correct byte for each trace tested in ANN. Basically, Mean Rank calculates the differences in positions with respect to the estimated byte with the correct byte of the key used. Users can check if for each trace they give to the network, it is training correctly and the network are getting closer to the correct byte each time or the trained model have not fit enough during the training phase to identify correctly the different bytes values.

The objective is: obtaining a smaller number of traces that allows users to obtain the value of the bytes in a correct way, and that is able to get closer and closer to the "correct" byte index.

The accuracy metric is not the best way to check the correct learning of the networks as it is quite difficult to identify with exact precision the byte used, since the leakage that allows the network to identify the byte used varies very little from one byte to another.

Therefore, Mean Rank is used as the main metric to check the behaviour of the trained ANN during the testing phase.

It is important to note that not only 1 byte is usually attacked, but several bytes of interest to try to obtain the whole key (or reduce the effort in a brute-force attack). Applied to deep learning, this is known as multi-label, which allows users to select which byte the neural network has to learn to identify.

The search of these best models have been based on [1].

- **MLP**: 6 layers, 200 neurons (with ReLu activation) for each layer and last layer with 256 neurons (with Softmax activation), RMSprop optimizer, 1e-5 as learning rate and cross-entropy loss function.

  The number of epochs is set to 200 with a batch size of 100. It shows a good trade-off between execution time and results (preventing over-fitting/under-fitting).

The tests will be carried out with the three sets of traces published for ASCAD: sync, desync50 and desync100. One best model will be obtained for each dataset, for MLP class. The aim is to analyse the limits of deep learning models and the importance of pre-processing the data before introducing them into the neural networks.

3.2.1 Hyperparameter tuning using SYNC. TRACES

ASCAD AUTHOR'S

The configuration of the different models follows the next layout:

MLP(#layers, #neurons per layer, optimizer, learning_rate, loss function)

The reference is taken as the optimised model that the authors developed and tested in the paper [1], following the following layout:

MLP optimized: (6, 200-200-200-200-200-256, RMSprop, 1e-5, cross-entropy)

The performance of this model has been tested, using both RMSprop and ADAM, which is another optimiser widely used during evaluations in practice as it performs well when training different models. The results have been:

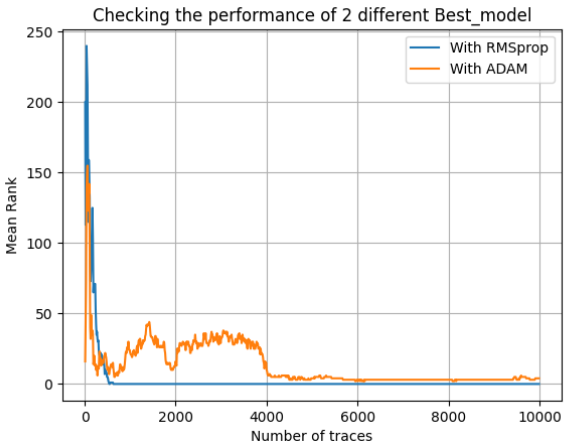

Figure 22: MLP optimized model: Comparison RMSprop vs. Adam for sync. traces

RMSprop has shown a 'faster' and more optimised behaviour in differentiating between bytes. In addition, it required fewer traces to perform a good classification.

The testing traces are introduced to the network, and Mean Rank checks that the plaintext of the XORed algorithm with the key (of fixed value) and applied to the output of the S-BOX is the same byte as the byte predicted by the network. In this way for each trace it is checked, with Mean Ranks it is possible to analyze how the metric tends to 0 because the predicted byte at the output is the correct one.

From now on, different trainings will be carried out by modifying the parameters seen before (See Annex B.1 for all test results).

NUMBER OF LAYERS

The first test carried out to obtain an MLP model with good performance is the modification of the number of layers. Starting from the MLP_optimised model, the number of layers has been modified to 3, 4, 5, 6, 7 and finally 8:

MLP(X, 200-200·X-256, RMSprop, 1e-5, Cross-Entropy)

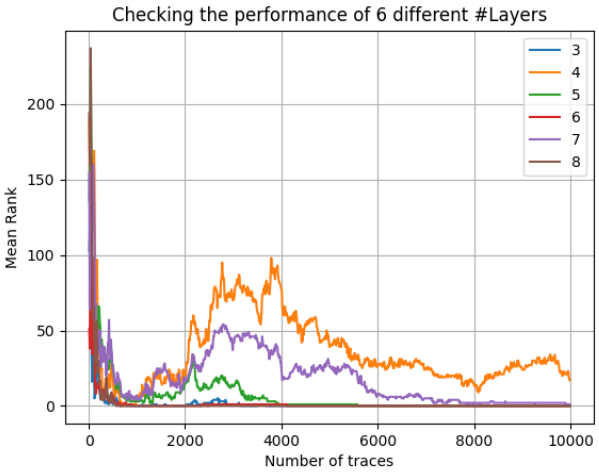

Figure 23: MLP searching best model: #layers for sync. traces

The result after training 6 models with different numbers of layers can be seen in the plot above. The structure that has always been used: input layer with 200 neurons, hidden layer variable according to the assigned layers, but with 200 neurons, output layer with 256 neurons. For 3 layers a good result has been obtained, in fact, for 3 and for 6, more than acceptable results have been obtained in terms of behaviour. Even so, will continue the experiments with the choice of **6 layers**, as this allows us to modify the number of neurons in the next step.

NUMBER OF NEURONS

The second test, which is carried out after having chosen the number of layers equal to 6, is to modify the distribution of neurons by layers.

MLP(6, X-Y-Z-...-256 , RMSprop, 1e-5, Cross-Entropy)

The models that have been used to test the results of using networks are smaller models, speeding up execution and training time, and larger models to see if they are really effective. A model based on 64 neurons has been selected, another with 128 neurons to see if performance improves by doubling the number of neurons, etc. The models between 200 and 500 neurons perform very well, converging to 0 quickly. Two models, Small-Large-Small (such as 256-512-1024-512-256) and Large-Small-Large (such as 256-128-64-128-256), have also been considered, represented by the following analogy:

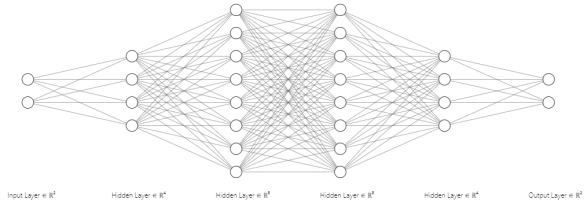

Figure 24: MLP searching best model: Small-Large-Small analogy

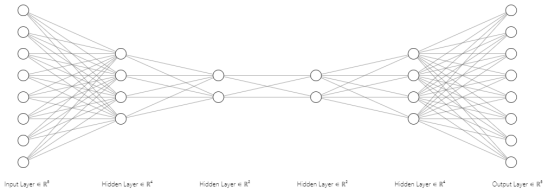

Figure 25: MLP searching best model: Large-Small-Large analogy

The result after training 8 models with different numbers of neurons can be seen in the following plot. The structure remains the same: output layer with 256 neurons.

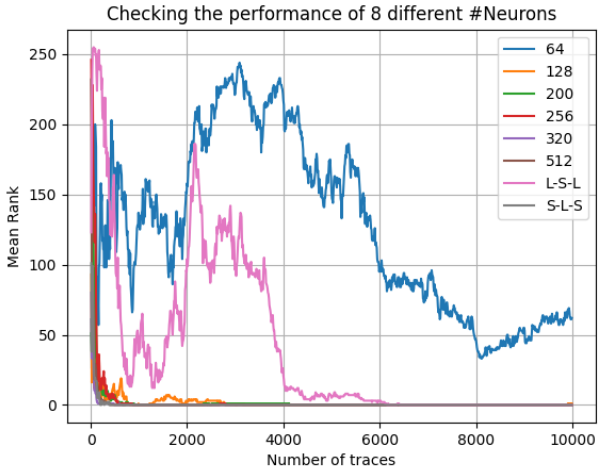

Figure 26: MLP searching best model: #neurons for sync. traces

The model proposed by the authors, with 200 neurons, has a very good behaviour and a rather fast convergence to 0 (few traces are needed).

However, the models that stand out for Mean Rank averaged are the Small-Large-Small model and the 320 neurons per layer model. Consider that any model chosen based on

the number of neurons in the range of Mean Rank averaged less than four could still be trained. The **320 neurons per layer** model is selected as the best option due to the runtimes between the Small-Large-Small model and 320 neurons are x10 apart. Please note that the number of neurons of the last layers is fixed: 256 neurons because we aim the ANN at distinguishing the value of a byte, which has 256 possible values.

OPTIMIZER

The third test is performed after having chosen the number of layers equal to 6 and the distribution of neurons in 320-320-320-320-320-256 by modifying the optimiser used.

MLP(6, 320-320-320-320-320-256 , X, 1e-5, Cross-Entropy)

The result after training with 5 different optimizers, in adition to Adam and RMSprop trained in ASCAD AUTHOR's models proposed, can be seen in the following plot:

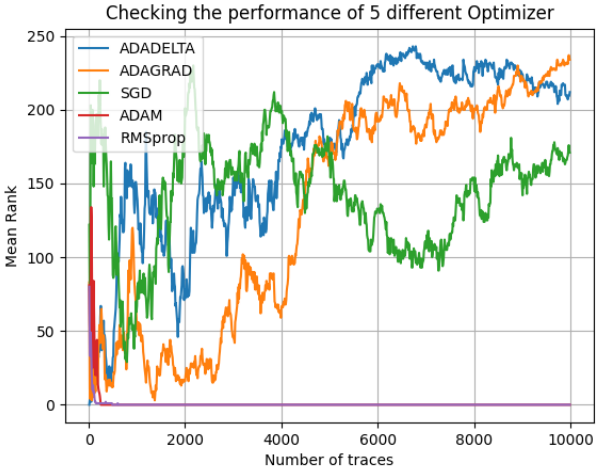

Figure 27: MLP searching best model: optimizers for sync. traces

The model performs very well with the RMSprop and ADAM optimizers, as expected. RMSprop is used by the authors of the paper as it gives good results and ADAM is used in production due to the good results also obtained with this optimiser.

RMSprop uses an adaptive learning rate instead of an initial learning rate as a hyperparameter. This means that the learning rate changes over time and maybe this feature is useful for training ANN applied in SCA.

Adam is the most generalised optimiser for any deep learning application, as it uses concepts of operation from both, RMSprop and Adadelta. These optimisers are based on the descent gradient, as explained in section 1.1.2. RMSprop is based on a similar algorithm, which maintains the behaviour of the descending gradient but averages the exponential differences of the squares of these gradients for each iteration. Adadelta maintains a training factor for each of the weights, based on a window of the previous iterations and not observing from the beginning of the run like AdaGrad. ADAM is a combination of both, so the performance is expected to be superior.

In this case we see that **RMSprop** is a better fit, therefore it is chosen as the best option.

LEARNING RATE

The fourth and last test is performed after having chosen the number of layers equal to 6, the distribution of neurons in 320-320-320-320-320-256 and the RMSprop optimiser by modifying the learning rate used.

MLP(6, 320-320-320-320-320-256 , RMSprop, X, Cross-Entropy)

The result after training with 6 different learning rates can be seen in the following plot:

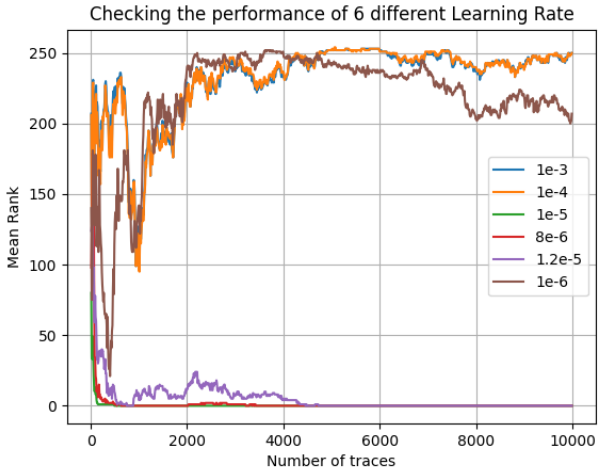

Figure 28: MLP searching best model: learning rates for sync. traces

Values around 1e-5 behave correctly, while learning rates that are far from this initial value of 1e-5 do not train correctly and, therefore, do not conclude in an adequate model. It is important to note that only the learning rate parameter has been modified for each optimiser. The **learning rate 1e-5** is chosen.

3.2.2 Hyperparameter tuning using DESYNC. (50) TRACES

In this section, the experiments have been repeated and will be explained in a shorter way, comparing the results with the synchronised traces to study the effect of desynchronisation and the introduction of unclean data into the neural network. In addition, a search for an optimal model adapted to this type of input data is performed again. See Annex B.2 for all test results.

ASCAD AUTHOR'S

The results of training with the RMSprop and the Adam optimizer are:

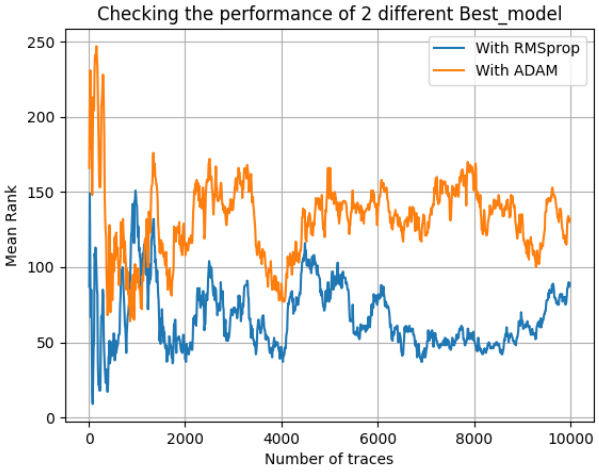

Figure 29: MLP optimized model: Comparison RMSprop vs. Adam for desync50 traces

The RMSprop optimiser has a more correct behaviour in terms of Mean Rank while ADAM diverges a bit more. The behaviours they are quite similar, although a quite differentiated offset it is observed for the two optimisers. Mean Rank averaged is 66.484 for RMSprop and 132.212 for ADAM, as can be seen in the table in the annex B.2.

NUMBER OF LAYERS

The 3 layers option is enough to perform quite well, with a Mean Rank averaged of 54.635. Both 5 layers and 6 layers also give good results, with 55.299 and 66.484 respectively. The results are far from the results obtained with the synchronised traces. The **5-layer model** is chosen because it has obtained better results and because it allows to define different tests by varying the number of neurons.

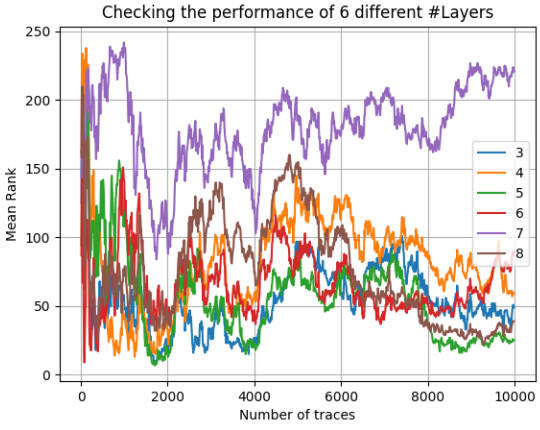

Figure 30: MLP searching best model: #layers for desync50 traces

NUMBER OF NEURONS

The results obtained are:

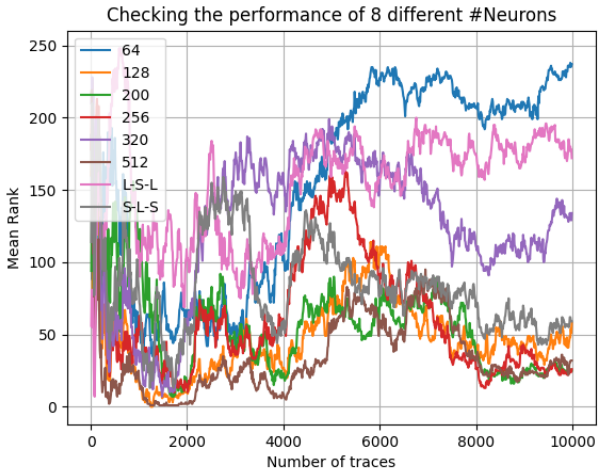

Figure 31: MLP searching best model: #neurons for desync50 traces

The best result is obtained with 512 neurons, with a Mean Rank averaged 33.934. Followed by the 200 neurons model which has a Mean Rank averaged of 55.299. While the 320 neurons model used for the synchronised traces, has a Mean Rank averaged of 125.421. Therefore the model chosen is the **512 neuron model**.

OPTIMIZER

Between the RMSprop optimiser and ADAM, the results are very similar (see the above Figure, where orange is ADAM and blue is RMSprop). They converge at about the same number of traces, around 2000. While the Mean Rank averaged for RMSprop is 33.934 and for ADAM it is 38.523.

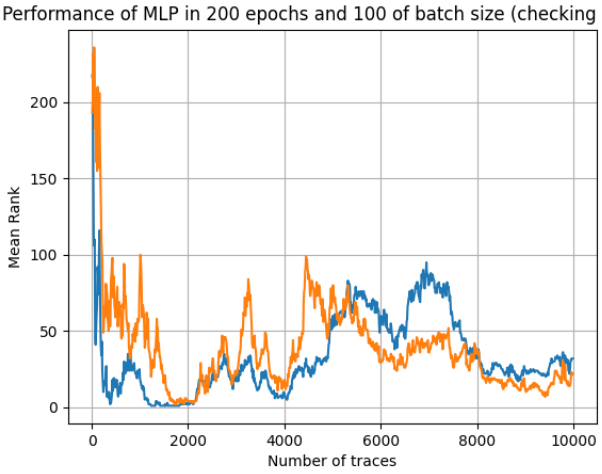

Figure 32: MLP searching best model: Adam and RMSprop for desync50 traces

The results for all optimizers tested, are plotted below:

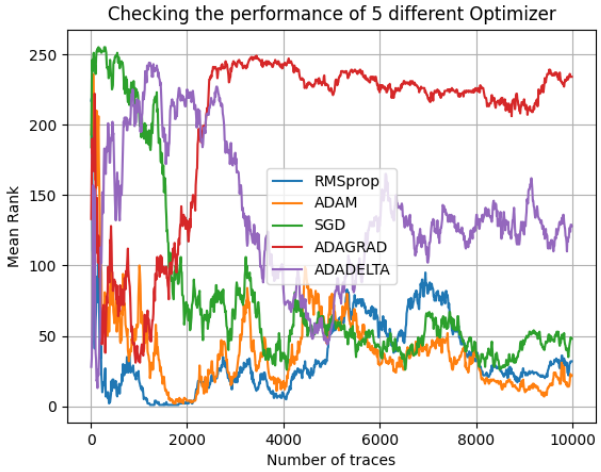

Figure 33: MLP searching best model: optimizers for desync50 traces

The optimiser of chose is **RMSprop**.

LEARNING RATE
The results obtained are:

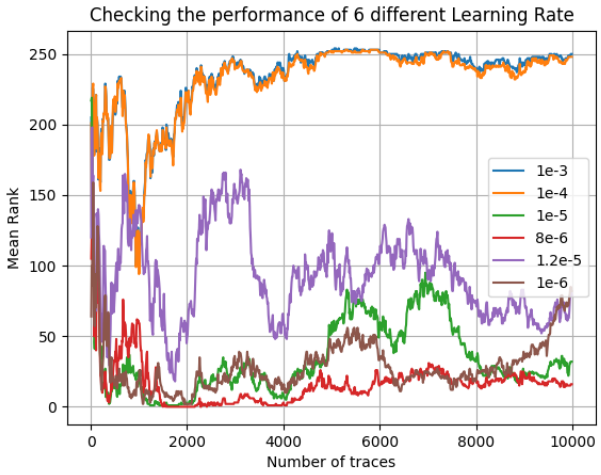

Figure 34: MLP searching best model: learning rate for desync50 traces

The learning rate value with the best Mean Rank averaged is 8e.5 equal to 16.149. The value of 1e-5 is still a good value to choose, with a Mean Rank averaged value equal to 33.934. Learning rate equal to **8e-6** is finally chosen.

3.2.3 Hyperparameter tuning using DESYNC. (100) TRACES

In this section, the experiments have been repeated and are explained in a shorter way, comparing the results with the synchronised traces to study the effect of desynchronisation (worst case, 100 random time delays) and the introduction of unclean data into the neural network. In addition, a search for an optimal model adapted to this type of input data is performed again. See Annex B.3 for all test results.

ASCAD AUTHOR'S
The results obtained are:

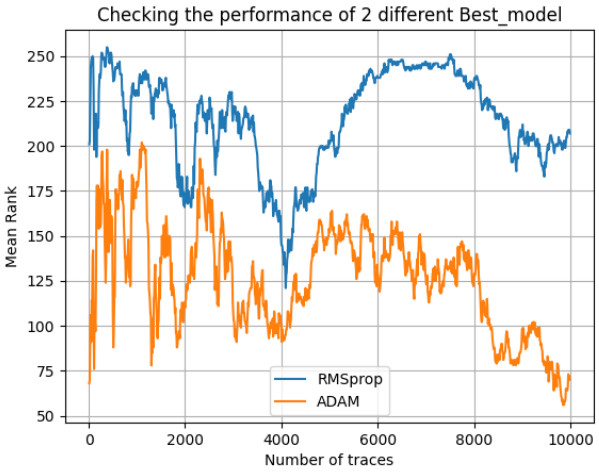

Figure 35: MLP optimized model: Comparison RMSprop vs. Adam for desync100 traces

A higher convergence and more successful behaviour is clearly observed for the ADAM optimiser, unlike RMSprop which fails to converge. Mean Rank averaged for RMSprop is 214.750 and 125.951 for ADAM. ADAM is selected for the choice of the best model.

NUMBER OF LAYERS
The results obtained are:

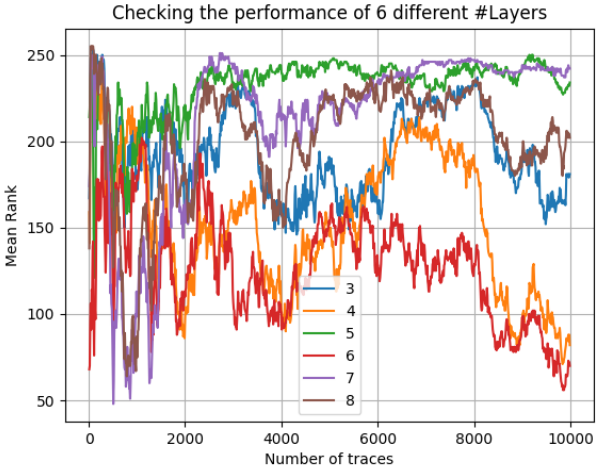

Figure 36: MLP searching best model: #layers for desync100 traces

The number of layers with the best Mean Rank averaged is 6, with a value of 125.95. It converges to a lower value for 10k traces, together with practically 4 layers. It is important to note that for about 500 traces for 7 and 8 layers we are talking about a good Rank (if we limit the number of attack traces). The default number of layers is set to **6**.

NUMBER OF NEURONS

The results for all neuron configurations tested, are:

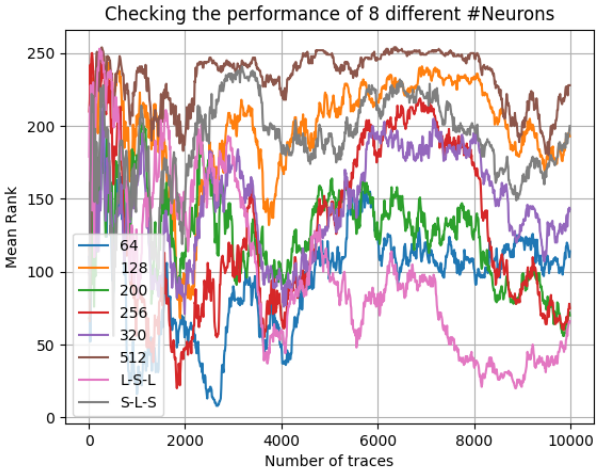

Figure 37: MLP searching best model: #neurons for desync100 traces

Of all the neuron configurations, it can be observed that for the 256 configuration with less than 2000 traces we can obtain a Mean Rank of less than 50.

It is interesting to note that when using models with a reduced number of neurons, for ten thousand traces, it can be observed that end up with a fairly interesting Mean Rank of less than 100. It should also be noted that the model with 200 neurons for 10 thousand traces manages to be below the threshold of 100.

Even so, the **L-S-L model (256-128-64-64-128-256)** is chosen as the best model for the following tests.

OPTIMIZER

The best optimiser is chosen as ADAGRAD, with an averaged Mean Rank of 31,422. The other optimisers do not manage to lower the Mean Rank to less than 100, so the improvement is remarkable. The results obtained are:

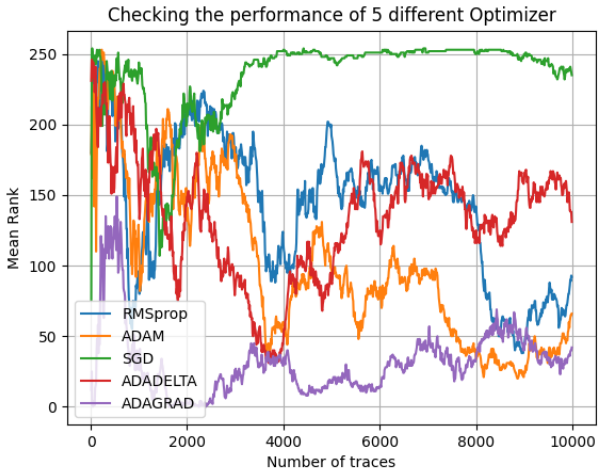

Figure 38: MLP searching best model: optimizer for desync100 traces

LEARNING RATE

The best learning rate 1e-5 is chosen. The results of the experiments are as follows:

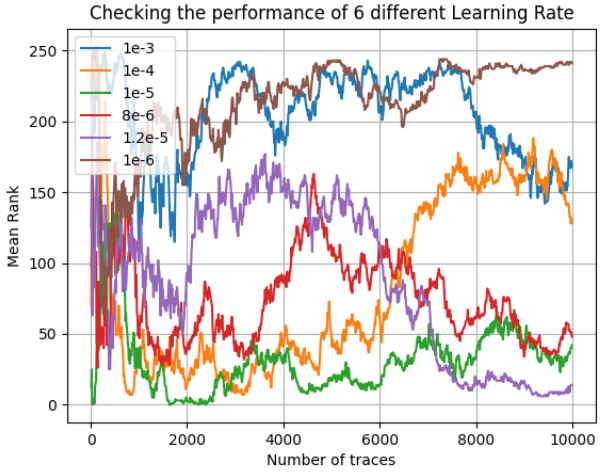

Figure 39: MLP searching best model: learning rate for desync100 traces

3.2.4 Choosing the optimize model

SYNCHRONISED TRACES

The best model obtained after several tests are:

MLP(6, 320-320-320-320-320-256, RMSprop, 1e-5, Cross-Entropy)

With a Mean Rank of **0.348**.

DESYNCHRONISED50 TRACES

The best model obtained after several tests are:

MLP(5, 512-512-512-512-256, RMSprop, 8e-5, Cross-Entropy)

With a Mean Rank of **16.149**.

DESYNCHRONISED100 TRACES

The best model obtained after several tests are:

MLP(6, 256-128-64-64-128-256, Adagrad, 1e-5, Cross-Entropy)

With a Mean Rank of **31.422**.

In addition to replace the loss function Cross-Entropy with Ranking Loss, analyzing the differences between Mean Rank and Fast Guess Entropy metrics will be tested in the second stage. Furthermore, for the three sets of traces, apart from the optimised model found, four models will be studied using not the best options with Ranking Loss function.

3.3 Second Stage

3.3.1 Experiment 1 - Ranking Loss vs. Cross-entropy

Once an optimal model based on the ANN MLP class has been found, the cross-entropy cost function is replaced by the RL function, and the results obtained are studied. Previously to carrying out this experiment, it is necessary to have understood the functioning of this cost function and the difference between the cross-entropy function and the Ranking Loss function, see 2.2.

Different types of loss functions can be found to train different topologies of deep learning models:

- Binary Classification: Binary Cross Entropy allows to classify between two classes.
- Multiclass Classification: Categorical Cross Entropy used in training in 3.2, Sparse Categorical Cross Entropy if the data entered are integers and Poisson Loss for data following a Poisson distribution.
- Object detection and Regression also are available.

In the *ASCAD_train_models.py* script, some of the functions included in the scripts provided by the authors of the RL function have been implemented in order to be able to run training with this new cost function. The RL scripts are based on a CNN structure and the guess entropy, similar to Mean Rank concept of learning to rank, is used as a form of testing. The structure of these scripts has been modified to be able to use the MLP topology, as well as to implement the testing procedure performed during 3.2 and to obtain training results to be able to compare them with the cross-entropy cost function.

For the training performed with the RL function, it is important to highlight the selection of the parameter alpha of the RL function equal to 10. According to the authors, this parameter is considered to be optimal.

Comparisons have been made between the two cost functions for the best model obtained for each set of traces. In addition, training has been carried out by modifying parameters such as the number of layers, the number of neurons, etc. In this way, it has been verified that the modification of the parameters can influence the behaviour of the new models trained with the Ranking Loss cost function.

SYNCHRONISED TRACES

Comparing models with the Cross-Entropy and Ranking Loss function:

MLP (6, 320-320-320-320-320-256 , RMSprop, 1e-5, Cross-Entropy)
vs.
MLP (6, 320-320-320-320-320-256, RMSprop, 1e-5, Ranking Loss)

After training, it can be seen that the CE loss function still performs much better than the RL loss function. Even so, for a little less than about 2000 traces, convergence with the RL function is observed but then it diverges again and fails to behave properly. The figure obtained is as follows:

Figure 40: CE vs. RL comparison between best models for synchronised traces

To analyse the weight of parameters such as the number of layers, the number of neurons, the optimiser, the learning rate in the Ranking Loss cost function, the following results have been obtained after several trainings by modifying these parameters (see B.1 to check the table results):

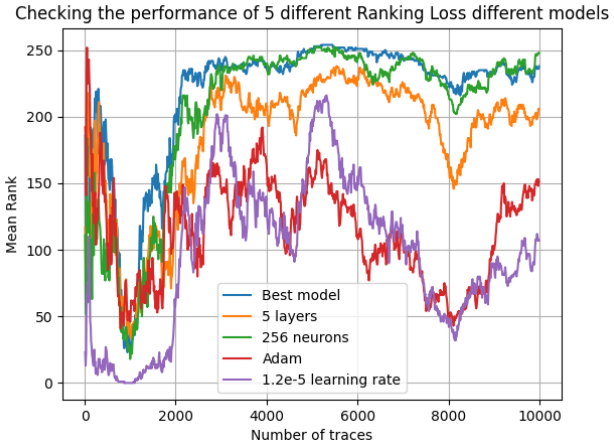

Figure 41: CE vs. RL comparison for different parameters for synchronised traces

Starting from the MLP model *(6, 320-320-320-320-320-256, RMSprop, 1e-5, Ranking Loss)*, the parameters have been changed to check the weight of these. We highlight as an improvement the replacement of the RMSprop optimiser by ADAM, which results in better performance. Also, the learning rate parameter has behaved better by increasing it, as can be seen in the graph.

DESYNCHRONISED (50) TRACES

Comparing models with the Cross-Entropy and Ranking Loss function:

MLP (5, 512-512-512-512-256, RMSprop, 8e-5, Cross-entropy)
vs.
MLP (5, 512-512-512-512-256, RMSprop, 8e-5, Ranking Loss)

The results obtained are as follows:

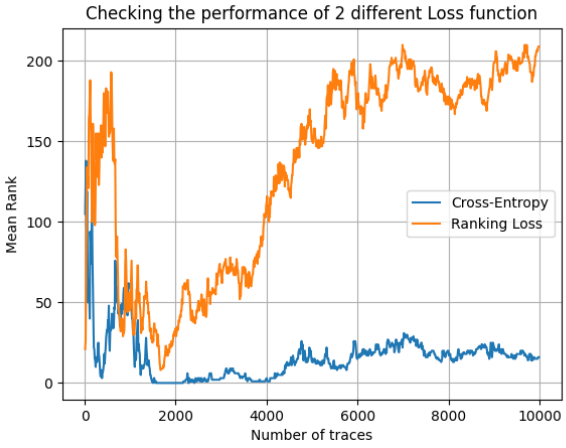

Figure 42: CE vs. RL comparison between best models for desync. 50 traces

After training, it can be seen that the CE loss function still performs much better than the RL loss function. Even so, for about 2000 traces, the RL loss function performs well, similar to the model trained with synchronised traces. We proceed with the modification of the different parameters in the same way as with the synchronised traces. The results are as follows:

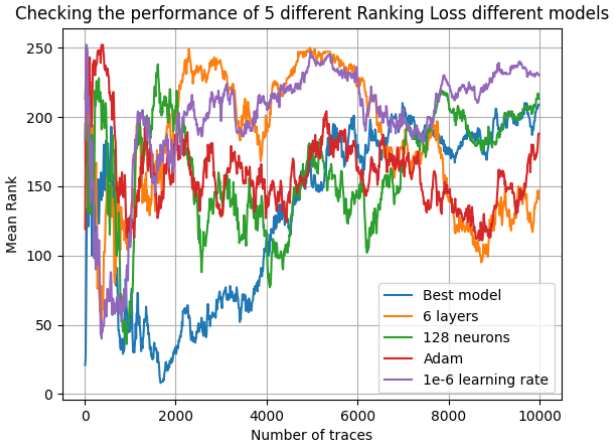

Figure 43: CE vs. RL comparison for different parameters for desync. 50 traces

In the same way as with the synchronised traces, starting from the MLP model *(6, 320-...-256, RMSprop, 1e-5, RL)*, the parameters have been changed to check the weight of these. No noticeable improvement is envisaged for any of the changed parameters.

DESYNCHRONISED (100) TRACES

Comparing models with the Cross-Entropy and Ranking Loss function:

MLP (6, 256-128-64-64-128-256, Adagrad, 1e-5, Cross-entropy)
vs.
MLP (6, 256-128-64-64-128-256, Adagrad, 1e-5, Ranking Loss)

The results obtained are as follows:

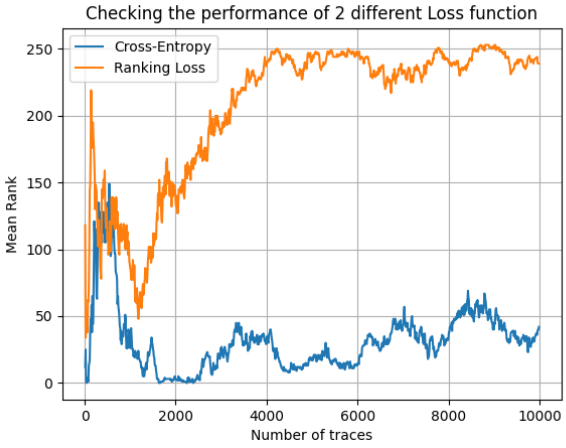

Figure 44: CE vs. RL comparison between best models for desync. 100 traces

It can be seen that the CE loss function still performs much better than the RL loss function. Even so, for before about 2000 traces, the RL cost function performs well. We proceed with the modification of the various parameters. The results are as follows:

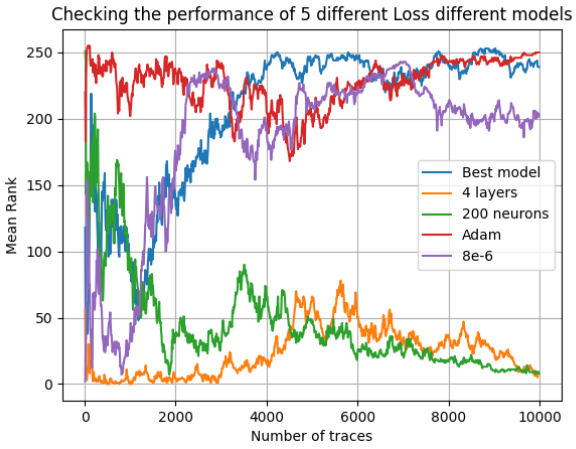

Figure 45: CE vs. RL comparison for different parameters for desync. 100 traces

Surprisingly, better results were obtained using models with a reduced number of layers. Moreover, using the same number of neurons, good results were also obtained. The behaviour is similar when using the CE loss function, but the Mean Rank averaged values obtained are 31.422 for CE and 24.064 for RL. This is an improvement of 23.41% in terms of Mean Rank averaged with respect to CE.

3.3.2 Experiment 2 - Fast Guess Entropy and Rank metrics

Once an optimal model based on the ANN MLP class has been found, the Fast Guess Entropy has been tested, and the results obtained are studied. Previously to carrying out this experiment, it is necessary to have understood the functioning of the different metric to train deep learning models with TensorFlow.

Metrics can be used during training and validation, and later in testing. Among the metrics available in deep learning through the Keras library, there are Regression and Classification metrics. For the Regression metrics, there are available the Mean Absolute Error or the Mean Squared Error, among others. For Classification metrics, there are available Accuracy, which is only used in the training carried out during the First Stage, since Mean Rank is used for testing (it does not directly classify a single byte, but rather the distance the network is from choosing the correct byte). This thesis is focused on the classification metrics, since as introduced in 3.2 the goal is classifying the bytes used and contained in the traces.

A good use of metrics is very important in order to improve a model and understanding the behavior of the trained model.

This is where the FGE technique comes into play (based on the Guess-Entropy metric), which consists of monitoring during each epoch the validation phase an ordered top with the predictions made by the network, in order to check whether the byte that is really the one that the network is getting the most correct. If the model is being trained correctly, the metric should converge to 1, as it will always hit the actual byte in the number one position of the top.

To implement the FGE technique the Guess-Entropy metric and early-stopping function introduced in the introduction of ANN has been used. Within the early-stopping technique, a series of parameters are defined to modify this stop training, among which are the **monitor**, which is the metric to "analyse", **patience**¸ the number of epochs that the values need to be maintained and converge, **verbose**, to indicate when the training ends indicating a message 'EPOCH X: early stopping' and finally **restore_best_weights**, which allows us once the training is finished, to reset the weights of the network to the best model before convergence.

In this way, GE metric is defined such as the **monitor** in early-stopping technique, to know when a good training point is achieved and not to train for more epochs. The difference between monitoring the Guess-Entropy and the Fast Guess-Entropy is defining a reduced number of traces for validation, for instance 1,000 instead of 10,000. The trained model will be more trained for generalisation, according to what the authors propose.

The first step of this experiment is to implement and test Guess-Entropy. Depending on the results obtained with the GE metric, models with the appropriate configuration will be trained to test the FGE technique.

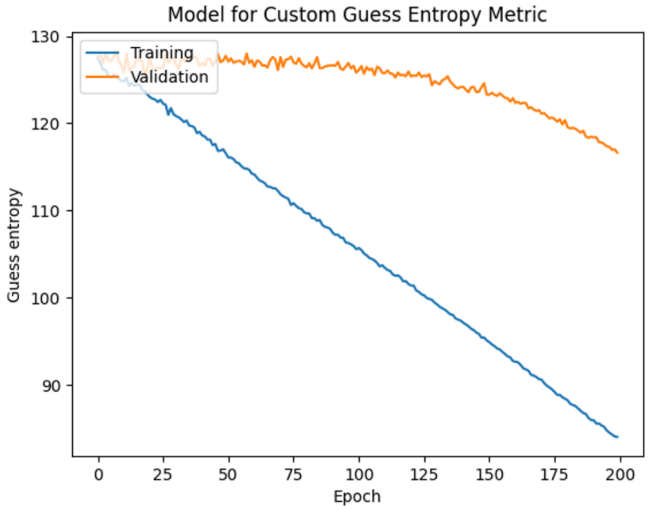

Figure 46: Trained model during 200 epochs with GE

The results obtained after training the previously obtained best synchronised trace model are not as expected. The Guess-Entropy metric does not reach 1 neither in the training nor in the validation phase.

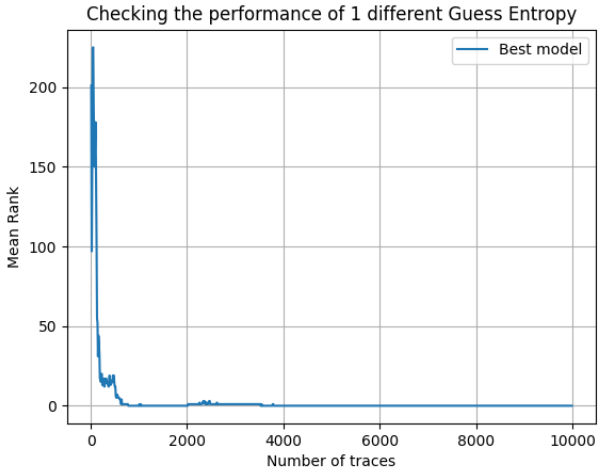

Figure 47: Testing results of trained model during 200 epochs with GE

Checking the behaviour of the trained model during the testing phase, and it is possible to observe that it performs well. How is it possible if the network fails to guess the byte correctly during training, but in the testing phase it identifies the correct byte?

This is because in the testing phase it is not analysed whether the network is able to identify the byte correctly in a direct way, but a "key guessing" is performed. This "key guessing" consists in checking with all the bytes that the network predicts the output of the S-BOX of the AES algorithm, and since the plaintext and the output that the network predicts are known, estimates are made of the possible key byte used. In this way, for a larger number of traces, the approximation is adjusted towards the key byte used.

Based on these unexpected results, it is proposed to carry out training with a number of 1,000 epochs. Two models have been trained for 1,000 epochs with 10,000 traces for validation and a smaller number of traces for validation, as proposed by the authors of the FGE technique.

The results of the trained models were:

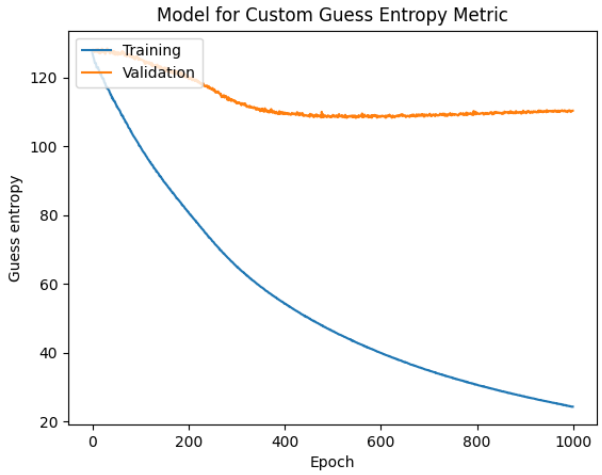

Figure 48: Trained model (1k val. traces) during 1,000 epochs

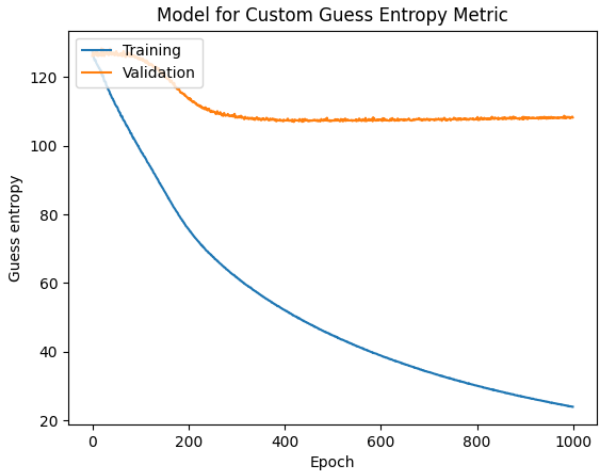

Figure 49: Trained model (10k val. traces) during 1,000 epochs

For the two models with different numbers of traces for validation, very similar results were obtained. In fact, for the validation in both models converges close to 110 for the Guess-Entropy metric. It would be interesting to train more models, but due to limited time it has not been possible. The approximate training time for 1,000 epochs has resulted in four to six hours.

Once trained, the testing phase was carried out to analyse the behaviour of the models:

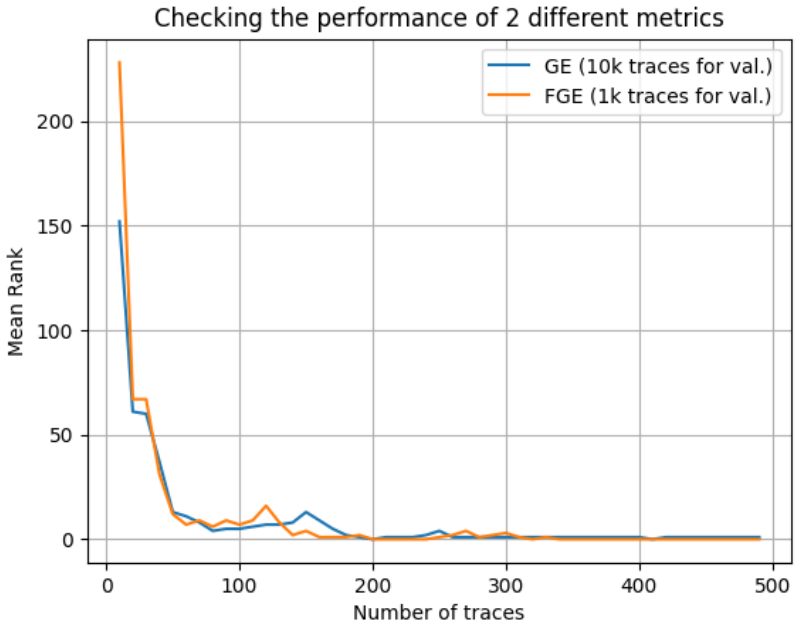

Figure 50: Testing results of trained models with GE and FGE duing 1,000 epochs

The Mean Rank averaged for GE is **9.10** and the Mean Rank averaged for FGE is **10.25**. The difference is not very distinguishable, so therefore the conclusion is that it is not an improvement a priori. Further experiments with FGE should be carried out.

Due to the unexpected behaviour of GE, will no be able to apply this technique (basically it would not make sense to apply GE as a monitoring metric for early-stopping when it does not have a good behaviour due to the configuration of the model). Even so, in the scripts it is possible to configure the training of the models to use early-stopping with the Guess Entropy or FGE metric, adjusting the number of traces for validation to 1k.

4 Results

4.1 Final Stage

In the First Stage (3.2), different models were trained with diverse configurations for different sets of traces as input data.

For the synchronised traces, after training different MLP configurations, very accurate results were obtained with 6 layers, with a configuration of 320 neurons, training with the RMSprop optimiser and a learning rate of 1e-5. The chosen model was compared by replacing the cost function with RL and no improvement was obtained.

For the desynchronised 50 time unit traces, after training different MLP configurations, very accurate results were obtained with 5 layers, with a configuration of 512 neurons, training with the RMSprop optimiser and a learning rate of 8e-6. The chosen model was compared by replacing the cost function with RL and no improvement was obtained.

For the desynchronised 100 time unit traces, after training different MLP configurations, very accurate results were obtained with 6 layers, with an L-S-L neuron model configuration, training with the Adagrad optimiser and a learning rate of 1e-5. The chosen model was compared by replacing the cost function by RL and the results obtained during the testing phase were improved using Mean Rank, obtaining a value of 24.064 Mean Rank averaged when training with a 4-layer MLP model, L-S-L neuron model, training with the Adagrad model and a learning rate of 1e-5, below the value obtained of 31.422 when training with cross-entropy loss function.

Based on these results, it is necessary to highlight the importance of synchronising the traces obtained from the devices with information to be exploited, as this facilitates the learning of the neural networks. Even so, it is also important to note the importance of modifying the different parameters to be configured when training with ANN, as there are large variations.

It is possible to conclude that the proposed function in [2] can be an improvement when training the different ANN models. Even so, it would be necessary to carry out further training to check the correct functioning of this technique, since an improvement was only obtained for the desynchronised traces.

As for the experiments carried out with the GE metric and the technique proposed in [3], the FGE, it has been found that the trained models do not have the expected behaviour. This may be due to the MLP typology used or the training dataset not having the right properties to allow the correct learning of the networks.

Due to the behaviour of the trained models, the FGE technique is not considered as a metric for the early-stopping technique. However, the fact that it is indeed a very important technique for reducing the training time it is not discarded, and further investigation.

It is concluded that the GE metric makes it easier for users to know the behaviour of the ANN, being more useful than using Accuracy.

In order to replicate the experiments, Python scripts are provided with the RL cost function added and the early-stopping technique, in addition to being able to use the Accuracy or GE metric.

5 Budget

This project has been carried out thanks to funding from **Applus+ Laboratories**, contributing the working hours of a junior electronic engineer as well as providing the work equipment. The following table shows a breakdown of the total amount of funding to carry out the work.

Concepts	Euros	Details	Total
Junior Electronic Engineer	€30/h	450h	€13,500
HP Probook 640 G3	€500	1	€500
Total price			€14,000

Table 1: Budget breakdown

6 Conclusions and future development

After performing a series of training sessions modifying the ANN parameters, the results obtained were studied in order to select a good model for predicting the value of the bytes.

After a large number of trainings for synchronised traces, desynchronised 50 time unit traces, and desynchronised 100 time unit traces, optimised models have been obtained for each type of trace, improving the results obtained in [1] through very similar models based on a more accurate search of their optimal parameters (see Section 3.2 for further details).

These models have been trained with the Cross-Entropy loss function, and subsequently replaced by the new Ranking Loss loss function proposed in [2] to check if the proposed technique does improve the training of the models.

In order not to discriminate the new function to a single model, 5 different models have been trained with the new Ranking Loss loss function and the results have been checked. For the synchronised and desynchronised 50 time unit traces the results could not be improved, while for the desynchronised 100 time unit traces the results could be improved.

Based on the results obtained with the new loss function, it can be concluded that it is a technique that improves the ANN training applied to SCA.

As for the FGE technique proposed in [3], it has not been possible to conclude if it implies an improvement in the training due to the behaviour of the models trained in this thesis. The authors of [3] worked with the CNN network topology, while this thesis is based on MLP topology and the results have differed. Therefore, it cannot be concluded whether this technique can improve the training of users using ANN applied to SCA.

Due to the limited time to complete this thesis and the resources available to carry out the training, the experiments have been limited to a single network topology and some specific configurations. As future work, it would be interesting to perform the same experiments but using other networks, for example CNN or other topologies, and to contrast the results with MLP.

As it has been proved, Ranking Loss is an improvement for some models. As future work it would also be useful to be able to carry out a more extensive search for optimal models with this technique.

References

[1] Emmanuel Prouff et al. *Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database. IACR Cryptol. ePrint Arch.*, 2018:53, 2018.

[2] Zaid et al. *Ranking Loss: Maximizing the Success Rate in Deep Learning Side-Channel Analysis. IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2021(1):25–55, 2020.

[3] Guilherme Perin, Lichao Wu, and Stjepan Picek. *The Need for Speed: A Fast Guessing Entropy Calculation for Deep Learning-based SCA*. 2021.

[4] Debayan D. et al. *STELLAR: A Generic EM Side-Channel Attack Protection through Ground-Up Root-cause Analysis*. page 11, 2019.

[5] Facundo B. et al. *Prediction of wind pressure coefficients on building surfaces using Artificial Neural Networks. Energy and Buildings*, 158, 2017.

[6] History about artificial neural network on wikipedia. https://en.wikipedia.org/wiki.

[7] Frank Rosenblatt. *The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review*, 65 6:386–408, 1958.

[8] Mike A. Perceptrón: ¿qué es y para qué sirve? https://datascientest.com/es/perceptron-que-es-y-para-que-sirve, 2022.

[9] Rahul Jayawardana and Thusitha Bandaranayake. *Analysis of optimizing neural networks and artificial intelligent models for guidance, control, and navigation systems. International Research Journal of Modernization in Engineering, Technology and Science*, page 745, 2021.

[10] Kevin J. Lang, Alex H. Waibel, and Geoffrey E. Hinton. *A time-delay neural network architecture for isolated word recognition. Neural Networks*, 3(1):23–43, 1990.

[11] Seppo Linnainmaa. *Taylor expansion of the accumulated rounding error. BIT*, 16:146–160, 1976.

[12] Long. Convolutional neural network. https://medium.com/@Aj.Cheng/convolutional-neural-network-d9f69e473feb, 2017.

[13] Leonardo Noriega. *Multilayer Perceptron Tutorial*. School of Computing, Staffordshire University, 2005.

[14] Pseudonym: Zhoulixue. Neural network and deeplearning (1.1). https://www.cnblogs.com/zhoulixue/p/6489724.html, 2016.

[15] Ken Hoffman. Machine learning: How to prevent overfitting. https://medium.com/swlh/machine-learning-how-to-prevent-overfitting-fdf759cc00a9, 2021.

[16] Divish et al. Rengasamy. *Deep Learning with Dynamically Weighted Loss Function for Sensor-Based Prognostics and Health Management. Sensors*, 20(3), 2020.

[17] Paul C. Kocher. *Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems*. Springer Berlin Heidelberg, 1996.

[18] R.L. et al. Rivest. *A Method for Obtaining Digital Signatures and Public-Key Cryptosystems*. 1978.

[19] W. Diffie and M. Hellman. New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654, 1976.

[20] National Institute of Standard and Technology (NIST). *Advanced Encryption Standard (AES)*. 2001.

[21] National Institute of Standard and Technology (NIST). *Data Encryption Standard (DES)*. 1999.

[22] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Advances in Cryptology — CRYPTO' 99*, pages 388–397. Springer Berlin Heidelberg, 1999.

[23] E. Brier et al. *Correlation Power Analysis with a Leakage Model*. 2004.

[24] Dakshi et. al. Agrawal. The em side—channel(s). In *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 29–45. Springer Berlin Heidelberg, 2003.

[25] S. Chari et al. *Template Attacks*. 2003.

[26] Benjamin Timon. *Non-Profiled Deep Learning-Based Side-Channel Attacks*. 2018.

[27] W. Schindler et al. *A stochastic model for differential side channel cryptanalysis*. 2005.

[28] Gabriel Hospodar et al. *Machine learning in side-channel analysis: a first study*. 2011.

[29] Gierlichs Benedikt et al. *Mutual Information Analysis*. 2008.

[30] Timon B. Non-profiled deep learning side-channel attacks with sensitivity analysis. https://eshard.com/posts/ches2019, 2019.

[31] François-Xavier Standaert and François Koeune. How to compare profiled side-channel attacks? In *Applied Cryptography and Network Security*, pages 485–498. Springer Berlin Heidelberg, 2009.

[32] François Durvaux and Marc Durvaux. *SCA-Pitaya: A Practical and Affordable Side-Channel Attack Setup for Power Leakage–Based Evaluations. Association for Computing Machinery*, 1(1), 2020.

[33] Altaf Mulani and Pradeep Mane. *High-Speed Area-Efficient Implementation of AES Algorithm on Reconfigurable Platform*, page 5. 2019.

[34] Mehdi-Laurent et al. Akkar. Power analysis, what is now possible... In *Advances in Cryptology — ASIACRYPT 2000*, pages 489–502. Springer Berlin Heidelberg, 2000.

[35] Thomas esserges, Ezzy Dabbish, and Robert Sloan. *Investigations of Power Analysis Attacks on Smartcards*. 1999.

[36] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[37] Trevor Hastie et al. *The Elements of Statistical Learning*. Springer New York, 2nd Edition, 2017.

[38] Benjamin Hettwer et. al. *Applications of machine learning techniques in side-channel attacks: a survey*. Journal of Cryptographic Engineering 10, 135-162, 2020.

[39] Ryad et. al. Benadjila. *Deep learning for side-channel analysis and introduction to ASCAD database*. Journal of Cryptographic Engineering 10, 163-188, 2020.

[40] Stjepan Picek, Annelie Heuser, Alan Jovic, Shivam Bhasin, and Francesco Regazzoni. The curse of class imbalance and conflicting metrics with machine learning for side-channel evaluations. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2019:209–237, 2018.

[41] Lutz Prechelt. *Early Stopping - But When?* 2000.

[42] Imperial college machine learning - neural networks. https://www.doc.ic.ac.uk/ nuric/teaching/imperial-college-machine-learning-neural-networks.html, 2022.

[43] Christof Paar and Jan Pelzl. *Understanding Cryptography : a Textbook for Students and Practitioners*. pages 87–117. Springer Berlin Heidelberg, 2009.

Appendices

A AES Definition

The Advanced Encryption Standard algorithm was proposed by Vincent Rijmen y Joan Daemen for a public contest organized by the National Institute of Standards and Technology (NIST) [20]. Also known as Rijndael, such as the names of the authors, is a block-cipher and symmetric-key algorithm. Uses the same key as well as to cipher and decipher. Nowadays, AES is the most implemented symmetric cryptographic algorithm.

The different phases and operation modes of the AES algorithm are explained in this appendix. The explanation is based on the Christof Paar and Jan Pelzl book [43].

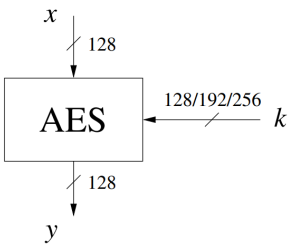

Figure A.51: AES general block schematic. Adapted from [43]

The first point to review is the length of the plaintext and the key. The plaintext consists of 16 bytes (128 bits) and the key length is variable. If the length of the key is 16 bytes (128 bits), the number of rounds is 10 and the subkeys needed for each round are 11. In the case of 24 bytes (192 bits), there are 12 rounds and 13 subkeys. For 32 bytes (256 bits), there are 14 rounds and 15 subkeys. The subkey transformation depends on the key length and consequently on the number of rounds.

A.1 Encryption

The encryption mode consists of different steps. The first step is the **Key Addition Layer** where the plaintext is XORed with the Transform 0 of the key defined. After the Key Addition Layer, round execution starts which of consists of: **Byte Substitution Layer**, **Shift Rows Layer**, **Mix Column Layer**, and another **Key Addition Layer** before continue with the next round. When all the rounds less the last round has been completed, the procedure of this last round is different. The layers change and the procedure is: **Byte Substitution Layer**, **Shift Rows Layer**, and the last **Key Addition Layer** with the latest key transformation. Once this last round has been completed, the plaintext has already been encrypted. AES encryption mode follows the next scheme:

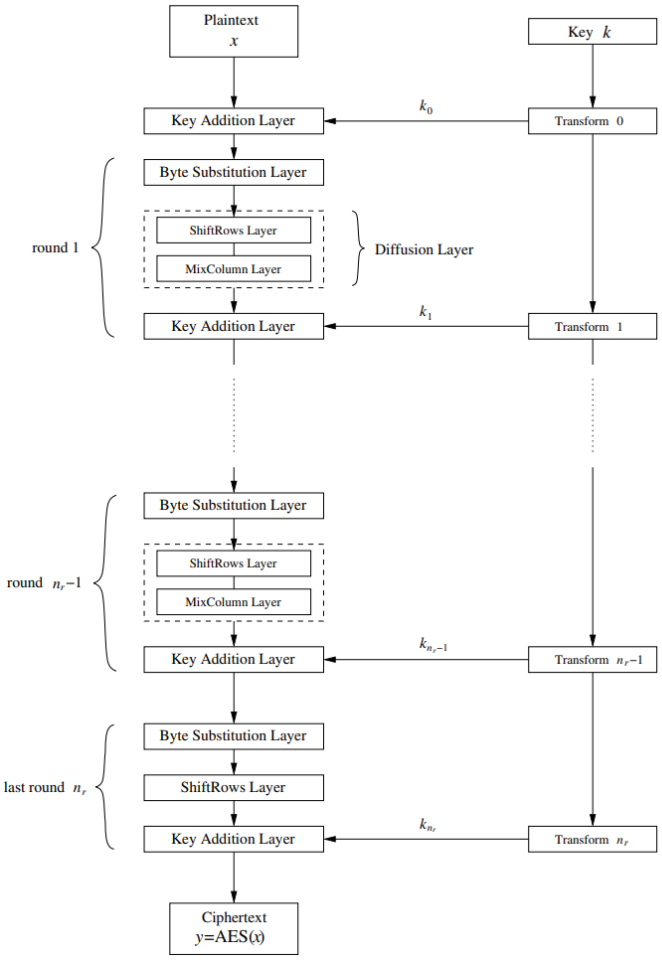

Figure A.52: AES encryption general scheme. Adapted from [43]

Before introducing the different layers, it is necessary to introduce the key schedule. The Key Schedule consists of generating from the initial key each subkey used for each round in **Key Addition Layer**. The total number of generated subkeys is equal to the number of rounds based on the key length plus one. Thus, for a length of 128 bits, the number of subkeys required is 11. For 192 bits, 13 subkeys are required. For 256, 15 subkeys. It is important to note that these subkeys are generated using words, i.e. 4 bytes in 4 bytes.

The subkeys are generated as follows, using the function g which is bound:

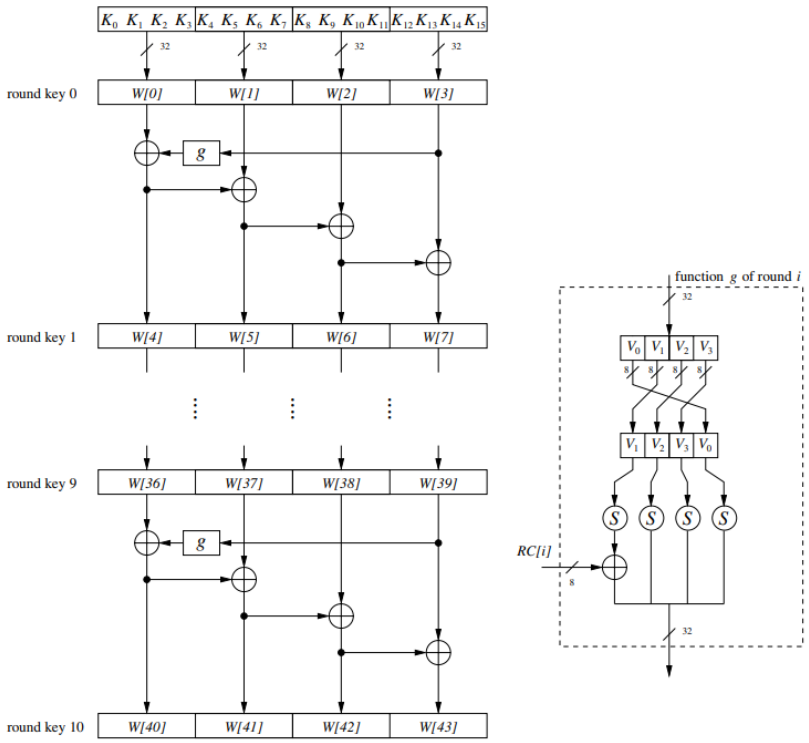

Figure A.53: AES key schedule scheme. Adapted from [43]

Once we have introduced how the subkeys are generated, we are going to see how the other layers that we find in the rounds of the algorithm work. In the following image we can see the modifications to which the data is subjected by each of the layers, as follows:

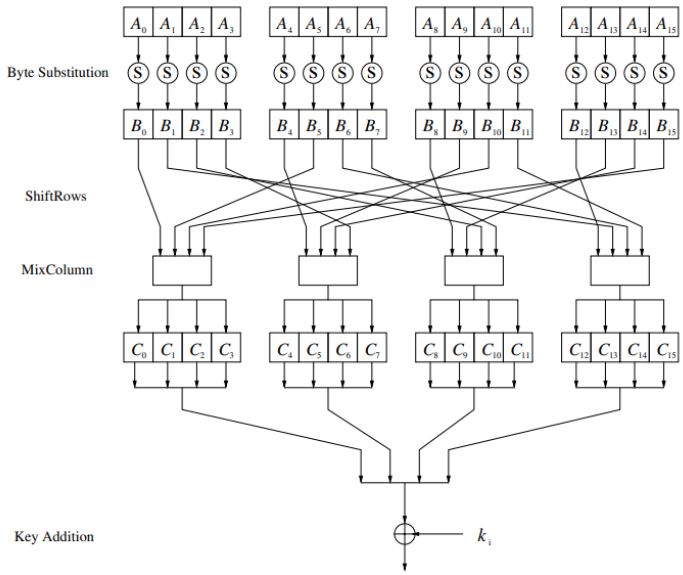

Figure A.54: AES encryption: internal layers structure scheme. Adapted from [43]

- **Byte Substitution Layer**: using defined matrices called S-BOXES, the input bytes are replaced by the byte specified in the column and row that the input byte defines. The first four bits of the input byte define the row and the other four bits define

the column, thus selecting the output byte. It is important to note that this is a non-linear function and that the inverse operation is performed with the inverse of the S-BOX matrix itself. The matrix S-BOX is as follows:

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

Figure A.55: AES S-BOX Matrix. Adapted from [43]

- **Shift Rows Layer**: together with **Mix Column**, generates data diffusion, an important feature for a cryptographic algorithm to be secure. This layer consists of applying left shifts. As can be seen in Figure A.54, the positions of the bytes [0...15] have been modified because they are subjected to the shifts. The operation is as follows:

B_0 B_4 B_8 B_{12}	no shift
B_5 B_9 B_{13} B_1	⟵ one position left shift
B_{10} B_{14} B_2 B_6	⟵ two positions left shift
B_{15} B_3 B_7 B_{11}	⟵ three positions left shift

Figure A.56: AES Shift Rows internal layer. Adapted from [43]

- **Mix Column Layer**: this layer consists of a linear transformation which mixes each column of the matrix resulting from the output of **Shift Rows Layer**, multiplying columns by the matrix below:

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix} = \begin{pmatrix} 02\ 03\ 01\ 01 \\ 01\ 02\ 03\ 01 \\ 01\ 01\ 02\ 03 \\ 03\ 01\ 01\ 02 \end{pmatrix} \begin{pmatrix} B_0 \\ B_5 \\ B_{10} \\ B_{15} \end{pmatrix}.$$

Figure A.57: AES Mix Column internal layer. Adapted from [43]

- **Key Addition Layer**: this layer consists of performing an XOR operation on the subkeys generated with the output of the **Mix Column Layer**.

A.2 Decryption

The decryption mode is very similar, except that the order of the rounds is reversed. The number of rounds is defined in the same way as for encryption, depending on the length of the key used. If we look at Figure A.58, we can see how the order of the subkeys is

inverted and **Inv Byte Substitution Layer**, **Inv Shift Rows Layer** and **Inv Mix Column Layer** are used, inverting in most cases the matrices defined in the encryption.

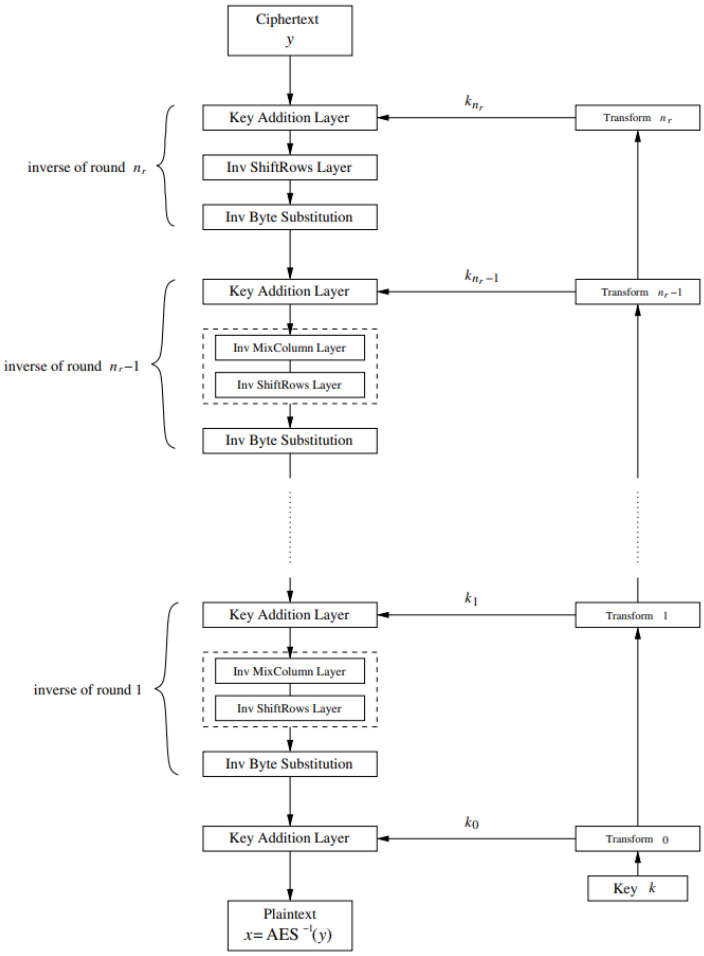

Figure A.58: AES decryption general scheme

For the **Key Addition Layer** in decryption it is the same procedure but in reverse, therefore the subkeys generated during encryption, are the same as the decryption but in reverse order.

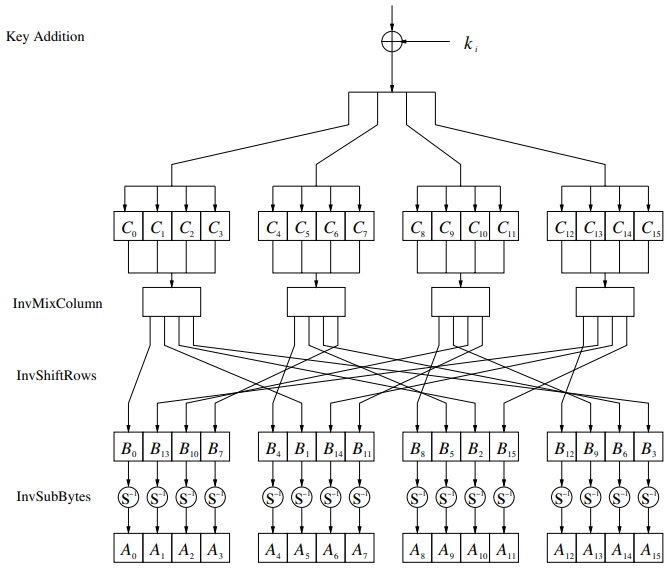

Figure A.59: AES decryption: internal layers structure scheme

- **Inv. Byte Substitution Layer**: similar to Byte Sub. in encryption mode, it uses defined matrices called INV S-BOXES, where the input bytes are replaced by the byte specified in the column and row that the input byte defines. The first four bits of the input byte define the row and the other four bits define the column, thus selecting the output byte. It is important to note that this is a non-linear function. The matrix INV S-BOX is as follows:

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

Figure A.60: AES Inv. S-BOX Matrix. Adapted from [43]

- **Inv. Shift Rows Layer**: the inverse layer of **Shift Rows Layer**. This layer consists of applying left shifts. As can be seen in Figure A.61, the positions of the bytes [0...15] have been modified because they are subjected to the shifts. The operation is as follows:

B_0	B_4	B_8	B_{12}	no shift
B_{13}	B_1	B_5	B_9	$\longrightarrow$ one position right shift
B_{10}	B_{14}	B_2	B_6	$\longrightarrow$ two positions right shift
B_7	B_{11}	B_{15}	B_3	$\longrightarrow$ three positions right shift

Figure A.61: AES Inv. Shift Rows internal layer. Adapted from [43]

- **Inv. Mix Column Layer**: the inverse layer of **Mix Column Layer**. this layer consists of a linear transformation which mixes each column of the matrix resulting from the output of **Inv. Shift Rows Layer**, multiplying columns by the matrix below:

$$\begin{pmatrix} B_0 \\ B_1 \\ B_2 \\ B_3 \end{pmatrix} = \begin{pmatrix} 0E & 0B & 0D & 09 \\ 09 & 0E & 0B & 0D \\ 0D & 09 & 0E & 0B \\ 0B & 0D & 09 & 0E \end{pmatrix} \begin{pmatrix} C_0 \\ C_1 \\ C_2 \\ C_3 \end{pmatrix}$$

Figure A.62: AES Inv. Mix Column internal layer. Adapted from [43]

- **Key Addition Layer**: the same of encryption mode. This layer consists of performing an XOR operation on the subkeys generated with the output of the **Inv. Mix Column Layer**.

B Experimental tables

B.1 MLP - Synchronised traces results

Epochs	Batch size	Layers	Neurons	Optimizer	LR	LF	MR averaged
			FIRST STAGE				
			MLP Configuration (ASCAD Sync)				
200	**100**	**6**	**200-...-256**	**RMSprop**	**1.00E-05**	**CE**	**3.445**
200	100	6	200-...-256	ADAM	1.00E-05	CE	13.082
200	100	3	200-200-256	RMSprop	1.00E-05	CE	1.404
200	100	4	200-...-256	RMSprop	1.00E-05	CE	36.771
200	100	5	200-...-256	RMSprop	1.00E-05	CE	6.299
200	100	**6**	200-...-256	RMSprop	1.00E-05	CE	1.456
200	100	7	200-...-256	RMSprop	1.00E-05	CE	18.421
200	100	8	200-...-256	RMSprop	1.00E-05	CE	1.991
200	100	6	64-...-256	RMSprop	1.00E-05	CE	126.670
200	100	6	128-...-256	RMSprop	1.00E-05	CE	1.535
200	100	6	200-...-256	RMSprop	1.00E-05	CE	1.456
200	100	6	256-...256	RMSprop	1.00E-05	CE	2.366
200	100	6	**320-...-256**	RMSprop	1.00E-05	CE	0.348
200	100	6	512-...-256	RMSprop	1.00E-05	CE	0.875
200	100	6	L-S-L*	RMSprop	1.00E-05	CE	38.930
200	100	6	S-L-S*	RMSprop	1.00E-05	CE	0.599
200	100	6	320-...-256	**RMSprop**	1.00E-05	CE	0.348
200	100	6	320-...-256	ADAM	1.00E-05	CE	1.067
200	100	6	320-...-256	SGD	1.00E-05	CE	142.994
200	100	6	320-...-256	Adadelta	1.00E-05	CE	132.775
200	100	6	320-...-256	Adagrad	1.00E-05	CE	173.753
200	100	6	320-...-256	RMSprop	1.00E-03	CE	231.806
200	100	6	320-...-256	RMSprop	1.00E-04	CE	231.846
200	100	6	320-...-256	RMSprop	**1.00E-05**	CE	0.348
200	100	6	320-...-256	RMSprop	8.00E-06	CE	1.390
200	100	6	320-...-256	RMSprop	1.20E-05	CE	5.179
200	100	6	320-...-256	RMSprop	1.00E-06	CE	218.171
			SECOND STAGE				
200	100	6	320-...-256	RMSprop	1.00E-05	RL	217.304
200	100	5	320-...-256	RMSprop	1.00E-05	RL	184.733
200	100	6	256-...-256	RMSprop	1.00E-05	RL	204.983
200	100	6	320-...-256	Adam	1.00E-05	RL	114.037
200	100	6	320-...-256	RMSprop	1.20E-05	RL	96.849

Table 2: MLP sync. traces experimental results.

B.2 MLP - Desynchronised (50 time units) traces results

Epochs	Batch size	Layers	Neurons	Optimizer	LR	LF	MR averaged
FIRST STAGE							
MLP Configuration (ASCAD Desync50)							
200	**100**	**6**	**200-...-256**	**RMSprop**	**1.00E-05**	**CE**	**66.484**
200	100	6	200-...-256	ADAM	1.00E-05	CE	132.212
200	100	3	200-200-256	RMSprop	1.00E-05	CE	54.635
200	100	4	200-...-256	RMSprop	1.00E-05	CE	81.996
200	100	**5**	200-...-256	RMSprop	1.00E-05	CE	55.299
200	100	6	200-...-256	RMSprop	1.00E-05	CE	66.484
200	100	7	200-...-256	RMSprop	1.00E-05	CE	180.036
200	100	8	200-...-256	RMSprop	1.00E-05	CE	75.575
200	100	5	64-...-256	RMSprop	1.00E-05	CE	154.982
200	100	5	128-...-256	RMSprop	1.00E-05	CE	47.694
200	100	5	200-...-256	RMSprop	1.00E-05	CE	55.299
200	100	5	256-...256	RMSprop	1.00E-05	CE	62.197
200	100	5	320-...-256	RMSprop	1.00E-05	CE	125.421
200	100	5	**512-...-256**	RMSprop	1.00E-05	CE	33.934
200	100	5	L-S-L*	RMSprop	1.00E-05	CE	153.841
200	100	5	S-L-S*	RMSprop	1.00E-05	CE	82.839
200	100	5	320-...-256	**RMSprop**	1.00E-05	CE	33.934
200	100	5	320-...-256	ADAM	1.00E-05	CE	38.523
200	100	5	320-...-256	SGD	1.00E-05	CE	79.998
200	100	5	320-...-256	Adadecta	1.00E-05	CE	138.242
200	100	5	320-...-256	Adagrad	1.00E-05	CE	198.870
200	100	5	320-...-256	RMSprop	1.00E-03	CE	232.838
200	100	5	320-...-256	RMSprop	1.00E-04	CE	230.599
200	100	5	320-...-256	RMSprop	1.00E-05	CE	33.934
200	100	5	320-...-256	RMSprop	**8.00E-06**	CE	16.149
200	100	5	320-...-256	RMSprop	1.20E-05	CE	92.236
200	100	5	320-...-256	RMSprop	1.00E-06	CE	28.538
SECOND STAGE							
200	100	5	512-...-256	RMSprop	8.00E-06	RL	133.489
200	100	6	512-...-256	RMSprop	8.00E-06	RL	182.039
200	100	5	128-...-256	RMSprop	8.00E-06	RL	162.163
200	100	5	512-...-256	Adam	8.00E-06	RL	159.790
200	100	5	512-...-256	RMSprop	1.00E-06	RL	199.189

Table 3: MLP desync50 traces experimental results.

B.3 MLP - Desynchronised (100 time units) traces results

Epochs	Batch size	Layers	Neurons	Optimizer	LR	LF	MR averaged
FIRST STAGE							
MLP Configuration (ASCAD Desync100)							
200	100	6	200-...-256	RMSprop	1.00E-05	CE	214.750
200	**100**	**6**	**200-...-256**	**ADAM**	**1.00E-05**	**CE**	**125.951**
200	100	3	200-200-256	RMSprop	1.00E-05	CE	193.720
200	100	4	200-...-256	RMSprop	1.00E-05	CE	152.135
200	100	5	200-...-256	RMSprop	1.00E-05	CE	232.401
200	100	**6**	200-...-256	RMSprop	1.00E-05	CE	125.951
200	100	7	200-...-256	RMSprop	1.00E-05	CE	213.970
200	100	8	200-...-256	RMSprop	1.00E-05	CE	200.983
200	100	6	64-...-256	RMSprop	1.00E-05	CE	94.274
200	100	6	128-...-256	RMSprop	1.00E-05	CE	198.689
200	100	6	200-...-256	RMSprop	1.00E-05	CE	125.951
200	100	6	256-...256	RMSprop	1.00E-05	CE	129.061
200	100	6	320-...-256	RMSprop	1.00E-05	CE	146.483
200	100	6	512-...-256	RMSprop	1.00E-05	CE	232.666
200	100	6	**L-S-L***	RMSprop	1.00E-05	CE	101.198
200	100	6	S-L-S*	RMSprop	1.00E-05	CE	191.677
200	100	6	320-...-256	RMSprop	1.00E-05	CE	143.628
200	100	6	320-...-256	ADAM	1.00E-05	CE	101.198
200	100	6	320-...-256	SGD	1.00E-05	CE	235.313
200	100	6	320-...-256	Adadelta	1.00E-05	CE	134.612
200	100	6	320-...-256	**Adagrad**	1.00E-05	CE	31.422
200	100	6	320-...-256	RMSprop	1.00E-03	CE	204.311
200	100	6	320-...-256	RMSprop	1.00E-04	CE	84.417
200	100	6	320-...-256	RMSprop	**1.00E-05**	CE	31.422
200	100	6	320-...-256	RMSprop	8.00E-06	CE	74.347
200	100	6	320-...-256	RMSprop	1.20E-05	CE	87.804
200	100	6	320-...-256	RMSprop	1.00E-06	CE	213.793
SECOND STAGE							
200	100	6	L-S-L	Adagrad	1.00E-05	RL	205.787
200	100	4	L-S-L	Adagrad	1.00E-05	RL	24.064
200	100	6	200-...-256	Adagrad	1.00E-05	RL	43.900
200	100	6	L-S-L	Adam	1.00E-05	RL	227.044
200	100	6	L-S-L	Adagrad	8.00E-05	RL	184.200

Table 4: MLP desync100 traces experimental results.

C Python scripts

C.1 ASCAD_training.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import os.path
import h5py
import numpy as np
import time
import random
import matplotlib.pyplot as plt
import tensorflow as tf
from tkinter import *
from keras.models import load_model
from keras.models import Model, Sequential
from keras.layers import Flatten, Dense, Input, Conv1D, MaxPooling1D, AveragePooling1D,
    Activation
from keras import backend as K
from tensorflow.keras.optimizers import RMSprop, Adam, SGD, Adadelta, Adagrad
from keras.callbacks import ModelCheckpoint, EarlyStopping
from tensorflow.keras.utils import to_categorical

class color:
    PURPLE = '\033[95m'
    CYAN = '\033[96m'
    DARKCYAN = '\033[36m'
    BLUE = '\033[94m'
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
    RED = '\033[91m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
    END = '\033[0m'

class Logger(object):
    def __init__(self, name_file):
        self.terminal = sys.stdout
        self.log = open(str(name_file) + ".log", "a")

    def write(self, message):
        self.terminal.write(message)
        self.log.write(message)

    def flush(self):
        # this flush method is needed for python 3 compatibility.
        # this handles the flush command by doing nothing.
        # you might want to specify some extra behavior here.
        pass

def shuffle_data(profiling_x, label_y):
    l = list(zip(profiling_x, label_y))
    random.shuffle(l)
    shuffled_x, shuffled_y = list(zip(*l))
    shuffled_x = np.array(shuffled_x)
    shuffled_y = np.array(shuffled_y)
    return (shuffled_x, shuffled_y)

def loss_sca(score_vector, alpha_value=10, nb_class=256):
    # Rank loss function
    def ranking_loss_sca(y_true, y_pred):
        alpha = K.constant(alpha_value, dtype='float32')
        # Batch_size initialization
        y_true_int = K.cast(y_true, dtype='int32')
        batch_s = K.cast(K.shape(y_true_int)[0], dtype='int32')
        # Indexing the training set (range_value = (?,))
        range_value = K.arange(0, batch_s, dtype='int64')
        # Get rank and scores associated with the secret key (rank_sk = (?,))
        values_topk_logits, indices_topk_logits = tf.nn.top_k(score_vector, k=nb_class,
            sorted=True)  # values_topk_logits = shape(?, nb_class) ; indices_topk_logits
```

```
            = shape(?, nb_class)
        rank_sk = tf.where(tf.equal(K.cast(indices_topk_logits, dtype='int64'), tf.
            reshape(K.argmax(y_true_int), [tf.shape(K.argmax(y_true_int))[0], 1])))[:,1]
            + 1  # Index of the correct output among all the hypotheses (shape(?,))
        score_sk = tf.gather_nd(values_topk_logits, K.concatenate([tf.reshape(range_value
            , [tf.shape(values_topk_logits)[0], 1]), tf.reshape(rank_sk - 1, [tf.shape(
            rank_sk)[0], 1])]))  # Score of the secret key (shape(?,))
        # Ranking Loss Initialization
        loss_rank = 0
        for i in range(nb_class):
            # Score for each key hypothesis (s_i_shape=(?,))
            s_i = tf.gather_nd(values_topk_logits, K.concatenate([tf.reshape(range_value,
                [tf.shape(values_topk_logits)[0], 1]), i * tf.ones([tf.shape(
                values_topk_logits)[0], 1], dtype='int64')]))
            # Indicator function identifying when (i == secret key)
            indicator_function = tf.ones(batch_s) - (K.cast(K.equal(rank_sk - 1, i),
                dtype='float32') * tf.ones(batch_s))
            # Logistic loss computation
            logistic_loss = K.log(1 + K.exp(- alpha * (score_sk - s_i))) / K.log(2.0)
            # Ranking Loss computation
            loss_rank = tf.reduce_sum((indicator_function * logistic_loss)) + loss_rank
        return loss_rank / (K.cast(batch_s, dtype='float32'))
    # Return the ranking loss function
    return ranking_loss_sca

#### Training model
def train_model_ranking_loss(X_profiling, Y_profiling, X_test, Y_test, model,
    save_file_name, epochs, batch_size, max_lr, FGE_boolean, metric):
    # Save model every epoch
    save_model = ModelCheckpoint(save_file_name)
    callbacks=[save_model]
    print("\nYour defined model is:")
    model.summary()
    print("\nTraining in 3, 2, 1...\n")
    # Get the input layer shape
    input_layer_shape = model.get_layer(index=0).input_shape
    Reshaped_X_profiling, Reshaped_X_test = X_profiling.reshape((X_profiling.shape[0],
        X_profiling.shape[1])), X_test.reshape((X_test.shape[0], X_test.shape[1]))
    history = model.fit(x=Reshaped_X_profiling, y=to_categorical(Y_profiling, num_classes
        =256), batch_size=batch_size, epochs=epochs, verbose=1, callbacks=callbacks,
        validation_data=(Reshaped_X_test, to_categorical(Y_test, num_classes=256)))
    plot_metrics(history, "accuracy", save_file_name)
    plot_metrics(history, "loss", save_file_name)
    return history

# Checking for Sanity all path routes introduced
def check_file_exists(file_path):
    file_path = os.path.normpath(file_path)
    if os.path.exists(file_path) == False:
        print("Error: provided file path '%s' does not exist!" % file_path)
        sys.exit(-1)
    return

# MLP Configuration model (X layers of Y units) (ADAM, RMSprop or SGD) (Cross-entropy,
    Ranking Loss, Fast Guess Entropy)
def mlp_config(training_model, lossfunctionoption, input_dim=700):
    model = Sequential()
    print("\n>>> How many neurons do you want to use in input layer?")
    input_node = int(input())
    print("\n>>> How many hidden layers do you want to use?")
    layer_nb = int(input())
    print("\n>>> How many neurons do you want to use in hidden layers?\n1: Large-Small-
        Large (256-128-64-64-128-256 or 256-128-64-128-256)\n2: Small-Large-Small
        (256-512-1024-1024-512-256 or 256-512-1024-512-256)")
    node = input()
    node_str = node
    model.add(Dense(input_node, input_dim=input_dim, activation='relu'))
    if node == "1":
        if layer_nb == 3:
            print("Creating model L-S-L with 5 hidden layers...")
            node = 256
```

```
            node_str = "L-S-L"
            model.add(Dense(node/2, activation='relu'))
            model.add(Dense(node/4, activation='relu'))
            model.add(Dense(node/2, activation='relu'))
            model.add(Dense(256, activation='softmax'))
        else:
            print("Creating model L-S-L with 6 hidden layers...")
            node = 256
            node_str = "L-S-L"
            model.add(Dense(node/2, activation='relu'))
            model.add(Dense(node/4, activation='relu'))
            model.add(Dense(node/4, activation='relu'))
            model.add(Dense(node/2, activation='relu'))
            model.add(Dense(256, activation='softmax'))
    elif node == "2":
        if layer_nb == 3:
            print("Creating model S-L-S with 5 hidden layers...")
            node = 256
            node_str = "S-L-S"
            model.add(Dense(node*2, activation='relu'))
            model.add(Dense(node*4, activation='relu'))
            model.add(Dense(node*2, activation='relu'))
            model.add(Dense(256, activation='softmax'))
        else:
            print("Creating model S-L-S with 6 hidden layers...")
            node = 256
            node_str = "S-L-S"
            model.add(Dense(node*2, activation='relu'))
            model.add(Dense(node*4, activation='relu'))
            model.add(Dense(node*4, activation='relu'))
            model.add(Dense(node*2, activation='relu'))
            model.add(Dense(256, activation='softmax'))
    else:
        for i in range(layer_nb):
            model.add(Dense(node, activation='relu'))
        model.add(Dense(256, activation='softmax'))
    print("\n>>> What optimizer do you want to use?\n1: ADAM\n2: RMSprop\n3: SGD\n4:
        Adadelta\n5: Adagrad")
    optimizer = input()
    print("\n>>> What learning rate do you want to define?")
    lr = float(input())
    optimizer_aux = ""
    if optimizer == "1":
        optimizer = Adam(learning_rate=lr)
        optimizer_aux = "ADAM"
    elif optimizer == "2":
        optimizer = RMSprop(learning_rate=lr)
        optimizer_aux = "RMSprop"
    elif optimizer == "3":
        optimizer = SGD(learning_rate=lr)
        optimizer_aux = "SGD"
    elif optimizer == "4":
        optimizer = Adadelta(learning_rate=lr)
        optimizer_aux = "ADADELTA"
    elif optimizer == "5":
        optimizer = Adagrad(learning_rate=lr)
        optimizer_aux = "ADAGRAD"
    else:
        print("ERROR: Optimizer doesn't exist.")
    if lossfunctionoption == False:
        print("\n>>> What metric do you want to use for checking the model learning? (
            Please, select GE if do you want execute FGE)\n1: Accuracy\n2: Guess-Entropy"
            )
        model_metric = int(input())
    else:
        model_metric = 1
    training_model += str(layer_nb + 2) + "layers_" + str(node_str) + "neurons_" + str(
        optimizer_aux) + "_" + str(
        lr) + "_"
    if lossfunctionoption == False:
        if model_metric == 1:
```

```
            print(">>> Selecting accuracy such as training and validation phase metric...")
            model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=['accuracy'])
            training_model += "crossentropy_accuracy"
        else:
            print(">>> Selecting guess_entropy such as training and validation phase metric...")
            model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=[guess_entropy], run_eagerly=True)
            training_model += "crossentropy_guessentropy"
    elif lossfunctionoption == True:
        score_layer = model.layers[layer_nb+1].output
        print("\n>>> What alpha value do you want to use for Ranking Loss function?")
        alpha_value = float(input())
        if model_metric == 1:
            print(">>> Selecting accuracy such as training and validation phase metric...")
            model.compile(loss=loss_sca(score_layer, nb_class=256, alpha_value=alpha_value), optimizer=optimizer, metrics=['accuracy'])
            training_model += "rankingloss_crossentropy_accuracy"
    else:
        print("ERROR: Loss function doesn't exist.")
    training_model = training_model +".h5"
    return model, training_model, lr, model_metric

# CNN Configuration model (ADAM or RMSprop) (Cross-entropy, Ranking Loss, Fast Guess Entropy)
def cnn_config(training_model, lossfunctionoption, classes=256,input_dim=700):
    print("\n>>> How many #ConvBlocks do you want to use?")
    convBlocks = int(input())
    print("\n>>> How many #ConvLayers do you want to use?")
    convLayers = int(input())
    print("\n>>> How many Filters per Block do you want to use? (Format: 64,128,256,512,512)")
    filtersPerBlock = input()
    filtersPerBlock = filtersPerBlock.split(",")
    for i in range(len(filtersPerBlock)):
        filtersPerBlock[i] = int(filtersPerBlock[i])
    print("\n>>> What is the size of the Kernel (same for padding)?")
    kernelSize = int(input())
    padding = kernelSize
    print("\n>>> What type of Pooling Layers do you want to use??\n1. Average Pooling\n2. Max Pooling")
    optionPooling = int(input())
    input_shape = (input_dim,1)
    img_input = Input(shape=input_shape)
    for i in range(convBlocks):
    # Block n
        for j in range(convLayers):
            x = Conv1D(filtersPerBlock[i], kernelSize, activation='relu', padding='same', name='block'+str(i+1)+'_conv'+str(j+1))(img_input)
            if optionPooling == 1:
                x = AveragePooling1D(2, strides=2, name='block'+str(i+1)+'_pool')(x)
            elif optionPooling == 2:
                x = MaxPooling1D(2, strides=2, name='block'+str(i+1)+'_pool')(x)
    # Classification block
    x = Flatten(name='flatten')(x)
    x = Dense(4096, activation='relu', name='fc1')(x)
    x = Dense(4096, activation='relu', name='fc2')(x)
    x = Dense(classes, activation='softmax', name='predictions')(x)
    inputs = img_input
    # Creating the model...
    model = Model(inputs, x, name='cnn_config')
    #Choosing the optimizer (ADAM, RMSprop or SGD) and the learning rate
    print("\n>>> What optimizer do you want to use?\n1: ADAM\n2: RMSprop\n3: SGD")
    optimizer = input()
    print("\n>>> What learning rate do you want to use?")
    lr = float(input())
    optimizer_aux = ""
    if optimizer == "1":
```

```
        optimizer = Adam(learning_rate=lr)
        optimizer_aux = "ADAM"
    elif optimizer == "2":
        optimizer = RMSprop(learning_rate=lr)
        optimizer_aux = "RMSprop"
    elif optimizer == "3":
        optimizer = SGD(learning_rate=lr)
        optimizer_aux = "SGD"
    else:
        print("ERROR: Optimizer doesn't exist.")
    #Choosing the loss function (Cross-Entropy or Ranking Loss)
    if lossfunctionoption == False:
        print("\n>>> What metric do you want to use for checking the model learning? (
            Please, select GE if do you want execute FGE)\n1: Accuracy\n2: Guess-Entropy"
            )
        model_metric = int(input())
    else:
        model_metric = 1
    if lossfunctionoption == False:
        if model_metric == 1:
            print(">>> Selecting accuracy such as training and validation phase metric ...
                ")
            model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=[
                'accuracy'])
        else:
            print(">>> Selecting guess_entropy such as training and validation phase
                metric ...")
            model.compile(loss='categorical_crossentropy', optimizer=optimizer, metrics=[
                guess_entropy])
    elif lossfunctionoption == True:
        print("\n>>> What alpha value do you want to use for Ranking Loss function?")
        alpha_value = float(input())
        score_layer = Dense(classes, activation=None, name='score')(x)
        predictions = Activation('softmax')(score_layer)
        print(">>> Selecting accuracy such as training and validation phase metric ...")
        model.compile(loss=loss_sca(score_layer, nb_class=256, alpha_value=alpha_value),
            optimizer=optimizer, metrics=['accuracy'])
    else:
        print("ERROR: Loss function doesn't exist.")
    training_model += str(convBlocks)+"convBlocks_"+str(convLayers)+"convLayers_"+str(
        filtersPerBlock)+"filters_"+str(kernelSize)+"kernel_"+str(optimizer_aux)+"_"+str(
        lr)+".h5"
    return model, training_model, lr, model_metric

def plot_metrics(history_training_data, metric, save_name): #Metrics: 'accuracy', 'loss',
     'val_accuracy', 'val_loss'
    if metric == "accuracy":
        plt.plot(history_training_data.history['accuracy'])
        plt.plot(history_training_data.history['val_accuracy'])
        plt.title('Model accuracy')
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Training', 'Validation'], loc='upper left')
        save_file = "accuracy_metrics_of_" + str(save_name) + ".png"
        plt.savefig(save_file)
        plt.show()
    elif metric == "loss":
        plt.plot(history_training_data.history['loss'])
        plt.plot(history_training_data.history['val_loss'])
        plt.title('Model loss')
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Training', 'Validation'], loc='upper left')
        save_file = "loss_metrics_of_" + str(save_name) + ".png"
        plt.savefig(save_file)
        plt.show()
    elif metric == "guess_entropy":
        plt.plot(history_training_data.history['guess_entropy'])
        plt.plot(history_training_data.history['val_guess_entropy'])
        plt.title('Model for Custom Guess Entropy Metric')
        plt.ylabel('Guess entropy')
```

```
        plt.xlabel('Epoch')
        plt.legend(['Training', 'Validation'], loc='upper left')
        save_file = "guess_entropy_metrics_of_" + str(save_name) + ".png"
        plt.savefig(save_file)
        plt.show()

def load_sca_model(model_file):
    check_file_exists(model_file)
    try:
            model = load_model(model_file)
    except:
        print("Error: can't load Keras model file '%s'" % model_file)
        sys.exit(-1)
    return model

def load_ascad(ascad_database_file, load_metadata):
    check_file_exists(ascad_database_file)
    # Open the ASCAD database HDF5 for reading
    try:
        in_file  = h5py.File(ascad_database_file, "r")
    except:
        print("Error: can't open HDF5 file '%s' for reading (it might be malformed) ..."
            % ascad_database_file)
        sys.exit(-1)
    # Load profiling traces
    X_profiling = np.array(in_file['Profiling_traces/traces'], dtype=np.float64)
    # Load profiling labels
    Y_profiling = np.array(in_file['Profiling_traces/labels'])
    # Load attacking traces
    X_attack = np.array(in_file['Attack_traces/traces'], dtype=np.float64)
    # Load attacking labels
    Y_attack = np.array(in_file['Attack_traces/labels'])
    if load_metadata == False:
        return (X_profiling, Y_profiling), (X_attack, Y_attack)
    else:
        return (X_profiling, Y_profiling), (X_attack, Y_attack), (in_file['
            Profiling_traces/metadata'], in_file['Attack_traces/metadata'])

def train_model(X_profiling, Y_profiling, model, save_file_name, epochs, batch_size,
    multilabel, validation_split, FGE_boolean, metric):
    check_file_exists(os.path.dirname(save_file_name))
    save_model = ModelCheckpoint(save_file_name)
    callbacks=[save_model]
    early_stopping = 0
    print("\n>>> Do you want to use early_stopping technique?:\n1: Yes\n2: No")
    early_stopping = int(input())
    if early_stopping == 1:
        if FGE_boolean == False:
            if metric == 1:
                monitor_option = 'accuracy'
                print(">>> Accuracy has been selected as early_stopping monitor")
            else:
                monitor_option = guess_entropy
                print(">>> Guess Entropy has been selected as early_stopping monitor")
        else:
            monitor_option = guess_entropy
            print(">>> Guess Entropy has been selected as early_stopping monitor")

        print("\n>>> Introduce the 'patience' (number of epoch for executing
            early_stopping):")
        patience_option = int(input())
        callbacks.append(EarlyStopping(monitor=monitor_option, mode="min", patience=
            patience_option, verbose=1, restore_best_weights=True))
    print("\nYour defined model is:")
    model.summary()
    print("\nTraining in 3, 2, 1...\n")
    # Get the input layer shape
    if isinstance(model.get_layer(index=0).input_shape, list):
        input_layer_shape = model.get_layer(index=0).input_shape[0]
    else:
        input_layer_shape = model.get_layer(index=0).input_shape
```

```
    # Sanity check
    if input_layer_shape[1] != len(X_profiling[0]):
        print("Error: model input shape %d instead of %d is not expected ..." % (
            input_layer_shape[1], len(X_profiling[0])))
        sys.exit(-1)
    # Adapt the data shape according our model input
    if len(input_layer_shape) == 2:
        # This is a MLP
        Reshaped_X_profiling = X_profiling
    elif len(input_layer_shape) == 3:
        # This is a CNN: expand the dimensions
        Reshaped_X_profiling = X_profiling.reshape((X_profiling.shape[0], X_profiling.
            shape[1], 1))
    else:
        print("Error: model input shape length %d is not expected ..." % len(
            input_layer_shape))
        sys.exit(-1)
    if (multilabel==1):
        y=multilabel_to_categorical(Y_profiling)
    elif (multilabel==2):
        y=multilabel_without_permind_to_categorical(Y_profiling)
    else:
        y=to_categorical(Y_profiling, num_classes=256)
    start = time.time()
    history = model.fit(x=Reshaped_X_profiling, y=y, batch_size=batch_size, verbose = 1,
        validation_split=validation_split, epochs=epochs, callbacks=callbacks)
    end = time.time()
    total_time = end-start
    print("\nTotal time execution for training phase: " + str(total_time)[0:8] + "s.")
    plot_metrics(history, "loss", save_file_name)
    if metric == 1:
        plot_metrics(history, "accuracy", save_file_name)
    else:
        plot_metrics(history, "guess_entropy", save_file_name)
    return history

def guess_entropy(y_true, y_pred): #MORE QUICKLY
    index = 0
    probability = 0
    rank = 0
    ranking = []
    y_true = K.get_value(y_true)
    y_pred = K.get_value(y_pred)
    for i in range(y_pred.shape[0]): #Batch size
        index = np.where(y_true[i] == 1)
        probability = y_pred[i][index[0][0]]
        #Save point... lets continue
        y_pred[i] = np.sort(y_pred[i])
        y_pred[i] = y_pred[i][::-1]
        rank = np.where(y_pred[i] == probability)
        #print(str(index[0][0]) + ", " + str(rank[0][0]))
        ranking.append(rank[0][0])
    ranking = np.array(ranking)
    ranking_average = np.sum(ranking) / y_pred.shape[0]
    return ranking_average

def main(rankinglossoption):
    #To define the dataset
    print("\n>>> What type of ASCAD traces do you want to load?\n1: Synchronised\n2:
        Desynchronised (50 time units)\n3: Desynchronised (100 time units)")
    database = int(input())
    if database == 1: ascad_database = "ASCAD.h5"
    elif database == 2: ascad_database = "ASCAD_desync50.h5"
    elif database ==3: ascad_database = "ASCAD_desync100.h5"
    print("*** 50.000 traces has been loaded ***")
    #To define the training and validation traces
    FGE_boolean = False
    if rankinglossoption == False:
        print("\n>>> Do you want to use Fast Guess Entropy?\n>>> If you decide FGE, I
            need to modify the training traces to 98% and 2% for validation ...\n1: Yes\n2
            : No")
```

```
        option = int(input())
        if option == 2:
            print("\n>>> Introduce the percentage (%) of traces for training:")
            percentage_training = int(input())
            print("\n>>> Introduce the percentage (%) of traces for validation:")
            percentage_validation = int(input())
            training_traces = int((percentage_training/100) * 50000)
            validation_traces = int((percentage_validation/100) * 50000)
            print("Split results: " + str(training_traces) + " training traces and " +
                str(validation_traces) + " validation traces.")
            if (percentage_training+percentage_validation) != 100:
                print("ERROR. The percentages are incorrect. Adjusting percentage for
                    trainig 80% and 20% for validation ...")
                percentage_training = 80
                percentage_validation = 20
                training_traces = int((percentage_training / 100) * 50000)
                validation_traces = int((percentage_validation / 100) * 50000)
                print("Split results: " + str(training_traces) + " training traces and "
                    + str(validation_traces) + " validation traces.")
        else:
            print("Adjusting percentage for training 98% and 2% for validation ...")
            percentage_training = 98
            percentage_validation =  2
            training_traces = int((percentage_training/100) * 50000)
            validation_traces = int((percentage_validation/100) * 50000)
            print("Split results: " + str(training_traces) + " training traces and " +
                str(validation_traces) + " validation traces.")
            FGE_boolean = True
    else:
        print("\n>>> Introduce the percentage (%) of traces for training:")
        percentage_training = int(input())
        print("\n>>> Introduce the percentage (%) of traces for validation:")
        percentage_validation = int(input())
        training_traces = int((percentage_training / 100) * 50000)
        validation_traces = int((percentage_validation / 100) * 50000)
        print("Split results: " + str(training_traces) + " training traces and " + str(
            validation_traces) + " validation traces.")
        if (percentage_training + percentage_validation) != 100:
            print(
                "ERROR. The percentages are incorrect. Adjusting percentage for trainig
                    80% and 20% for validation ...")
            percentage_training = 80
            percentage_validation = 20
            training_traces = int((percentage_training / 100) * 50000)
            validation_traces = int((percentage_validation / 100) * 50000)
            print("Split results: " + str(training_traces) + " training traces and " +
                str(
                validation_traces) + " validation traces.")
    validation_split = float(percentage_validation/100)
    multilabel = 0
    train_len = 0
    #To define the epochs and the batch_size
    print("\n>>> Introduce the number of epochs to training the network:")
    epochs = int(input())
    print("\n>>> Introduce the batch size to training the network:")
    batch_size = int(input())
    #To define the ANN topology
    print("\n>>> Do you want to use Multi-layer Perceptron (MLP) or Convolutional Neural
        Network (CNN)?:\n1: MLP\n2: CNN")
    network_type = input()
    if network_type == "1": network_type = "mlp"
    elif network_type == "2": network_type = "cnn"
    else:
        network_type = "mlp"
        print("ERROR, defining MLP topology...\n")
    #To define the trained model name depends on the dataset loaded
    if database == 1:  training_model = "trained_model_" + str(network_type) + "_" + str(
        epochs) + "epochs_" + str(batch_size) + "batchsize_FGE" + str(FGE_boolean)+ "_"
    elif database == 2: training_model = "trained_model_desync50_" + str(network_type) +
        "_" + str(epochs) + "epochs_" + str(batch_size) + "batchsize_FGE" + str(
        FGE_boolean)+ "_"
```

```
    elif database == 3: training_model = "trained_model_desync100_" + str(network_type) +
        "_" + str(epochs) + "epochs_" + str(batch_size) + "batchsize_FGE" + str(
        FGE_boolean) + "_"
    print("\n*****_Dataset_" + color.BLUE + str(ascad_database) + color.END + "_loaded,_
        network_topology_set_to_" + color.RED + str(network_type).upper() + color.END + "
        ,_epochs_set_to_" + color.YELLOW + str(epochs) + color.END + "_and_batch_size_
        equal_to_" + color.YELLOW + str(batch_size) + color.END + "....*****")

    #To define the model depends on the ranking loss option
    if rankinglossoption == True:
        (X_profiling, Y_profiling), (X_attack, Y_attack), (plt_profiling, plt_attack) =
            load_ascad(ascad_database, True)
        X_profiling = X_profiling.astype('float32')
        if (network_type == "mlp"):
            best_model, training_model, lr, metric = mlp_config(training_model, True,
                input_dim=len(X_profiling[0]))
        elif (network_type == "cnn"):
            best_model, training_model, lr, metric = cnn_config(training_model, True,
                256, input_dim=len(X_profiling[0]))
        else:  # display an error and abort
            print("Error:_no_topology_found_for_network_'%s'_..." % network_type)
            sys.exit(-1);
    else:
        (X_profiling, Y_profiling), (X_attack, Y_attack) = load_ascad(ascad_database,
            False)
        if (network_type == "mlp"):
            best_model, training_model, lr, metric = mlp_config(training_model, False,
                input_dim=len(X_profiling[0]))
        elif (network_type == "cnn"):
            best_model, training_model, lr, metric = cnn_config(training_model, False,
                256, input_dim=len(X_profiling[0]))
        else:  # display an error and abort
            print("Error:_no_topology_found_for_network_'%s'_..." % network_type)
            sys.exit(-1);
    # Training for Cross-entropy Loss Function
    if (train_len == 0 and training_model.count("rankingloss") == 0):
        # load traces
        (X_profiling, Y_profiling), (X_attack, Y_attack) = load_ascad(ascad_database,
            False)
        train_model(X_profiling, Y_profiling, best_model, training_model, epochs,
            batch_size, multilabel, validation_split, FGE_boolean, metric)
    # Training for Ranking Loss Function
    elif(training_model.count("rankingloss") == 1):
        train_model_ranking_loss(X_profiling[:training_traces], Y_profiling[:
            training_traces], X_profiling[training_traces:], Y_profiling[training_traces
            :], best_model, training_model, epochs, batch_size, lr, FGE_boolean, metric)
    return training_model
```

C.2 ASCAD_testing.py

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
import os.path
import h5py
import numpy as np
import matplotlib.pyplot as plt
import time
import metrics
from ASCAD_training import loss_sca, guess_entropy
from tkinter import *
from tkinter.filedialog import askopenfilename
from keras.models import Sequential
from keras.models import load_model
from keras.layers import Dense

class color:
    PURPLE = '\033[95m'
    CYAN = '\033[96m'
    DARKCYAN = '\033[36m'
```

```
    BLUE = '\033[94m'
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
    RED = '\033[91m'
    BOLD = '\033[1m'
    UNDERLINE = '\033[4m'
    END = '\033[0m'

# The AES SBox that we will use to compute the rank
AES_Sbox = np.array([
        0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0
            xD7, 0xAB, 0x76,
        0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0
            xA4, 0x72, 0xC0,
        0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0
            xD8, 0x31, 0x15,
        0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0
            x27, 0xB2, 0x75,
        0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0
            xE3, 0x2F, 0x84,
        0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0
            x4C, 0x58, 0xCF,
        0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0
            x3C, 0x9F, 0xA8,
        0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0
            xFF, 0xF3, 0xD2,
        0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0
            x5D, 0x19, 0x73,
        0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0
            x5E, 0x0B, 0xDB,
        0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0
            x95, 0xE4, 0x79,
        0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0
            x7A, 0xAE, 0x08,
        0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0
            xBD, 0x8B, 0x8A,
        0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0
            xC1, 0x1D, 0x9E,
        0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0
            x55, 0x28, 0xDF,
        0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0
            x54, 0xBB, 0x16
        ])

# Two Tables to process a field multplication over GF(256): a*b = alog (log(a) + log(b)
    mod 255)
log_table=[ 0, 0, 25, 1, 50, 2, 26, 198, 75, 199, 27, 104, 51, 238, 223, 3,
    100, 4, 224, 14, 52, 141, 129, 239, 76, 113, 8, 200, 248, 105, 28, 193,
    125, 194, 29, 181, 249, 185, 39, 106, 77, 228, 166, 114, 154, 201, 9, 120,
    101, 47, 138, 5, 33, 15, 225, 36, 18, 240, 130, 69, 53, 147, 218, 142,
    150, 143, 219, 189, 54, 208, 206, 148, 19, 92, 210, 241, 64, 70, 131, 56,
    102, 221, 253, 48, 191, 6, 139, 98, 179, 37, 226, 152, 34, 136, 145, 16,
    126, 110, 72, 195, 163, 182, 30, 66, 58, 107, 40, 84, 250, 133, 61, 186,
    43, 121, 10, 21, 155, 159, 94, 202, 78, 212, 172, 229, 243, 115, 167, 87,
    175, 88, 168, 80, 244, 234, 214, 116, 79, 174, 233, 213, 231, 230, 173, 232,
    44, 215, 117, 122, 235, 22, 11, 245, 89, 203, 95, 176, 156, 169, 81, 160,
    127, 12, 246, 111, 23, 196, 73, 236, 216, 67, 31, 45, 164, 118, 123, 183,
    204, 187, 62, 90, 251, 96, 177, 134, 59, 82, 161, 108, 170, 85, 41, 157,
    151, 178, 135, 144, 97, 190, 220, 252, 188, 149, 207, 205, 55, 63, 91, 209,
    83, 57, 132, 60, 65, 162, 109, 71, 20, 42, 158, 93, 86, 242, 211, 171,
    68, 17, 146, 217, 35, 32, 46, 137, 180, 124, 184, 38, 119, 153, 227, 165,
    103, 74, 237, 222, 197, 49, 254, 24, 13, 99, 140, 128, 192, 247, 112, 7 ]
alog_table =[1, 3, 5, 15, 17, 51, 85, 255, 26, 46, 114, 150, 161, 248, 19, 53,
    95, 225, 56, 72, 216, 115, 149, 164, 247, 2, 6, 10, 30, 34, 102, 170,
    229, 52, 92, 228, 55, 89, 235, 38, 106, 190, 217, 112, 144, 171, 230, 49,
    83, 245, 4, 12, 20, 60, 68, 204, 79, 209, 104, 184, 211, 110, 178, 205,
    76, 212, 103, 169, 224, 59, 77, 215, 98, 166, 241, 8, 24, 40, 120, 136,
    131, 158, 185, 208, 107, 189, 220, 127, 129, 152, 179, 206, 73, 219, 118, 154,
    181, 196, 87, 249, 16, 48, 80, 240, 11, 29, 39, 105, 187, 214, 97, 163,
    254, 25, 43, 125, 135, 146, 173, 236, 47, 113, 147, 174, 233, 32, 96, 160,
    251, 22, 58, 78, 210, 109, 183, 194, 93, 231, 50, 86, 250, 21, 63, 65,
```

```
    195, 94, 226, 61, 71, 201, 64, 192, 91, 237, 44, 116, 156, 191, 218, 117,
    159, 186, 213, 100, 172, 239, 42, 126, 130, 157, 188, 223, 122, 142, 137, 128,
    155, 182, 193, 88, 232, 35, 101, 175, 234, 37, 111, 177, 200, 67, 197, 84,
    252, 31, 33, 99, 165, 244, 7, 9, 27, 45, 119, 153, 176, 203, 70, 202,
    69, 207, 74, 222, 121, 139, 134, 145, 168, 227, 62, 66, 198, 81, 243, 14,
    18, 54, 90, 238, 41, 123, 141, 140, 143, 138, 133, 148, 167, 242, 13, 23,
    57, 75, 221, 124, 132, 151, 162, 253, 28, 36, 108, 180, 199, 82, 246, 1 ]

# Multiplication function in GF(2^8)
def multGF256(a,b):
    if (a==0) or (b==0): return 0
    else: return alog_table[(log_table[a]+log_table[b]) %255]

# Checking for Sanity all path routes introduced
def check_file_exists(file_path):
    file_path = os.path.normpath(file_path)
    if os.path.exists(file_path) == False:
        print("Error: provided file path '%s' does not exist!" % file_path)
        sys.exit(-1)
    return

def load_sca_model(model_file):
    check_file_exists(model_file)
    model = Sequential()
    model.add(Dense(200, input_dim=700, activation='relu'))
    model.add(Dense(256, activation='softmax'))
    score_layer = model.layers[1].output
    if model_file.endswith("rankingloss.h5"): model = load_model(filepath=model_file,
        custom_objects={'ranking_loss_sca': loss_sca(score_layer)})
    else: model = load_model(filepath= model_file, custom_objects={'guess_entropy':
        guess_entropy})
    return model

# Compute the rank of the real key for a give set of predictions
def rank(predictions, metadata, real_key, min_trace_idx, max_trace_idx,
    last_key_bytes_proba, target_byte, simulated_key):
    # Compute the rank
    if len(last_key_bytes_proba) == 0:
        # If this is the first rank we compute, initialize all the estimates to zero
        key_bytes_proba = np.zeros(256)
    else:
        # This is not the first rank we compute: we optimize things by using the
        # previous computations to save time!
        key_bytes_proba = last_key_bytes_proba
    for p in range(0, max_trace_idx-min_trace_idx):
        # Go back from the class to the key byte. '2' is the index of the byte (third
            byte) of interest.
        plaintext = metadata[min_trace_idx + p]['plaintext'][target_byte]
        key = metadata[min_trace_idx + p]['key'][target_byte]
        for i in range(0, 256):
            # Our candidate key byte probability is the sum of the predictions logs
            if (simulated_key!=1):
                proba = predictions[p][AES_Sbox[plaintext ^ i]]
            else:
                proba = predictions[p][AES_Sbox[plaintext ^ key ^ i]]
            if proba != 0:
                key_bytes_proba[i] += np.log(proba)
            else:
                # We do not want an -inf here, put a very small epsilon
                # that correspondis to a power of our min non zero proba
                min_proba_predictions = predictions[p][np.array(predictions[p]) != 0]
                if len(min_proba_predictions) == 0:
                    print("Error: got a prediction with only zeroes ... this should not
                        happen!")
                    sys.exit(-1)
                min_proba = min(min_proba_predictions)
                key_bytes_proba[i] += np.log(min_proba**2)
    # Now we find where our real key candidate lies in the estimation.
    # We do this by sorting our estimates and find the rank in the sorted array.
    sorted_proba = np.array(list(map(lambda a : key_bytes_proba[a], key_bytes_proba.
        argsort()[::-1])))
```

```
    real_key_rank = np.where(sorted_proba == key_bytes_proba[real_key])[0][0]
    return (real_key_rank, key_bytes_proba)

def full_ranks(predictions, dataset, metadata, min_trace_idx, max_trace_idx, rank_step,
    target_byte, simulated_key):
    print(">>> Computing rank for targeted byte {}...\n".format(target_byte))
    # Real key byte value that we will use. '2' is the index of the byte (third byte) of
        interest.
    if (simulated_key!=1):
        real_key = metadata[0]['key'][target_byte]
    else:
        real_key = 0
    # Check for overflow
    if max_trace_idx > dataset.shape[0]:
        print("Error: asked trace index %d overflows the total traces number %d" % (
            max_trace_idx, dataset.shape[0]))
        sys.exit(-1)
    index = np.arange(min_trace_idx+rank_step, max_trace_idx, rank_step)
    f_ranks = np.zeros((len(index), 2), dtype=np.uint32)
    key_bytes_proba = []
    for t, i in zip(index, range(0, len(index))):
        real_key_rank, key_bytes_proba = rank(predictions[t-rank_step:t], metadata,
            real_key, t-rank_step, t, key_bytes_proba, target_byte, simulated_key)
        f_ranks[i] = [t - min_trace_idx, real_key_rank]
    return f_ranks

def load_ascad(ascad_database_file, load_metadata=False):
    check_file_exists(ascad_database_file)
    # Open the ASCAD database HDF5 for reading
    try: in_file = h5py.File(ascad_database_file, "r")
    except:
        print("Error: can't open HDF5 file '%s' for reading (it might be malformed) ..."
            % ascad_database_file)
        sys.exit(-1)
    # Load profiling traces
    X_profiling = np.array(in_file['Profiling_traces/traces'], dtype=np.int8)
    # Load profiling labels
    Y_profiling = np.array(in_file['Profiling_traces/labels'])
    # Load attacking traces
    X_attack = np.array(in_file['Attack_traces/traces'], dtype=np.int8)
    # Load attacking labels
    Y_attack = np.array(in_file['Attack_traces/labels'])
    if load_metadata == False: return (X_profiling, Y_profiling), (X_attack, Y_attack)
    else: return (X_profiling, Y_profiling), (X_attack, Y_attack), (in_file['
        Profiling_traces/metadata'], in_file['Attack_traces/metadata'])

# Compute Pr(Sbox(p^k)*alpha|t)
def proba_dissect_beta(proba_sboxmuladd, proba_beta):
    proba = np.zeros(proba_sboxmuladd.shape)
    for j in range(proba_beta.shape[1]):
        proba_sboxdeadd = proba_sboxmuladd[:, [(beta^j) for beta in range(256)]]
        proba[:,j] = np.sum(proba_sboxdeadd*proba_beta, axis=1)
    return proba

# Compute Pr(Sbox(p^k)|t)
def proba_dissect_alpha(proba_sboxmul, proba_alpha):
    proba = np.zeros(proba_sboxmul.shape)
    for j in range(proba_alpha.shape[1]):
        proba_sboxdemul = proba_sboxmul[:, [multGF256(alpha,j) for alpha in range(256)]]
        proba[:,j] = np.sum(proba_sboxdemul*proba_alpha, axis=1)
    return proba

# Compute Pr(Sbox(p[permind]^k[permind])|t)
def proba_dissect_permind(proba_x, proba_permind, j):
    proba = np.zeros((proba_x.shape[0], proba_x.shape[2]))
    for s in range(proba_x.shape[2]):
        proba_1 = proba_x[:,:,s]
        proba_2 = proba_permind[:,:,j]
        proba[:,s] = np.sum(proba_1*proba_2, axis=1)
    return proba
```

```
# Check a saved model against one of the ASCAD databases Attack traces
def check_model(model_file, ascad_database, num_traces, target_byte, multilabel,
    simulated_key, save_file, parameter):
    check_file_exists(model_file)
    check_file_exists(ascad_database)
    # Load profiling and attack data and metadata from the ASCAD database
    (X_profiling, Y_profiling), (X_attack, Y_attack), (Metadata_profiling,
        Metadata_attack) = load_ascad(ascad_database, load_metadata=True)
    # Load model
    model = load_sca_model(model_file)
    # Get the input layer shape
    input_layer_shape = model.get_layer(index=0).input_shape[0]
    if isinstance(model.get_layer(index=0).input_shape, list):
        input_layer_shape = model.get_layer(index=0).input_shape[0]
    else:
        input_layer_shape = model.get_layer(index=0).input_shape
    # Sanity check
    if input_layer_shape[1] != len(X_attack[0, :]):
        print("Error: model input shape %d instead of %d is not expected ..." % (
            input_layer_shape[1], len(X_attack[0, :])))
        sys.exit(-1)
    # Adapt the data shape according our model input
    if len(input_layer_shape) == 2:
        # This is a MLP
        input_data = X_attack[:num_traces, :]
    elif len(input_layer_shape) == 3:
        # This is a CNN: reshape the data
        input_data = X_attack[:num_traces, :]
        input_data = input_data.reshape((input_data.shape[0], input_data.shape[1], 1))
    else:
        print("Error: model input shape length %d is not expected ..." % len(
            input_layer_shape))
        sys.exit(-1)
    # Predict our probabilities
    predictions = model.predict(input_data)
    if (multilabel!=0):
        if (multilabel==1):
            predictions_sbox = multilabel_predict(predictions)
        else:
            predictions_sbox = multilabel_without_permind_predict(predictions)
        for target_byte in range(16):
            ranks_i = full_ranks(predictions_sbox[target_byte], X_attack, Metadata_attack
                , 0, num_traces, 10, target_byte,       simulated_key)
            # We plot the results
            x_i = [ranks_i[i][0] for i in range(0, ranks_i.shape[0])]
            y_i = [ranks_i[i][1] for i in range(0, ranks_i.shape[0])]
            plt.plot(x_i, y_i, label="key "+str(target_byte))
        if model_file.count("desync50") == 1:
            plt.title('Performance of ' + model_file[23:26].upper() + ' in ' + model_file
                [27:30] + ' epochs and ' + model_file[37:40] + ' of batch size (checking
                ' + parameter + ')')
        elif model_file.count("desync100") == 1:
            plt.title('Performance of ' + model_file[24:27].upper() + ' in ' + model_file
                [28:31] + ' epochs and ' + model_file[38:41] + ' of batch size (checking
                ' + parameter + ')')
        else:
            plt.title('Performance of ' + model_file[14:17].upper() + ' in ' + model_file
                [18:21] + ' epochs and ' + model_file[28:31] + ' of batch size (checking
                ' + parameter + ')')
        plt.xlabel('Number of traces')
        plt.ylabel('Rank')
        plt.grid(True)
        plt.legend(loc='upper right')
        if (save_file != ""):
            plt.savefig(save_file)
        else:
            plt.show(block=False)
    else:
        predictions_sbox_i = predictions
      # We test the rank over traces of the Attack dataset, with a step of 10 traces
        ranks = full_ranks(predictions_sbox_i, X_attack, Metadata_attack, 0, num_traces,
```

```
            10, target_byte, simulated_key)
        # We plot the results
        x = [ranks[i][0] for i in range(0, ranks.shape[0])]
        y = [ranks[i][1] for i in range(0, ranks.shape[0])]

        if model_file.count("desync50") == 1:
            plt.title('Performance of ' + model_file[23:26].upper() + ' in ' + model_file
                [27:30] + ' epochs and ' + model_file[37:40] + ' of batch size (checking
                ' + parameter + ')')
        elif model_file.count("desync100") == 1:
            plt.title('Performance of ' + model_file[24:27].upper() + ' in ' + model_file
                [28:31] + ' epochs and ' + model_file[38:41] + ' of batch size (checking
                ' + parameter + ')')
        else:
            plt.title('Performance of ' + model_file[14:17].upper() + ' in ' + model_file
                [18:22] + ' epochs and ' + model_file[29:32] + ' of batch size (checking
                ' + parameter + ')')
        plt.xlabel('Number of traces')
        plt.ylabel('Mean Rank')
        save_file += "_Rank.png"
        plt.grid(True)
        aux = 0
        for i in range(len(x)):
            aux += y[i]
        aux = aux / len(x)
        print("Mean Rank averaged for " + save_file + " is: " + color.BOLD + str(aux) +
            color.END +"\n")
        plt.plot(x, y)
        if (save_file != ""):
            plt.savefig(save_file)
        else:
            plt.show(block=False)
        x.clear()
        y.clear()
        metrics.x_vector.append(y)
        metrics.y_vector.append(y)
        metrics.y_average.append(aux)

def main_original(model_name, parameter):
    #Default parameters values for testing
    ascad_database=traces_file="ASCAD.h5"
    num_traces=10000
    target_byte= 2
    multilabel=0
    simulated_key=0
    model_name = model_name[::-1][3:][::-1]
    save_file="results_of_testing_"+parameter+"_"+str(model_name)
    #Checking model
    print("\nTesting in 3, 2, 1...\n")
    start = time.time()
    check_model(model_name, ascad_database, num_traces, target_byte, multilabel,
        simulated_key, save_file, parameter)
    end = time.time()
    total_time = end - start
    print("Total time execution for testing phase: " + str(total_time)[0:8] + "s.\n")
    print(color.BOLD + "\n
        ##########################################################################\n" + color.
        END)

def automatic_main(model_saved):
    print(color.BOLD + "\n
        ##########################################################################\n" + color.
        END)
    parameters = []
    print("What is the variable to check? (#Layers, #Neurons, Optimizer, Learning Rate
        ...)")
    variable = input()
    print("What is the "+variable+" to test?")
    parameter = input()
    parameters.append(parameter)
    if model_saved.endswith('rankingloss.h5'):
```

```
        print("Waiting, this model has been trained with Ranking Loss ... I need to change
            the file structure ... ")
        model_saved = config_models(model_saved)
    model_saved += ".h5"
    main_original(model_saved, parameter)
    metrics.plot_all_ranks(variable, parameters)

def manual_main():
    print(color.BOLD + "\n
        ##########################################################################\n" + color.
        END)
    print("How many times do you want to execute a testing phase/testing models?")
    testing_phase = int(input())
    parameters = []
    print("What is the variable to check? (#Layers, #Neurons, Optimizer, Learning Rate
        ...)")
    variable = input()
    for i in range(testing_phase):
        print("Introduce the path to the trained model for testing (or enter to select in
            file explorer): ")
        model = input()
        if model == "":
            root = Tk()
            while model == "":
                model = askopenfilename()  # show an "Open" dialog box and return the
                    path to the selected file
                model = os.path.basename(model) # return only the file name of the
                    complete path
        root.destroy()
        if model.endswith('rankingloss.h5'):
            print("Waiting, this model has been trained with Ranking Loss ... I need to
                change the file structure ... ")
            model = config_models(model)
        print(str(model) + " LOADED SUCCESFULLY!\n")
        print("What is the "+variable+" to test?")
        parameter = input()
        parameters.append(parameter)
        model += ".h5"
        main_original(model, parameter)
    metrics.plot_all_ranks(variable, parameters)

def config_models(model):
# datetime object containing current date and time
    model_updated = model[::-1][3:][::-1]+'_updated.h5'
    with h5py.File(model_updated,'w') as f_dest:
        with h5py.File(model,'r') as f_src:
            with h5py.File('structure.h5', 'r') as f_example:
                #print(list(f_src.keys()))
                f_dest.attrs.create('backend',f_src.attrs['backend'])
                f_dest.attrs.create('keras_version', f_src.attrs['keras_version'])
                f_dest.attrs.create('model_config', f_src.attrs['model_config'])
                f_dest.attrs.create('training_config' ,f_src.attrs['training_config'])

                f_dest.copy(f_src['model_weights'], 'model_weights', name="model_weights"
                    ) #Copy all information in model_weights
                f_dest.copy(f_example['optimizer_weights'], 'optimizer_weights', name="
                    optimizer_weights") #Copy the folder optimizer_weights
                f_dest.copy(f_src['optimizer_weights']['training'], f_dest['
                    optimizer_weights']) #Copy all information in optimizer_weights
                f_dest['optimizer_weights']['training'].clear()
                del f_dest['optimizer_weights']['training']
                #print(list(f_dest['optimizer_weights'].keys()))
                f_example.close()
            f_src.close()
        f_dest.close()
    return model_updated
```

C.3 Metrics.py

```
import matplotlib.pyplot as plt
```

```
x_vector = []
y_vector = []
y_average = []

def legends(parameters):
    if len(parameters) == 1: plt.legend([parameters[0]])
    if len(parameters) == 2: plt.legend([parameters[0], parameters[1]])
    if len(parameters) == 3: plt.legend([parameters[0], parameters[1], parameters[2]])
    if len(parameters) == 4: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3]])
    if len(parameters) == 5: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3], parameters[4]])
    if len(parameters) == 6: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3], parameters[4], parameters[5]])
    if len(parameters) == 7: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3], parameters[4], parameters[5], parameters[6]])
    if len(parameters) == 8: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3], parameters[4], parameters[5], parameters[6], parameters[7]])
    if len(parameters) == 9: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3], parameters[4], parameters[5], parameters[6], parameters[7],
        parameters[8]])
    if len(parameters) == 10: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3], parameters[4], parameters[5], parameters[6], parameters[7],
        parameters[8], parameters[9]])
    if len(parameters) == 11: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3], parameters[4], parameters[5], parameters[6], parameters[7],
        parameters[8], parameters[9], parameters[10]])
    if len(parameters) == 12: plt.legend([parameters[0], parameters[1], parameters[2],
        parameters[3], parameters[4], parameters[5], parameters[6], parameters[7],
        parameters[8], parameters[9], parameters[10], parameters[11]])

def plot_all_ranks(variable, parameters):
    #Plot ranks
    plt.title('Checking the performance of ' + str(len(parameters)) + ' different ' +
        variable)
    plt.xlabel('Number of traces')
    plt.ylabel('Mean Rank')
    plt.grid(True)
    legends(parameters)
    plt.plot(x_vector, y_vector)
    plt.savefig("comparison_of_"+str(len(parameters))+"_"+variable+"
        _in_terms_of_MEAN_RANK.png")
    plt.show()

    #Plot ranks averaged
    plt.title('Checking the performance of ' + str(len(parameters)) + ' ' + variable)
    plt.xlabel(variable)
    plt.ylabel('Mean Rank (average for 10k traces)')
    plt.grid(True)
    legends(parameters)
    plt.plot(parameters, y_average)
    plt.savefig("comparison_of_"+str(len(parameters))+"_"+variable+"
        _in_terms_of_MEAN_RANK_AVERAGED.png")
    plt.show()
```

C.4 Main.py

```
import ASCAD_testing
import ASCAD_training
from tensorflow.python.framework.ops import disable_eager_execution

class color:
    PURPLE = '\033[95m'
    CYAN = '\033[96m'
    DARKCYAN = '\033[36m'
    BLUE = '\033[94m'
    GREEN = '\033[92m'
    YELLOW = '\033[93m'
```

```
    RED = '\033[91m'
    B = '\033[1m'
    UNDERLINE = '\033[4m'
    E = '\033[0m'

model_saved = ""

def print_init():
    print(color.B+color.BLUE+"\n
        ##############################################################################\n"+color.E)
    print(color.B+color.BLUE+"\t  ASCAD_Python_Scripts "+color.E+" modified by Cristian
        Fernandez\n\t\t  Only for academic purposes | UPC 2022")
    print(color.B+color.BLUE+"\n
        ##############################################################################\n"+color.E)
def print_testing():
    print(color.B+color.RED+"\n
        ##############################################################################"+color.E)
    print("\n\t\t\t\t Executing "+color.B+color.RED+" ASCAD_testing "+color.E+" script"+
        color.E)
    print(color.B+color.RED+"\n
        ##############################################################################\n"+color.E)
def print_training():
    print(color.B+color.YELLOW+"\n
        ##############################################################################"+color.E)
    print("\n\t\t\t\t Executing "+color.B+color.YELLOW+" ASCAD_training "+color.E+" script"
        +color.E)
    print(color.B+color.YELLOW+"\n
        ##############################################################################"+color.E)
def print_exit():
    print(color.B+"\n
        ##############################################################################\n"+color.E)
    print("Goodbye!!!")
    print(color.B+"\n
        ##############################################################################\n"+color.E)

print_init()
print(">>> Do you want train or test?\n1: Training\n2: Testing\n3: Exit")
option = int(input())

### Training phase ###
if option == 1:
    print_training()
    print("\n>>> Will you use Cross-Entropy loss function or Ranking Loss function?\n1:
        Cross-entropy\n2: Ranking Loss")
    option = int(input())
    if option == 2:
        print("Disabling eager execution for training models with Ranking Loss function
            ...")
        disable_eager_execution()
        rankinglossoption = True
    else: rankinglossoption = False
    model_saved = ASCAD_training.main(rankinglossoption)
    print("\n>>> Do you want to execute testing phase?\n1: YES\n2: NO")
    option_2 = int(input())
    if option_2 == 1:
        ### Testing phase ###
        print_testing()
        ASCAD_testing.automatic_main(model_saved)
        print_exit()
    else:
        print_exit()

### Testing phase ###
elif option == 2:
    print_testing()
    ASCAD_testing.manual_main()
    print_exit()

### Exit ###
else:
    print_exit()
```

Glossary

AES - Advanced Encryption Standard
S-BOX - Substitution-box
DES - Data Encryption Standard
ANN - Artificial Neural Network
SCA - Side Channel Analysis
CNN - Convolutional Neural Network
MLP - Multi-layer Perceptron
RL - Ranking Loss (loss function)
GE - Guess Entropy (metric)
FGE - Fast Guess Entropy
LR - Learning rate
LF - Loss function
L-S-L - Large-small-large
S-L-S - Small-large-small
MR - Mean Rank
SYNC - Synchronised
DESYNC - Desynchronised