Apprentissage automatique pour les attaques par canaux auxiliaires

Christophe Genevey-Metat

▶ **To cite this version:**

Christophe Genevey-Metat. Apprentissage automatique pour les attaques par canaux auxiliaires. Machine Learning [cs.LG]. Université de Rennes, 2023. English. NNT : 2023URENS030 . tel-04241537

HAL Id: tel-04241537
https://theses.hal.science/tel-04241537

Submitted on 13 Oct 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

THÈSE DE DOCTORAT DE

L'UNIVERSITÉ DE RENNES

ÉCOLE DOCTORALE N° 601
Mathématiques, Télécommunications, Informatique, Signal, Systèmes, Électronique
Spécialité : Informatique

Par
Christophe Genevey-Metat

Machine learning for side-channel analysis

Thèse présentée et soutenue à Rennes, le 28 Septembre 2023
Unité de recherche : IRISA

Rapporteurs avant soutenance :

Philippe Maurine — Associate professor, HDR, University of Montpellier
Jean-Luc Danger — Professor, TELECOM Paris

Composition du Jury :

Président :	Elisa Fromont	Professor, University of Rennes
Examinateur :	Louis Goubin	Professor, University of Versailles
Dir. de thèse :	Jean-Marc Jezequel	Professor, University of Rennes
Co-encadrant :	Benoit Gérard	Engineer, DGA-MI

Invité(s) :

Pierre-Alain Fouque — Professor, University of Rennes

ACKNOWLEDGEMENT

In this section, I wanted to thank several people who allowed me to do my PhD thesis.

First, I would like to thank my supervisors, Benoit Gérard and Annelie Heuser, with whom I have spent the most time over the past three years. During these last three years, Annelie and Benoit have given me a lot of knowledge and support. They trusted in me and the experiments I conducted since the beginning of my thesis, so I thank them for that. I would also like to thank Jean-Marc Jezequel who accepted to be my thesis director and with whom I had interesting exchanges during the meetings we had with my supervisors. I would also like to thank Pierre-Alain Fouque for taking the time to proofread my manuscript and giving me advice on how to improve it.

Secondly, I would like to thank the members of my jury: Clémentine Maurice, Elisa Fromont, Goubin Louis, Maurine Philippe, and Jean-Luc Danger. I would like to thank them already for agreeing to be part of my jury.

I did my thesis on different teams: the TAMIS team, the EMSEC team, and the CAPSULE team at IRISA and INRIA laboratories. I would like to thank all people from these teams for the discussion we had around a cup of coffee. In particular, Mathieu with whom I did sports during the lunch break, Celine and Yoann with whom I play games (like Go, Skulls), Phuc and Lamine with whom I had interesting discussions.

Next, I would like to thank my friends for their support. In particular, Clement is a long-time friend, who has always been there for me, and who has supported me in times of doubt. I would like to thank Malo who in times of stress allowed me to see new horizons. Also, I thank Evan, Nap and Leopold for their support during my PhD thesis.

Finally, I also would like to thank my family for their support during these three years. In particular, my mother has always had confidence in me and has always supported me in my different projects.

Résumé en français

Motivations

De nos jours, nous utilisons quotidiennement des algorithmes cryptographiques. Ces algorithmes sont présents dans nos vies courantes et nous permettent de payer avec nos cartes bancaires, aller sur internet ou envoyer un message de manière sécurisée. Les algorithmes cryptographiques sont de plus en plus répandus dans les systèmes embarqués afin de garantir un certain niveau de sécurité à leurs utilisateurs, ainsi nous pouvons par exemple payer sans contact avec notre carte bancaire sans crainte qu'un attaquant nous dérobe notre code bancaire. Avec l'émergence de "l'internet des objets", de plus en plus d'attention a été apportée aux attaques peuvant possiblement affecter les systèmes embarqués. Parmi ces attaques sur les systèmes embarqués, les "attaques par canaux auxiliaires" sont particulièrement craintes car non détectables par nature. Celles-ci ont été introduites en 1980 par Kocher [37]. Ce type d'attaque consiste à utiliser une grandeur physique telle que le temps d'exécution du programme sur un système, sa consommation électrique, les ondes électromagnétiques, le son ou la chaleur générés par le système embarqué, afin de découvrir de l'information. Ce sont toutes ces grandeurs auxquelles on fait référence quand on parle de "canaux auxiliaires".

Dans la plupart des cas, et c'est aussi le contexte retenu dans le cadre de cette thèse, les attaques par canaux auxiliaires sont utilises contre des algorithmes cryptographiques embarqués. Dans ce contexte, l'attaquant va cibler un algorithme cryptographique qui s'exécute sur un système embarqué afin de découvrir la clé utilisée par celui-ci. Dans la vie de tous les jours, une telle attaque pourrait permettre à un attaquant de découvrir le code bancaire d'une personne. La vulnérabilité exploitée par les attaques par canaux auxiliaires ne provient pas de la conception de l'algorithme en lui-même. En effet, même si l'algorithme est considéré comme inviolable du point de vue mathématique, l'implantation de cet algorithme peut laisser fuire des informations sur ses opérations internes et le secret utilisé via des canaux auxiliaires. Plus précisément, l'algorithme cryptographique va prendre en entrée un texte clair et produire en sortie un texte chiffré. Durant l'exécution de cet algorithme, le système va générer différentes valeurs intermédiaires qui vont fuire via les canaux auxiliaires. En effet, ces canaux auxiliaires sont directement liés à l'état interne du système embarqué et donc aux opérations effectuées. Un attaquant peut ainsi capturer plusieurs mesures qui proviennent des différents canaux auxiliaires, et utiliser des outils statistiques ou de l'intelligence artificielle afin de découvrir des informations sensibles utilisées par l'algorithme. Ainsi dans le cadre d'un algorithme cryptographique, l'attaquant peut directement

découvrir la clé ou, plus généralement, des valeurs intermédiaires utilisées par l'algorithme qui le mèneront, au final, à la clé.

D'autres attaques visent à perturber l'algorithme cryptographique, c'est par exemple le cas des attaques par injection de fautes dont le but est de contourner certaines opérations d'un algorithme pendant son exécution. Ce type d'attaque ne fait pas partie du périmètre de cette thèse. Nous nous concentrons uniquement sur des attaques ne modifiant pas le comportement de l'algorithme et se basant donc sur l'écoute et l'observation de celui-ci.

Les premières attaques par canaux auxiliaires, auxquelles on fera référence par le terme "attaques non-profilées" consistent à capturer un nombre limité de mesures et a effectué des analyses statistiques telles qu'un calcul de corrélation pour découvrir certaines informations. Ces attaques ont l'avantage d'utiliser un modèle générique pour exploiter la fuite d'information. Cet avantage fait que ces attaques sont toujours étudiées par la communauté scientifique, cependant elles sont maintenant peu efficaces face aux contre-mesures qui sont déployées. Les autres attaques par canaux auxiliaires, auxquelles on fera référence par le terme "attaques profilées" classiques consistent à capturer un plus grand nombre de mesures et à estimer le modèle de fuite d'information de différentes valeurs de clé à travers les traces. La plus connue des "attaques profilées" classiques est la "template attack". Ces attaques sont plus efficaces que les attaques "non profilées". Ces attaques nécessitent un ensemble de données étiquetées, c'est-à-dire que pour chaque trace (électromagnétique ou de consommation de courant) acquise, nous devons avoir connaissance de la clé utilisée par l'algorithme. Avec un ensemble de données étiquetées, un attaquant peut ainsi estimer le modèle de fuite d'information de différentes valeurs de clé à travers les traces. Ce type d'attaque par canaux auxiliaires permet également de contourner certaines des contre-mesures actuelles. L'émergence de l'intelligence artificielle et notamment de l'apprentissage profond a permis de développer de nouvelles attaques "profilées". Ces attaques sont considérées comme plus efficaces que les "attaques profilées" classiques. Ainsi suivant le contexte de l'attaque et la capacité de l'attaquant à acquérir des traces étiquetées ou non étiquetées, il peut monter différentes attaques par canaux auxiliaires: attaques non-profilées, attaques profilées classiques ou attaques profilées avec de l'intelligence artificielle.

Depuis l'émergence de l'intelligence artificielle, de nombreux chercheurs ont étudié les performances de différentes architectures de réseaux de neurones pour les attaques par canaux auxiliaires à travers divers contextes. Le contexte principal qui se retrouve dans la plupart des travaux est celui dans lequel un réseau de neurones a été entraîné sur une source d'information, cette source d'information est aussi celle utilisée pour la phase d'attaque du réseau. Ainsi la variance entre les deux phases est inexistante. Cependant dans un contexte réaliste, l'attaquant peut se retrouver avec plusieurs sources d'information qui contiennent chacune un certain niveau de bruit et ainsi un certain niveau de variance entre elles. Ce niveau de variance peut être dû à la

différence entre les systèmes embarqués, la différence entre la position de la sonde ou même une différence au niveau de l'implémentation logicielle. Un nombre restreint de travaux ont étudié ce contexte où la source d'information utilisée pour l'entrainement et la source d'information utilisée pour l'attaque sont différentes, et peuvent donc impacter l'efficacité du réseaux pendant l'attaque. Ainsi, nous allons voir dans cette thèse, comment un attaquant peut tirer profit de différentes sources d'information.

Objectifs

Dans cette thèse, nous allons étudier les techniques d'apprentissage profond dans le contexte où nous avons différentes sources d'information disponibles. Afin de montrer la pertinence de nos recherches, nous appliquons les approches proposées aux trois architectures de réseaux de neurones les plus en vue dans l'état de l'art actuel: ASCAD [54], Zaid [73], and NoConv [66]. Il n'existe pas de jeux de données publiques ayant la variabilité requise pour nos tests. Afin de mener à bien ces travaux et donc d'avoir accès à différentes sources d'information, nous avons donc décidé de générer nos propre jeux de données : "DATABASE_AVR" et "DATABASE_STM32". Le jeu de données "DATABASE_AVR" est composé de 780,000 traces d'ondes électromagnétiques provenant d'une plateforme STK500. Le jeu de données "DATABASE_STM32" est composé de 1,250,000 traces d'ondes électromagnétiques, ainsi que les mesures de consommation électrique correspondant, provenant d'une plateforme ChipWhisperer.

Le code source et les données utilisés sont disponibles à l'URL suivante: `https://github.com/GeneveyC/Machine-learning-with-SCA`.

Contributions

Dans un premier temps, nous avons cherché à savoir si la combinaison de plusieurs canaux d'observation provenant d'un même système mais de sondes différentes peut aider un attaquant à élaborer une attaque plus efficace qu'avec une source unique. Nous avons démontré que dans le cas où on ajoute une source d'information plus bruitée à une source unique, le réseau de neurones ne tirera aucun avantage de la nouvelle source d'information. Cependant, si l'on ajoute une source d'information moins bruité à une source unique, alors les performances du réseau de neurones seront améliorées.

Dans un deuxième temps, nous avons cherché à savoir si des sources d'information supplémentaires qui n'ont pas été mesurées à partir de la même plateforme/sonde/système peuvent aider un attaquant à monter une attaque plus puissante ou bien à accélérer le temps d'une évaluation

en facilitant la convergence d'un réseau. Nous avons utilisé la technique appelée apprentissage par transfert pour monter ce type d'attaque. Nous avons démontré que l'apprentissage par transfert permet en général de faciliter la convergence du réseau et, dans certains cas, d'améliorer son efficacité finale.

Enfin, nous avons cherché à savoir si deux sources d'information avec des efficacités différentes du point de vue de l'attaque peuvent être utilisées pour monter une attaque plus puissante. Dans ce contexte, nous avons étudié s'il était possible de traduire les traces d'un système qui est plus difficile à attaquer en d'autres traces d'un système qui est plus facile à attaquer. Nous avons démontré qu'il est possible de traduire des traces d'un système à un autre grâce à un réseau GAN (Generative Adversarial Network). Ainsi, nous avons traduit des traces d'ondes électromagnétiques en traces de consommation de courant et nous avons également traduit des traces de plusieurs systèmes de la famille STM32. Nous avons également démontré comment un réseau GAN peut être utilisé dans le cadre des attaques par canaux auxiliaires.

Contenu du manuscrit

Combining sources of side-channel information. Dans le chapitre 5, nous proposons d'explorer une approche avec plusieurs canaux auxiliaires, appelé multicanaux, grâce à l'apprentissage profond (DL). Nous étudions deux types de combinaisons multicanaux. Tout d'abord, nous étudions la combinaison d'émissions électromagnétiques provenant de différentes localisations et capturant des informations de fuite dépendantes des données sur le dispositif. Deuxièmement, nous étudions la combinaison des signaux de fuite classiques et d'une mesure de bruit ambiant. Nous décrivons également comment étendre une architecture de réseau de neurone convolutif (CNN) pour prendre en entrée plusieurs canaux. Le chapitre 5 est principalement basé sur les résultats publiés dans la conférence C&ESAR 2019 [19].

Train or Adapt a Deeply Learned Profile. Dans le chapitre 6, nous abordons le problème de la quantité limitée de données pour l'entraînement (nombre limité de clés de chiffrement connues, contraintes de temps lors de la phase d'acquisition). Nous étudions l'avantage d'utiliser des poids déjà initialisés, qui peuvent provenir d'un entraînement précédent du réseau sur des données acquises dans une configuration différente. Cette approche est connue sous le nom d'apprentissage par transfert (TL). L'idée sous-jacente est que les différentes attaques par canaux auxiliaires partagent des points communs dans le sens où une partie du réseau doit comprendre le lien entre les signaux électriques/électromagnétiques et la variable intermédiaire correspondante. Nous explorons l'impact de l'apprentissage par transfert sur différentes architectures de réseaux de neurones convolutif (CNN): ASCAD [54], Zaid [73], and NoConv [66]. Le chapitre 6 est principalement basé sur les résultats publiés dans la conférence LatinCrypt 2021 [20].

Trace-to-trace translation for SCA. Dans leur travail [67], Wu et Picek utilisent des auto-encodeurs comme prétraitement pour la réduction du bruit. L'idée principale est d'entraîner les auto-encodeurs en utilisant comme entrées des traces qui contienne un bruit et des traces moins bruyantes de sorte que l'auto-encodeur soit capable de supprimer une partie du bruit dans l'ensemble des données d'attaque. Dans le chaptire 7, nous proposons d'étendre cette idée d'utiliser les réseaux de neurones pour le prétraitement en utilisant le réseau GAN (Generative Adversarial Network) pour la traduction de trace à trace. Nous avons étudié différents types de traduction: traduction des traces d'ondes électromagnétiques vers des traces de consommation de courant entre différents dispositifs. De plus, nous étudions l'impact des hyperparamètres sur l'architecture du réseau GAN. Le chapitre 7 est principalement basé sur les résultats qui sont publiés à la conférence CARDIS 2021.

Conclusion

Ainsi à travers ces trois chapitres, nous démontrons que l'utilisation de différentes sources d'information peut être bénéfique dans certains contextes par rapport à l'utilisation d'une seule source d'information. Dans chaque chapitre nous avons présenté nos méthodes et démontré nos résultats avec différentes architectures de réseaux de neurones, qui ont été développées précisément pour les attaques par canaux auxiliaires par la communauté scientifique. Nous avons également étudié différents scénarios afin de montrer et comparer les différentes approches qu'un attaquant peut suivre quand il a à disposition une ou plusieures sources d'information. Ces travaux ont permis de montré de nouveaux vecteurs d'attaque possibles avec de l'intelligence artificielle pour les attaques par canaux auxiliaires (notamment concernant nos travaux sur la traduction de traces).

Table of Contents

List of acronyms

Advanced Encryption Standard (AES)
National Institute of Standards and Technology (NIST)
Data Encryption Standard (DES)
Side-Channel Attack (SCA)
ElectroMagnetic (emission) (EM)
Points of Interest (PoI)
Template Attack (TA)
Signal-to-Noise Ratio (SNR)
Simple Power Analysis (SPA)
Differential Power Analysis (DPA)
Correlation Power Analysis (CPA)
Artificial Intelligence (AI)
Random Forest (RF)
Support Vector Machine (SVM)
Artificial Neural Network (ANN)
Machine Learning (ML)
Deep Learning (DL)
Graphical Processing Unit (GPU)
Central Processing Unit (CPU)
Transfer Learning (TL)
Generative Adversarial Network (GAN)
Guessing Entropy (GE)
Convolutional Auto-Encoder (CAE)
Density-Based Spatial Clustering of Applications with Noise (DBSCAN)
Principal Component Analysis (PCA)
Linear Discriminant Analysis (LDA)
MultiLayers Perceptron (MLP)
Long Short-Term Memory (LSTM)
Deep Neural Network (DNN)
Convolutional Neural Network (CNN)
AutoEncoder (AE)
Stochastic Gradient Descent (SGD)
Root Mean Square Propagation (RMSprop)
Conditional Generative Adversarial Network (CGAN)
Speech Enhancement GAN (SEGAN)
Neural Network (NN)

List of Figures

List of Tables

Chapter 1

Context, Objectives and Contributions

1.1 Introduction to Cryptography

Since the first conflict in the world, armies have been using *Cryptology* during the battle for securing their communications or to intercept the information of the enemy. Cryptology is divided into two branches (as shown in Fig 1.1), *Cryptography* and *Cryptanalysis*. Cryptography is the art of creating secure communication between two entities, for example in the army when the Major state wants to send an order to a special division. Cryptanalysis is the art of breaking secure communication within the context of our previous example, it corresponds to a spy trying to intercept and read the order sent by the Major state. To securely transmit information and to be sure that the spy cannot read this information, the sender must use an encryption algorithm and a secret (a.k.a. the key). The key is used for encrypting the message. The message before being encrypted is called the *plaintext*, and the encrypted message is called the *ciphertext*.

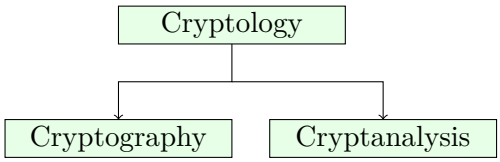


Figure 1.1 – The branches of cryptology

With the development of Computer Science, cryptographers designed new encryption techniques based on mathematical tools and more powerful calculations. These new encryption techniques, belonging to what is called Modern Cryptography, are present in our daily lives, for payments with our bank card, surfing on the internet, or sending a text message. The wide use of these algorithms makes their design increasingly sensitive. Indeed, poorly designed encryption schemes will have more and more impact as they are deployed. To provide a certain level of security and to prevent the effect of an attacker, four aspects have been defined in information security:

1. *confidentiality*: guarantees that information is only readable by an authorized person.
2. *data integrity*: ensures that any modification by an unauthorized person will be detected.
3. *authentication*: allows a party to check the identity claimed by another.
4. *non-repudiation*: prevents any party to deny actions or commitments.

Two branches of algorithms for cryptography may be distinguished: *symmetric-key cryptography* and *public-key cryptography*, also called *asymmetric cryptography*. Due to its advanced mathematical foundations, asymmetric cryptography has been developed later at the end of the 20 century while symmetric-key cryptography is far older.

The concept of symmetric-key cryptography is to use a shared key between the sender and the receiver, this key must be only known by these two parties. The sender and the receiver use the same key to encrypt and decrypt their messages. Thus an attacker who knows the key can decrypt the message of the sender and can spy on the conversation.

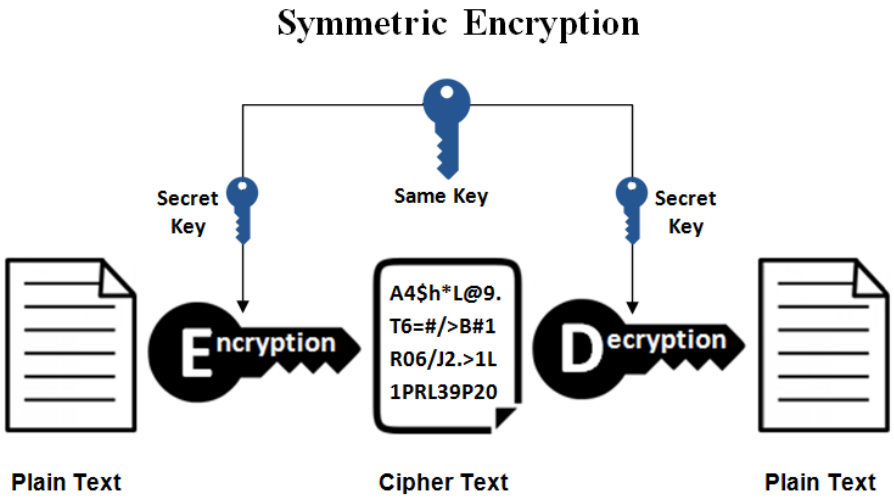


Figure 1.2 – Symmetric cryptography

The concept of asymmetric cryptography is to use a pair of keys, a public key (which may be known by everyone) and a private key (which must be known only by the receiver). The sender uses the public key of the receiver to encrypt the plaintext, and the receiver uses his private key to decrypt the ciphertext. Thus an attacker who knows the private key can decrypt all the messages received by this person.

In this thesis, we will only focus on symmetric encryption, and more precisely on the Advanced Encryption Standard (AES) algorithm.

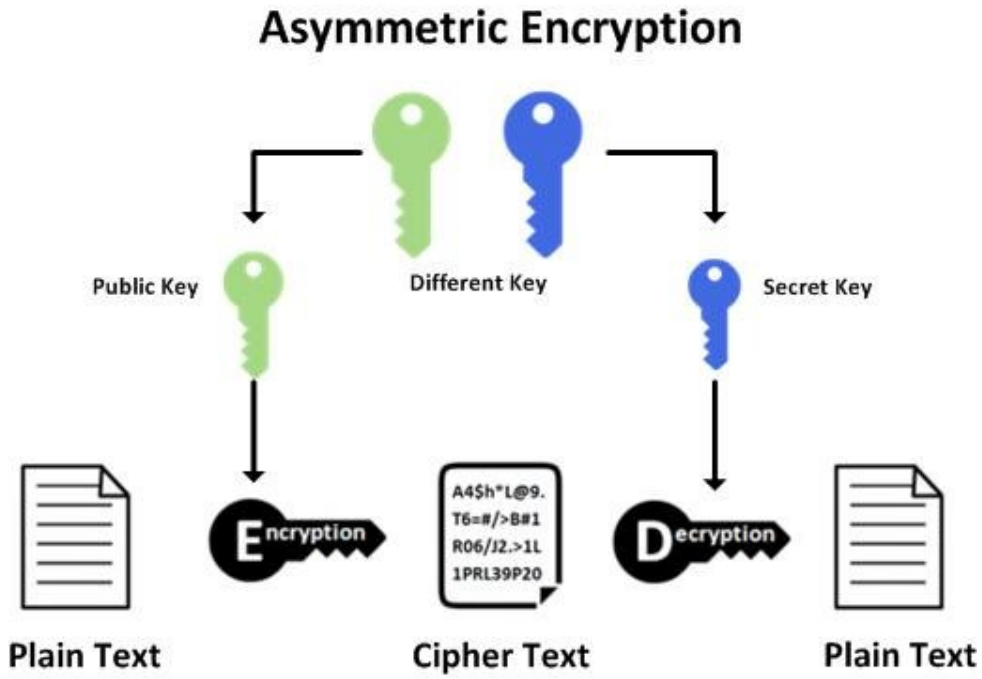


Figure 1.3 – Asymmetric cryptography

1.2 Advanced Encryption Standard

In 1998, Vincent Rijmen and Joan Daemen proposed a symmetric-key algorithm [16], originally named Rijndael for the AES competition organized by the National Institute of Standards and Technology (NIST). Their algorithm has been selected by the NIST and adopted by the U.S. government to replace the previously used Data Encryption Standard (DES). A new standard was needed because the predecessor DES used a too-small key and so became vulnerable to brute-force attacks. Three variants of AES exist, with different key lengths: 128, 192, and 256 bits.

As illustrated in Figure 1.4 illustrates all operations made by the AES algorithm. As with most symmetric schemes, AES have an iterative structure that is, a sequence of operations is repeated in what is referred to as a *round*. The number of rounds depends on the key length, the encryption process uses respectively 10, 12 and 14 rounds for the key of sizes 128,192 and 256 bits.

The AES algorithm is composed of the basic following operations:

— **Key Expansion**: Key expansion derives from the initial master key, a list of subkeys that will be used along the process (one subkey per round).

— **AddRoundKey**: A XOR operation is made between the bytes of the current state and the bytes of the subkey.

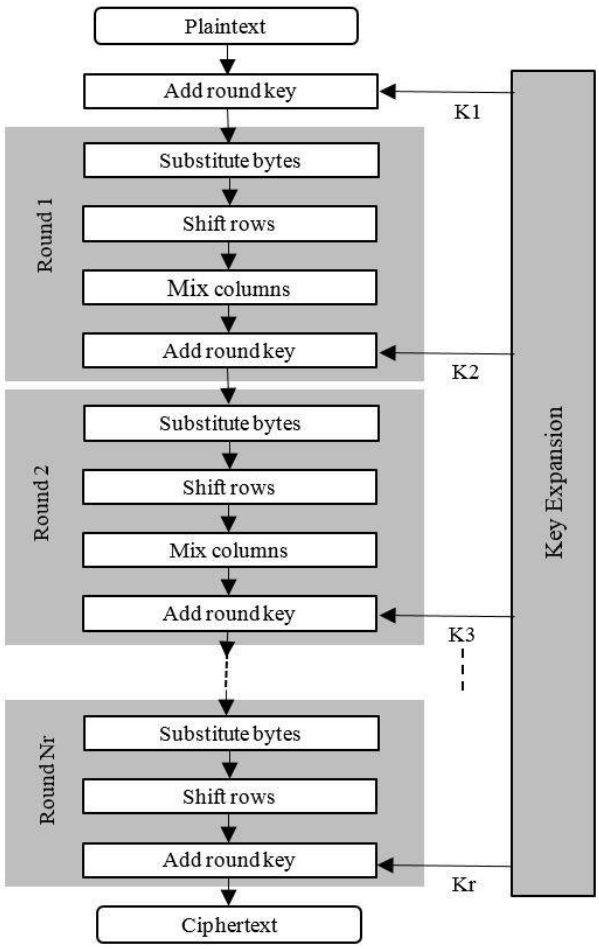


Figure 1.4 – Advanced Encryption Standard (AES)

— **SubBytes**: Each byte is substituted according to a predefined table (called *Sbox*).

— **ShiftRows**: A transposition step where the last three rows of the state are shifted cyclically a certain number of steps.

— **MixColummns**: Each column of the state is multiplied by a matrix with algebraic operations in a finite field of size 256.

1.3 Side-Channel Attack

Two criteria must be considered to have a safe cryptographic algorithm: the mathematical point of view and the implementation point of view. The first point of view, the *mathematical aspect*, refers to the situation where the attacker has only access to knowing a couple of plaintexts and ciphertexts and tries to recover the key. The second point of view, the *implementation aspect*, refers to the case where an attacker can take advantage of the particular implementation of the algorithm. An Side-Channel Attack (SCA) is one among all the implementation attacks. More precisely it belongs to the family of physical attacks since, in most situations, it requires having physical access to the targeted device to perform the attack.

The SCA consists in using some physical measurement (e.g. the power consumption) to discover some sensitive information used in an algorithm. Some physical attacks (e.g. fault injection), directly attack the cryptographic algorithms by injecting faults to bypass some operations or change the value of a state during the process. However, most physical attacks do not change the behaviour of the algorithm. They rely on the observation of the algorithm via the side channel.

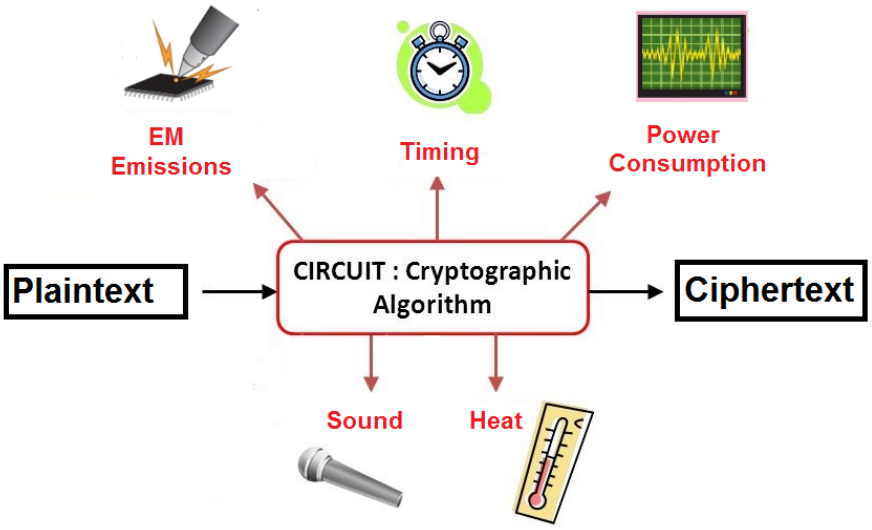


Figure 1.5 – The different source of side channel

As illustrated in Figure 1.5 illustrates the principle of SCA. A cryptographic algorithm is running on a smartcard or any other embedded device with physical access granted to the attacker. This cryptographic algorithm takes as input a plaintext and produces as output a ciphertext. During the execution of this algorithm, the device produces different side channels such as ElectroMagnetic (emission) (EM), timing, power consumption, sound, or heat. These

side channels are directly related to the internal state and processing of the device, so directly linked to the operations made by the cryptographic algorithm. An attacker may capture several traces from these different side channels. Then he can use statistical tools or artificial intelligence to discover sensitive information from these traces. In the case of a cryptographic algorithm, he can try to guess the key or some intermediate target value used by the algorithm during the encryption process. The attacker aims at finding the correct key or intermediate target value with as few traces as possible.

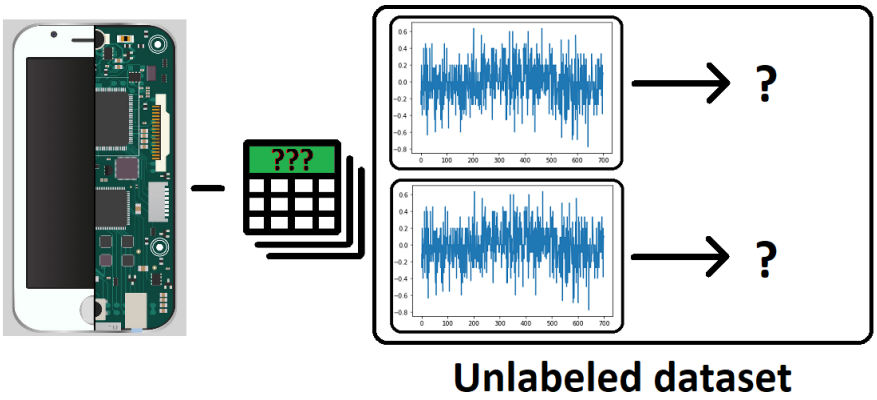


Figure 1.6 – Unlabelled dataset

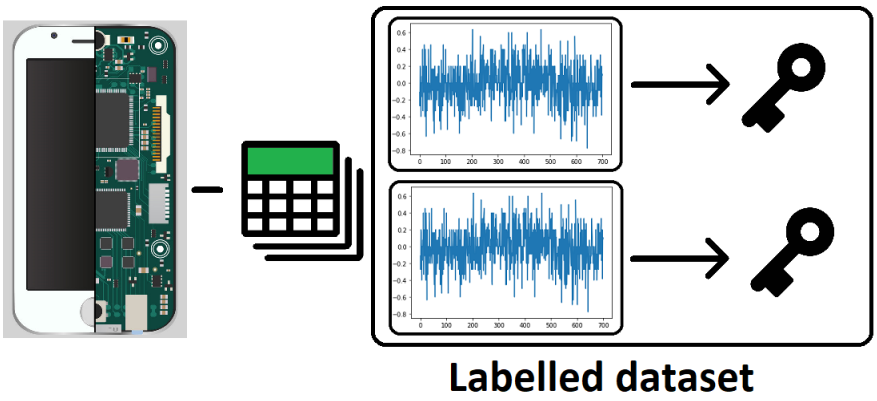


Figure 1.7 – Labelled dataset

Depending on the context and the capability of the attacker to acquire a certain amount of traces, he can mount different SCAs. First, the attacker always has an unlabelled dataset (as represented in Figure 1.6). An unlabelled dataset is a dataset composed of a set of traces but without the knowledge of the corresponding key (or intermediate target value). This dataset is

composed of the measurements obtained during the processing of an algorithm. These measurements are acquired without knowledge of the key (in general these measurements correspond to a fixed key that one seeks to find). Secondly, the attacker can have, in addition to the unlabelled dataset, a labelled dataset (as represented in Figure 1.7). A labelled dataset is a dataset composed of a set of traces with their corresponding key.

Sometimes the number of available observations for a given key is limited due to some counter-measures or use cases (e.g. decryption of some software in a secure boot procedure). Also, the interaction with some devices can be limited. In these cases, acquiring a certain amount of labelled data is a difficult task. The easiest way for an attacker is to have a clone of the target device. A clone means the same device with the same system and the same implementation. Thus, he will have full control of this device and will be able to capture as many traces as necessary, and he will know the corresponding key. The clone must be as close as possible to the device to be attacked because if the attacker succeeds to attack the clone, he will succeed to attack the targeted device. The issue of acquiring a certain amount of data, and a labelled dataset is not specific to the side-channel community. We found the same problem in the machine learning research area.

When an attacker has only access to an unlabelled dataset then he can perform a non-profiled attack. Then if he additionally have a labelled dataset then he can perform a profiled attack.

— *Non-profiled attack*: The attacker directly uses the unlabelled dataset and some mathematical tools (e.g. correlation metric) to find the secret key used by the algorithm.

— *Profiled attack*: The attacker first uses the labelled dataset, called profiling set, to estimate the leakage model. Secondly, he uses the target unlabelled dataset, called attack set, to recover the secret key used by the algorithm.

The first discovered side-channel attack is called Simple Power Analysis (SPA). SPA belongs to the non-profiled side-channel attacks. This attack consists in a simple analysis of one trace to find the correct key. Two other well-known non-profiled side-channel attacks are Differential Power Analysis (DPA) and Correlation Power Analysis (CPA). DPA consists in analyzing the difference between two sets of traces to guess the right key. For the CPA, the attacker creates a power consumption model during the analysis phase. He makes a prediction of power consumption using this model. Then, the correlation is calculated between the predicted power and the real (measured) power consumption to discriminate the right key.

Profiled side-channel attacks are more efficient than non-profiled side-channel attacks. The Profiled attack requires fewer traces since the knowledge of a better model allows for extracting more information from the traces. However, it requires a labelled dataset which is not required for a non-profiled attack.

The two main profiled side-channel attacks are template attacks and deep learning-based side-channel attacks. Template attacks consist in exploiting a Gaussian Modeling of the leakage using a Bayesian approach. The maximum likelihood principle enables them to reveal some part of the key for each set of traces.

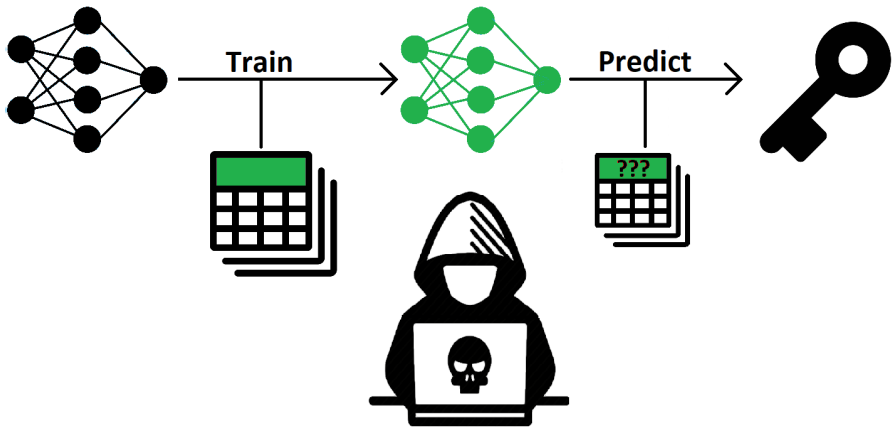


Figure 1.8 – Regular deep learning-based attacks

As illustrated in Figure 1.8, the regular deep-learning-based attacks. The principle of these attacks is quite similar to the one of template attacks. The only difference is the way the model is learned (using a neural network training instead of being computed based on the Gaussian assumption). The network was learned by using two datasets: a profiling set and an attack set. As mentioned previously, the profiling set is a labelled dataset with many traces obtained from different keys. Generally, the profiling set comes from a clone of the targeted device. The attack set is an unlabelled dataset with a certain number of traces and a fixed unknown key. It comes from the device to the target. An attacker uses the profiling set to train the network to classify each trace with its corresponding key value. Once the training is over, he uses the attack set to predict the key of the target device. The goal of the neural network is to guess the key (or intermediate target value) using as few traces as possible.

In most of the work considering side-channel attacks with deep learning techniques, the profiling set and the attack set come from similar measurement setups (using the same probe at a similar position on the same device). This guarantees a small variation between the profiling set and the attack set and thus ensures that the performance of the deep neural network will be the same on the profiling set and the attack set.

On the contrary, if no constraint is given on the training set, variations may arise from various sources:

— **The variation between probe type/position**: The attacker cannot use the same kind of probe or the same position to record the profiling set and the attack set.

— **The variation between side-channels**: The attacker cannot use the same side channel for the profiling set and the attack set. This is the case, for instance, when the attacker can have physical access to a clone device but not on the device to target. In this situation, he can acquire a profiling set with power consumption and an attack set with electromagnetic emission.

— **The variation between implementations**: The attacker cannot use the same implementation for on clone device compared to the one on the device to target.

— **The variation between devices**: The attacker cannot use a similar device as the the device to target.

Another hypothesis that is often made is that a sufficient amount of data is available to train the neural network.

A deep neural network is sensitive to these limitations. During the learning phase, the neural network will learn a mapping function from data that follows a certain distribution. However, if this distribution is different between the training phase and the attack phase (due to the variations mentioned above), the performance of the neural network may be degraded. Furthermore, the neural network may never converge toward an optimal solution if the amount of data is low. This drawback is not specific to side-channel attacks and can be found in other fields. In this thesis, we focus on the profiled attack with deep learning technique with the motivation of overcoming these limitations.

1.4 Contribution of the Thesis

The literature on side-channel attacks with deep learning techniques mostly considers situations where the variation between the profiling set and the attack set is small. This ensures that the performance of deep learning will not be altered due to the variation between these two sets. In some real-world scenarios, having a similar measurement setup for both profiling and attacking is difficult. In this thesis, we will show how to make a successful attack even with an attack set that contains a certain level of variation compared to the profiling set. As previously mentioned, variations may arise from various sources, the difference between probe types/positions, side channels, implementations, and devices. We use them with specific deep learning techniques in order to take profit from them.

Combining sources of side-channel information. In Chapter 5, we propose to explore a multi-channel approach thanks to Deep Learning (DL). We investigate two kinds of multi-

channel combinations. First, we investigate the combination of EM emissions from different locations capturing data-dependent leakage information on the device. Secondly, we investigate the combination of the classical leakage signals and a measure of mostly ambient noise. We also describe how to extend a Convolutional Neural Network (CNN) architecture (e.g. ASCAD, Zaid, and NoConv) to take as input multiple channels. Chapter 5 is mainly based on results published at the C&ESAR 2019 conference [19].

Train or Adapt a Deeply Learned Profile. In Chapter 6, we tackle the problem of the limited amount of data for training (limited number of known key encryption, timing constraint on the acquisition phase). We investigate the advantage of using already initialized weights. These weights may come from previous training of the network on some data acquired on a different setup. This approach is known as Transfer Learning (TL). The idea behind this is that different side-channel attacks share common points in the sense that part of the network has to understand the link between power/electromagnetic signals and the corresponding intermediate variable. We explore the impact of TL on different CNN architectures: ASCAD [54], Zaid [73], and NoConv [66]). Chapter 6 is mainly based on results published at the Latincrypt 2021 conference [20].

Trace-to-trace translation for SCA. In their work [67], Wu and Picek use auto-encoders as preprocessing for noise reduction. The main idea is to train auto-encoders using as inputs noisy traces and less noisy traces so that the auto-encoder is able to remove part of the noise in the attack dataset. In Chapter 7 we propose to extend this idea of using NN for pre-processing by using Generative Adversarial Network (GAN) for trace-to-trace translation. We investigate different kinds of translation: translation from EM to power traces and translation between different devices. Additionally, we study the impact of hyper-parameters on the GAN architecture. Chapter 7 is mainly based on results that have been published at the CARDIS 2021 conference.

Chapter 2

Machine Learning and Deep Learning

2.1 General Concept of Machine Learning

2.1.1 Introduction to Machine Learning

The term Artificial Intelligence (AI) is first introduced in the late 1950s and relates to the ability of a computer program to *think and learn*. Before this period, many mathematicians, and philosophers have been working on AI, among them Alan Turing and his famous Turing test [61]. Alan Turing is a brilliant young British mathematician who created a machine to break the Enigma encryption system. He is considered the father of AI. One important branch of AI that carries a lot of interest is Machine Learning (ML). ML is the study of different algorithms that automatically improve themselves as the experience progresses.

Before the machine learning algorithm can learn a task to reproduce it, we need to show some examples. These examples can be found in two datasets: **labelled dataset**, or **unlabelled dataset**. A labelled dataset is composed of raw data (e.g. pictures, audio files) and the corresponding labels, whereas an unlabelled dataset is only composed of raw data. Depending on the kind of datasets we have, we will use different kinds of algorithms.

Generally, the way datasets are used is very important because if they are not used properly, the algorithm can have a "rote" experience and not a "general" experience. When the algorithm has a "rote" experience, it is only able to reproduce the task on seen examples, and not able to reproduce the task on unseen examples. The algorithms which have a "general" experience can however reproduce the task on unseen examples.

To be sure that the acquired experience is general and not rote, we must divide the first dataset into two subsets and have a second dataset, thus we have a **training set** and a **validation set** to come from the first dataset and a **test set** from the second dataset. The algorithm accumulates experience with the **training set** during a phase called the **training step**, the algorithm used the **validation set** to be sure that the experience acquired is not a "rote" experience. After the **training step**, we do another check that the algorithm has learned the general and not "rote" experience with the **test set** during a phase called the **test step**. ML algorithms

using a labelled dataset are more powerful than the ones only using an unlabelled dataset.

Interest in ML has led to the development of different learning methods for different types of problems to be solved. These different learning methods also depend on the kind of dataset we can acquire. There are three main learning methods, called *machine learning paradigms*:

— unsupervised learning,

— supervised learning,

— reinforcement learning.

In addition to these three paradigms, two other learning methods exist but are not considered as main machine learning paradigms: *semi-supervised learning* and *transfer learning*. As illustraed in Figure 2.1 illustrates the hierarchy of these different methods (three paradigms and two additional learning methods). The green colour represents the method we use/study in this thesis.

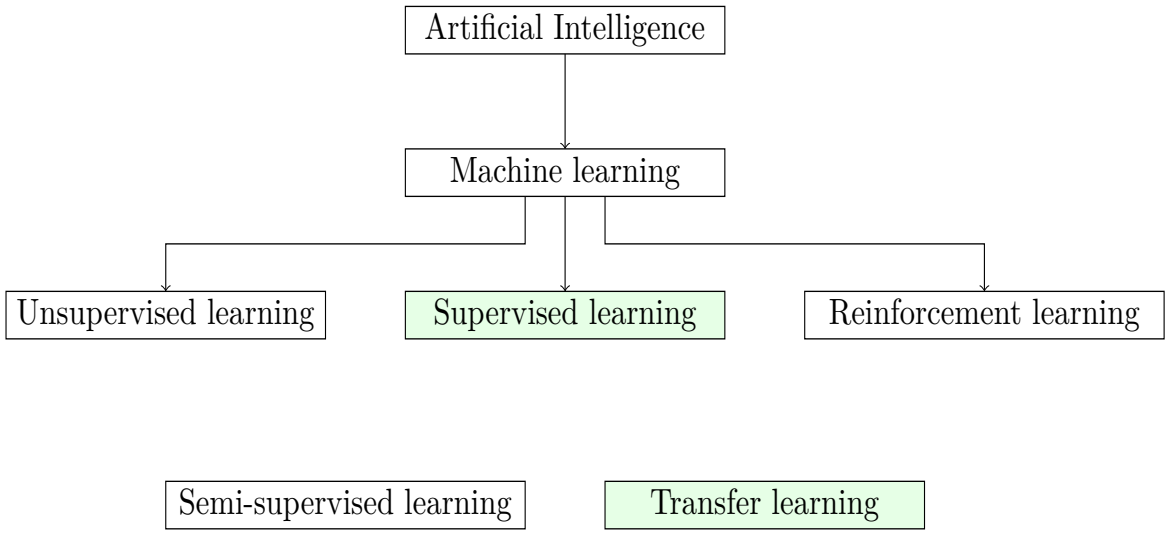


Figure 2.1 – The different methods of learning

Unsupervised learning: Unsupervised learning aims at learning patterns (or distributions) from an *unlabelled dataset* without any prior knowledge. Several unsupervised learning algorithms are used for clustering applications. Clustering aims to learn how to group data according to the difference between their distribution. Other unsupervised algorithms are used for anomaly detection or dimensionality reduction, e.g, Principal Component Analysis (PCA) and kernel-PCA.

Supervised learning: Supervised learning aims to learn from a *labelled dataset*. The labelled dataset means that we give the ML algorithms the desired data and solutions, called *labels*.

Two typical supervised algorithms are classification and regression algorithms. Classification algorithms are used to predict and classify discrete values such as Cat or Dog, Spam or No Spam, etc. Regression algorithms are used to predict continuous values such as the value of a company's stock, house prices, etc. Supervised learning is considered to be more powerful than unsupervised learning.

Reinforcement learning: Reinforcement learning aims to accumulate experience by performing actions in an environment to maximize a reward. In detail, an *agent* observes the environment, performs an action, and receives a *reward* in return. The reward in return can be positive (if the *agent* made the right decision) or negative (if the *agent* made the wrong decision). The *agent* must learn the best strategy, called *policy*, to get the best reward. Reinforcement learning is generally applied to various problems, such as robot control, telecommunications, or video games.

Semi-Supervised learning: Semi-supervised learning aims to combine *unsupervised learning* and *supervised learning* because it learns from a dataset that is partially labelled. Acquiring labelled data can be a difficult task because it is time-consuming, and sometimes we cannot assign a label to the data. However, we know that supervised learning leads to better performance than unsupervised learning. This is the reason why we want to do semi-supervised learning. In semi-supervised learning, we have two datasets, one labelled, and one unlabelled.

Transfer learning: Transfer learning aims to use the experience gained from solving one problem and applying it to another problem which has a similar context. Transfer learning is usually applied to computer vision, for example using the knowledge gained from recognizing cars to recognize trucks.

2.2 Supervised Learning with Neural Networks

2.2.1 Artificial Neural Network

In this section, we present only the classical supervised learning methods based on neural networks. Indeed, there exist other supervised learning methods which are not based on neural networks such as Random Forest (RF), and Support Vector Machine (SVM) but they are out of the scope of this document.

An artificial Neuron is a mathematical function inspired by our biological system. An Artificial Neural Network (ANN) is composed of one or multiple layers, and each layer is composed of one or more neurons. One of the first ANN architectures for supervised learning is the *Perceptron* which was invented in 1958 by Frank Rosenblatt. The Perceptron is a linear classifier

with one layer composed of one neuron. A Perceptron takes an input vector of size N, denoted by $x_1, \ldots, x_N \in X$, and returns a scalar value denoted by y. It is composed of a weight vector, denoted by $w_1, \ldots, w_N \in W$, and a bias θ. Each weight w_i represents the connection between the input value x_i and the only neuron.

$$y = g(\sigma) \tag{2.1}$$

$$\sigma = \theta + \sum_{i=1}^{N} w_i * x_i \tag{2.2}$$

$$\text{Heaviside}(\sigma) = g(\sigma) = \begin{cases} 1, & \text{if } \sigma \geq 0 \\ 0, & \text{if } \sigma < 0. \end{cases} \tag{2.3}$$

The Perceptron learns the mapping function from the input X to the target label $\hat{y}$. It computes a weighted sum between its input vector X and the weight vector W, then applies an *activation function*, denoted by g in the equation 2.2, to that sum and outputs a scalar value y. The most common activation function used in Perceptrons is the *Heaviside function* (see equation 2.3), but several other activation functions exist. We detail activation functions in Section 2.3.5.

$$w_i^+ = w_i + \eta(y - \hat{y}) * x_i. \tag{2.4}$$

The Perceptron updates its weights using a process named backpropagation. It consists in acquiring some experience to converge to an optimal solution (prediction close to the target label). Equation 2.4 describes the learning process and how the Perceptron updates its weights. We update each weight w_i by taking into account the error of the prediction made by the Perceptron. The error represents the difference between the prediction y_i and the target label $\hat{y_i}$. We use a *learning rate*, denoted by η, to take into account the degree of error. The machine learning process helps the reinforcement of the connection between the weight w_i and the input value x_i.

During the training of the Perceptron (or another ANN architecture), two other parameters must be taken into account: the number of *epochs* and the *batch size*. **Epoch** is a parameter that defines the number of iterations of the algorithm over the data. The **batch size** is a parameter that defines the maximum quantity of data the algorithm can see before updating its weights.

MultiLayers Perceptron (MLP): The universal approximation theorem [29] tells us that a single hidden layer forward propagation network containing a finite number of neurons can approximate any continuous function. Thus, the Perceptron can generalize any linear problem, but not non-linear problems. To solve the non-linear problems, researchers created another ANN architecture called MLP. The MLP can be viewed as an extension of Perceptron because instead of being composed of one layer like the Perceptron, it is composed of several layers. In the MLP architecture, we can distinguish three main layers: the *input layer*, the *hidden layer*, and the *output layer*. Each layer can be composed of one or multiple neurons. The **input layer** brings the initial data to the neural network. **Hidden layers** are the intermediate layers between the input layer and the output layer and is where all the computation takes place. The **output layer** produces the prediction for the given input. When an ANN contains a deep stack of hidden layers, it is called a *Deep Neural Network (DNN)*.

$$y_j^\ell = g(\sigma_j^\ell) \tag{2.5}$$

$$\sigma_j^0 = \theta_j^0 + \sum_{i=1}^{N} w_{i,j}^0 \cdot x_i \tag{2.6}$$

$$\sigma_j^\ell = \theta_j^\ell + \sum_{i=1}^{N_{\ell-1}} w_{i,j}^\ell \cdot y_i^{\ell-1} \tag{2.7}$$

The MLP learns the mapping function between input X and target label $\hat{y}$ like the Perceptron. However, with multiple layers, each neuron does not perform the same calculation as the single neuron in the Perceptron.

The input layer in the MLP architecture takes an input vector of size N, denoted by $x_1, \ldots, x_N \in X$, and returns a list of values (each neuron j returns a scalar value y_j). Equation 2.6 described the computation made by each neuron, the weight $w_{i,j}^0$ represents the connection between the input value x_i and the neuron j at the first layer. The activation function is denoted by g, and the bias to apply to neuron j is denoted by θ_j^0.
The hidden layer in the MLP architecture takes all outputs of the neurons of the previous layer of size $N_{\ell-1}$, denoted by $y_1^{\ell-1}, \ldots, y_N^{\ell-1}$, and returns a new list of value (each neuron j return a scalar value y_j^ℓ, the computation described in equation 2.7). In equation 2.7, the weight $w_{i,j}^\ell$ represents the connection between the output of neuron i at the previous layer $\ell - 1$ and the neuron j at the layer ℓ, g represents the activation function, and θ_j^ℓ represents the bias to apply to neuron j at layer ℓ. The hidden layer and the output layer are also called **dense layers**.

Convolutional Neural Network: Another ANN architecture arose from the scientific community to perform computer vision tasks namely Convolutional Neural Network (CNN). The CNN can be viewed as an improvement of MLP because it automatically extracts some features from the input X before learning the mapping function from the input X to the target label $\hat{y}$. The feature extraction is performed by two new types of layers: the *convolutional layers* and the *pooling layers*. Learning the mapping function between these features and target label $\hat{y}$ is always performed by one or more **dense layers**.

The **convolution layer** allows extracting features from the input X by applying one or more kernels to it. In the convolutional layer, we still have neurons, but we talk about *kernels* than neurons.

Each kernel in the convolutional layer still calculates a dot product of its weight W with the input X, followed by an activation function g, but their connectivity is limited to a local level. The other difference between the convolutional layer and the dense layer is that the convolutional layer uses a *shared weight matrix*. By sharing the weights, the network is able to detect a feature at different parts of the image. This means that all the neurons in the layer detect exactly the same feature, just at different locations in the input data. As illustrated in Figure 2.2 shows an example of how a convolutional layer applies the kernel of size 5x5x3 on an input of size 32x32x3.

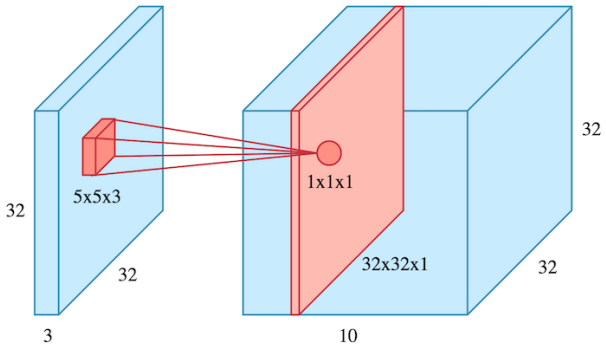


Figure 2.2 – Convolution kernel

The **pooling layer** permits the reduction of the size of the data by summarizing a window into a single value based on the maximum value (*Max pooling layer*) or based on the average value, (*Average pooling layer*) after applying the convolutional layer. These two methods of pooling are helpful to extract features from data.

2.2.2 Learning Process

To converge toward an optimal solution at each epoch, the MLP and CNN use a learning process too. The learning process of the MLP and CNN is called the *backpropagation algorithm*. The **backpropagation algorithm** uses a loss function, it is a function that computes the error committed by the network, for the classification task we use a binary cross-entropy function, and for the regression task, we mainly use mean squared error. Thus, the **backpropagation algorithm** consists in calculating the gradient of the loss function with respect to the model parameters. In detail, it can be determined how each connection weight and each bias term should be adjusted to reduce the loss error. The loss function must be derivable and its gradient should be efficiently computed. Once these gradients are calculated, we apply the *Gradient Descent* algorithm or another *Stochastic optimization* algorithm, and we repeat this process until the network converges toward a solution.

Stochastic optimization refers to the use of randomness in optimization algorithms. The main challenge for the optimization algorithms is to find a global minimum, avoiding the pitfalls of local minimums. Stochastic optimization algorithms reduce the probability of finding a local minimum and getting stuck there, thus increasing the probability of finding the global minimum. The space search to find the global minimum depends on the number of parameters used by the network. Thus, it will take more time and more data to find the global minimum for a CNN than for an MLP.

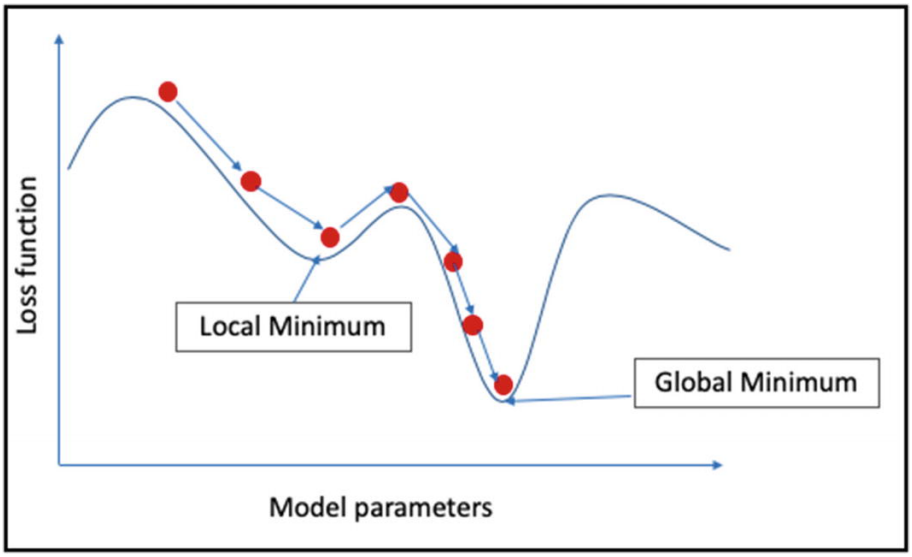


Figure 2.3 – Illustration of local minimum and global minimum

As illustrated in Figure 2.3 we shows the difference between the local minimum and the

global minimum. The local minima are good solutions that have interesting properties to solve our problem but do not correspond to the best optimal solution. The global minimum is the optimal solution to solve our problem. The global minimum is defined over all data coming from the training dataset. Then we assume that with enough data, the global minimum from the training dataset corresponds to the real global minimum.

There is no difficulty for convex functions to find a global minimum because, for convex functions, a local minimum is a global minimum, thus stochastic optimization algorithms trivially go toward the global minimum. On the other hand, non-convex functions have the problem of finding the global minimum and not getting stuck in the local minimum. There are many Stochastic optimization algorithms, also called **optimizers**, such as Stochastic Gradient Descent (SGD), Root Mean Square Propagation (RMSprop), or Adam that can be used to circumvent this local minimum issue.

2.2.3 Training and Validation in Supervised Learning

Data is required to train, validate and test a machine-learning algorithm. In the case of a supervised learning algorithm, the data is divided into three parts: a *training set*, a *validation set*, and a *test set*. We usually take 80% of the dataset for the training and validation sets, and the remaining 20% for the test set, as shown in Figure 2.4.

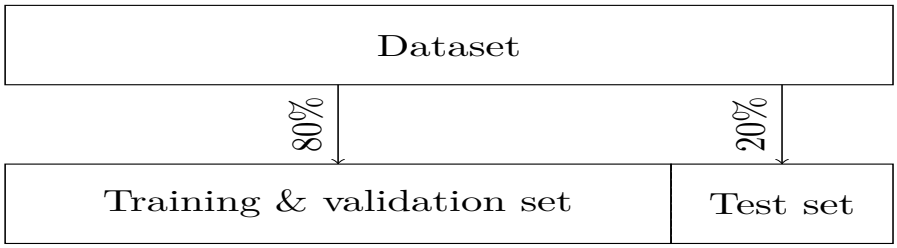


Figure 2.4 – Ratio of training & validation set

Training phase: During this step, the supervised learning algorithms attempt to estimate the mapping function based on the *training dataset*. The training dataset is composed of data, denoted by $X_{training}$, and the corresponding labels, denoted by $Y_{training}$. We provide several couples of features and labels to the algorithm to estimate the mapping function. The backpropagation is applied once a certain amount of pairs (namely the batch size) have been seen by the algorithm.

Validation phase: After each epoch, we estimate the committed error by the supervised learning algorithms on the *validation dataset*. We perform a validation phase to ensure that the network error continues to decrease and that it is not overfitting. If the loss is still decreasing it

confirms that the network is still generalizing. During the validation phase, only the validation dataset is used to estimate the error of the network. Notice that the weights of the network are not updated at all during validation. The network parameters that correspond to the lowest error on the validation dataset correspond to the parameters which are the closest to the global minimum.

Test phase: During this step, we apply the mapping function with the *test dataset*. If the network found the global minimum during the training phase, it must predict a value, denoted by $y_{predict}$ close or equal to the target value, denoted by $\hat{y}$. An important aspect is that the test dataset should not be used during the training step because it introduces a bias in the training of the neural network.

Cross-validation process: This process is useful to be sure that the supervised learning algorithms perform well on different test datasets. Cross-validation consists in performing many pieces of training on different datasets. These datasets are generated from the same data but by partitioning in different ways. For each training, we have a different training dataset, validation dataset, and test dataset. This process allows for validating that the obtained network has good performances on several test datasets. The problem is if the network performs well on one test dataset but not at all, it can be estimated that the network is overfitting.

2.2.4 Main Challenge in Supervised Learning

Several problems can occur in the learning of a neural network. These problems are common to machine learning and can occur in different domains. In this section, we will detail each of these problems.

The **vanishing gradient problem**: During the learning process, the gradient of the loss function can get smaller and smaller up to the point where it finally vanishes. A gradient close to zero means that the weights of the neural network will not be updated, so the neural network will never converge toward a good solution. This problem can be either due to the weight initialization or the activation function used in the neural network. For example, the sigmoid function is bounded into 0 and 1, and its derivative goes from 0 to 0.25. During the learning process, the gradient of the sigmoid at very high or low values is almost 0, which means no weights of the neural network will be updated.

The **exploding gradient problem**: The exploding gradient problem is the inverse problem of the vanishing gradient problem. During the learning process, the gradient of the loss function is getting higher and higher. A high gradient means that the weight of the neural network will be updated with an extreme value, so the neural network will diverge too much and become unstable.

The **insufficient quantity of training data problem**: DNN are particularly affected by this problem because the parameter space (on which the global minimum is searched for) is wider for a DNN than for an MLP. To have a good estimation of the mapping function and to generalize well on test data, a DNN requires a certain amount of data. This amount of data depends on the kind of problem to solve and the complexity of the neural network. A deeper neural network such as a CNN requires more training data than an MLP .

The **non-representative data problem** means that the model performs well on the training dataset, but does not on the test dataset. This problem can occur when the distribution of the data into the training and test dataset are not the same. It is important to have similar distributions of the data in both datasets because when the network has been trained, it expects to see the same distribution. A typical example is Apple's facial unlocking application. The model was only trained to recognise Western faces and when asked to recognise Asian faces, the model could not distinguish them.

The **poor-quality data problem**: The network may have difficulties performing well on training and test datasets if one of these two datasets contains errors, or noise (e.g. due to poor-quality measurements). Most of the time, scientists spend time cleaning their training and test data to obtain better performances for their network. A typical example of poor-quality data is an audio dataset containing a high level of noise.

The **irrelevant features problem**: To obtain good performances of the neural network, we must feed it with relevant features. For Perceptron and MLP, the relevant features can be obtained with the *feature engineering process*. The feature engineering process includes a human expert who determines and selects the best features to give to the neural network. For CNN the features are directly selected by the network as part of its training process.

The **hyperparameter tuning problem**: The hyperparameters of the network such as activation functions, the number of nodes, and optimizers can affect the guessing performances and the training time. To be sure to find the global minimum, we must define the correct hyperparameters for the network, this step is called hyperparameter tuning. The hyperparameter tuning aims to test a lot of hyperparameters (e.g. different activation functions with different optimizers) to reduce the error committed by the network with the validation dataset. Due to the large range of hyperparameter values, this step is time-consuming. The cross-validation step is useful for the hyperparameter tuning problem because it allows us to check that the chosen hyperparameters are optimal on different datasets.

The **overfitting problem** means that the model performs well on the training dataset but cannot generalize well on the test data, we said that the model acquires a "rote" experience

during the learning part. A DNN which approximates a function must detect some patterns in the training data and use it on the test dataset. However, if the training data is too small or the detected patterns are noisy or biased, the DNN cannot generalize well on the test data, which leads to the overfitting problem.

The **underfitting** problem is the inverse of the **overfitting** problem. The **underfitting** problem means that the model cannot approximate a function on the training data and the test data. A model cannot approximate the function if it is too simple or not complex enough. The first idea to solve the underfitting problem is to select a deeper neural network with more parameters/weights to update. The second idea to solve this problem is to give better features to the neural network (if our neural network does not automatically extract the features).

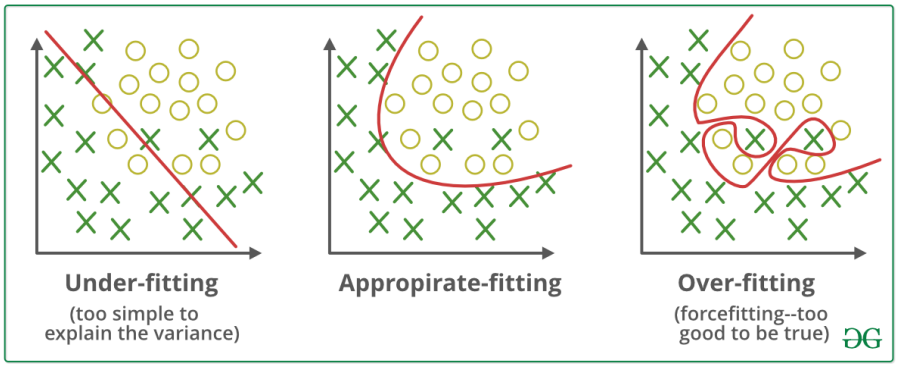


Figure 2.5 – Overfitting vs underfitting

2.3 Advanced Methods of Supervised Learning

In this section, we describe several techniques that have been developed by the machine learning community to solve the problem of insufficient data. This issue is described in Section 2.2.4. In this thesis, we use some of these techniques and apply them to the side-channel context.

2.3.1 Generative Adversarial Network

Generative Adversarial Network (GAN) is a special kind of ANN introduced by Ian Goodfellow [24]. GAN is mainly used to solve **the problem of insufficient data** by learning how to generate synthetic additional data presented in Section 2.2.4 and to make the neural network more robust. A GAN is composed of two neural networks, called **discriminator** and **generator**. These two networks have specific roles and confront each other to improve themselves. The

confrontation principle is based on the same principle as the min-max algorithms, each network tries to maximize its score and beats its opponent.

The *generator* aims to generate new samples that are close to the distribution of the target data. More precisely, the generator attempts to make a more realistic distribution to deceive the discriminator. The generator outputs synthetic samples that are drawn from a distribution that should be as close to the actual one as possible. During the training of the generator, the synthetic distribution will get closer and closer to the target distribution. The generator tries to minimize the min-max loss function, described in equation 2.8.

The *discriminator* aims to distinguish if the distribution comes from a real dataset or if it was made by the generator. In detail, the discriminator attempts to identify the synthetic distribution made by the generator. The discriminator takes as input one distribution and outputs the probability that this distribution is real or fake. During the training, the discriminator will improve and will be able to distinguish the two distributions. The discriminator tries to maximize the min-max loss function, described in equation 2.8.

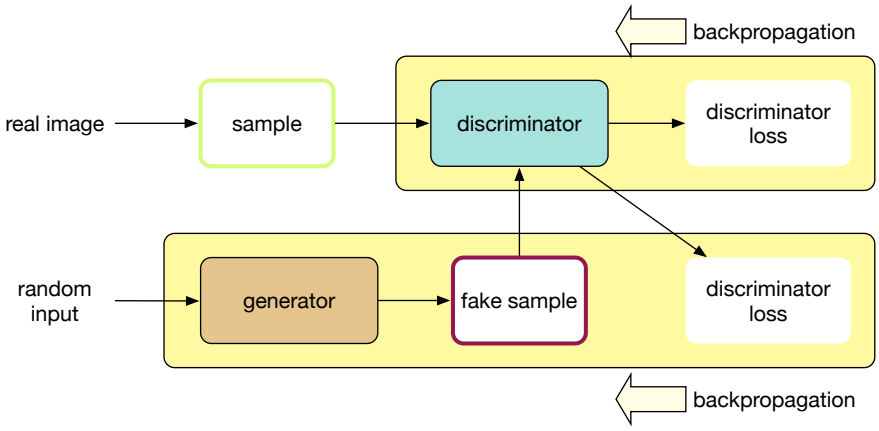


Figure 2.6 – General overview of GAN

As illustrated in Figure 2.6 we shows the general process of GAN with the discriminator and generator part.

$$L_{GAN} = E_x \left[\log(D(x))\right] + E_z \left[\log(1 - D(G(z)))\right]. \tag{2.8}$$

In equation 2.8:

— $D(x)$ is the discriminator estimate of the probability that real data instance x is real

— $G(z)$ is the generator output for a given noise z.

— $D(G(z))$ is the discriminator estimate of the probability that fake instance is real.

— E_x, E_z are respectively the expectation over real instances and over random inputs to the generator.

There exist a lot of variants based on the same principle as the classical GAN. A known variant is Conditional Generative Adversarial Network (CGAN) which was introduced by Mirza et al [49]. CGAN uses additional information to produce synthetic distribution. The generator of the CGAN takes as input a random distribution and the conditional class to generate the fake distribution. The CGAN is used for data augmentation to make samples for a specific class. The min-max loss function was adjusted for CGAN architecture because it takes into account the conditional class in addition, denoted by y in equation 2.9.

$$L_{cGAN} = E_x[log(D(x))] + E_{y,z}[log(1 - D(G(y, z)))]. \tag{2.9}$$

In the community of computer vision, the state-of-the-art concerning CGAN are Pix2Pix [31] and CycleGAN [74]. The architecture of these two CGAN are different but both are used for image-to-image translation. The image-to-image translation consists in transforming an image into another, e.g. transforming a drawing into a real image, or transforming a horse into a zebra. Another difference is the constraint on the input data, Pix2Pix network is trained on paired datasets whereas CycleGAN uses unpaired datasets. We define both notions in the next paragraphs.

Paired Datasets. Two datasets are paired when the following one-to-one relationship exists between values in the two datasets.

1. Each dataset has the same number of entries.
2. Each entry in one dataset is related to one, and only one, entry in the other dataset.

Unpaired Datasets. Two datasets are unpaired when their entries do not satisfy the paired datasets conditions.

2.3.2 Siamese Network

The Siamese neural network is used to mitigate the **problem of insufficient data**, presented in section 2.2.4. The Siamese network is used to solve **the problem of insufficient data** by learning the similarity between two data. The idea of the Siamese network is to have two or more identical networks, however each network takes independent and different inputs. The Siamese network is particularly useful in the multi-classification task where we have a large number of

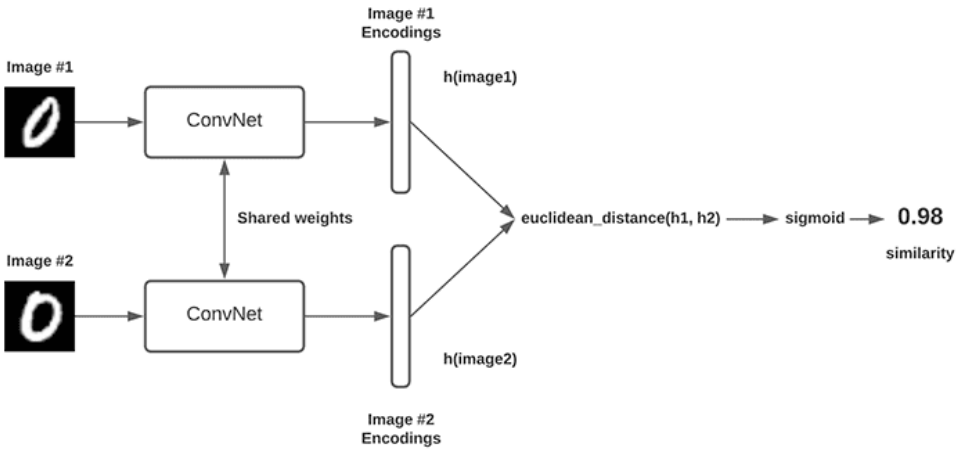


Figure 2.7 – General overview of Siamese network

classes with small samples per data. In such case, where we do have not enough data to train a DNN, so we can train a Siamese network to determine if two data belong to the same class.

The term *identical* networks means that all networks have the same architecture with the same parameters and the same weights. To update the weight of all networks during the learning process, the backpropagation is mirrored across both networks. The advantage of the siamese neural network compared to a classical network (MLP, CNN) is that it can obtain good performance with fewer data, however, it needs more training time than a classical network.

The siamese neural network is illustrates in Figure 2.7. We provide a couple of differents inputs to these networks and each network computes the features of one input. Then, the similarity of features is computed using the Euclidean distance and a logistic function. For two different inputs within the same class, the target output is 1 otherwise the target output is 0. The siamese network is trained to minimize the distance between samples of the same class and increase the inter-class distance. The loss function used for the siamese network is a similarity function.

2.3.3 Transfer Learning

transfer learning is used to mitigate the **problem of insufficient data**, presented in Section 2.2.4. The idea of transfer learning is the following. Instead of using a random weight initialization, the network is initialized using weights from another network trained on a specific task. We can use the weights of a neural network after the first training, and then fine-tune them during a second training. There are different methods for applying transfer learning. Figures 2.8 and 2.9 illustrate two kinds of transfer learning. Layers are initialized with weights from previous learning and, in green, are the layers whose weights will be updated during the new training. In

Figure 2.8, we update all the weights of a pre-trained network while in Figure 2.9, we update only the weights of the fully connected layer. In this later example, we keep the same feature extraction, done by the convolutional layers during the first training, for the second training.

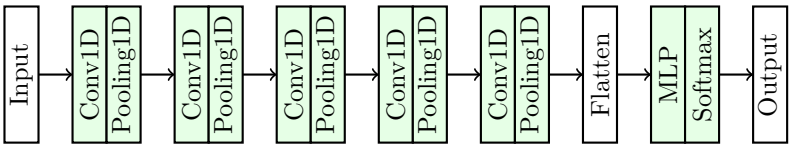


Figure 2.8 – Transfer learning by updating all weights in network

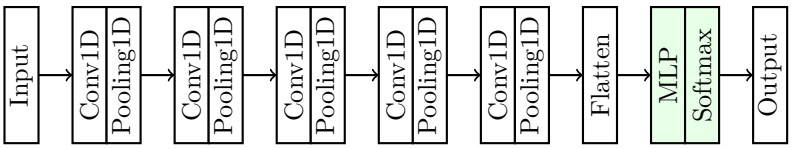


Figure 2.9 – Transfer learning by updating only the weights of fully-connected layer

2.3.4 Additional Layers

The **Batch normalization layer** contributes to solving the **vanishing gradient issue**, presented in Section 2.2.4. Moreover the batch normalization layer allows a more stable and faster convergence of a neural network. The batch normalization layer normalizes the output of a layer by applying a transformation that maintains the mean equal to 0 and the output deviation close to 1. As its name suggests, Batch Normalization is applied to all the data from a batch, that is both the mean and the variance are computed on the full batch.

The **Dropout layer** permits to prevent of the **overfitting issue**, presented in Section 2.2.4. The dropout layer randomly disables a certain amount of units at each new batch so that the weights of these neurons are not adjusted during the learning phase. The proportion of discarded units is known as the *dropout rate*.

2.3.5 Additional Activation Function

The ReLU Activation Function. Equation 2.10 describes the ReLU activation function. The ReLU activation function is not perfect because, during the training, some neurons of the network can *die*. We say that a neuron dies when it only outputs 0 and its weights cannot be updated because the gradients fail to flow during the learning process. This problem is known as the *dying ReLUs* [43].

$$ReLU(z) = \max(0, z) \tag{2.10}$$

The Leaky ReLU Activation Function. Equation 2.11 describes the Leaky ReLU activation function. The Leaky ReLU activation function is a variant of the ReLU activation function that helps to solve the *dying ReLUs* problem. The parameter λ is the slope of the function when $z < 0$. This parameter ensures that the neuron which has leaky ReLU as an activation function never dies.

$$LeakyReLU(z) = \max(\alpha z, z) \tag{2.11}$$

The SeLU Activation Function. Equation 2.12 describes the SeLU (Scaled Exponential Linear Units [35]) activation function. The SeLU activation function is also a variant of the ReLU activation function that also helps to solve the *dying ReLUs* problem because the SELUs activation cannot die.

$$SeLU(z) = \lambda \cdot \begin{cases} \alpha(e^z - 1) & \text{for } z < 0 \\ z & \text{for } z \geq 0 \end{cases} \tag{2.12}$$

Chapter 3

Side-channels attacks

In everyday life, cryptography is necessary to secure our exchanged messages and protect our private life. Even if we develop more secure encryption algorithms, attackers will develop new effective attacks. Cryptography is in perpetual evolution. On the one hand, defenders propose new countermeasures to limit and avoid attacks. On the other hand, attackers develop new attacks to defeat these countermeasures. As presented in Chapter 1, there are two main families of algorithms in cryptography, namely symmetric-key cryptography and asymmetric cryptography. An attacker can have access to an embedded system and in this case can carry out what we call physical attacks. Some physical attacks use physical measurements (e.g. electromagnetic emission) to discover some sensitive information used in an algorithm during these operations. Other physical attacks directly attack the cryptographic algorithm by injecting faults to bypass some operations of the cryptographic algorithms.

3.1 General Concept of Physical Attack

3.1.1 Introduction to Physical Attack

Physical attacks regroup different kinds of attacks. We can classify them into two groups with two main axes into each group. *Invasive attacks/non-invasive attacks*, and *active attacks/passive attacks*.

— **Invasive attacks**: The attacker gets direct access to the components inside the device to target (e.g. the attacker can directly connect a probe on the data bus to see the transferred data).

— **Non-invasive attacks**: The attacker exploits externally available information (typically eavesdropping electromagnetic emission).

— **Active attacks**: The attacker tries to alter the targeted device by introducing some error and affecting their operations. These attacks involve modifying the current state of the device to bypass some operations and obtain some privileges.

— **Passive attacks**: The attacker tries to learn some information about the device to target by recording physical measurements but does not affect it. Many physical quantities can be

used for this kind of attack, such as power consumption, electromagnetic emission, time, sound or heat.

3.1.2 Overview of Side-Channel Attack

In the thesis, we will mainly focus our research on non-invasive passive attacks. The non-invasive passive attacks are a family of physical attacks that regroups different kinds of attacks that exploit different side-channel to learn some information about the target device. The most used side channels are electromagnetic emission, power consumption, timing, sound, and photon emission.

These side channels lead to various attacks that are depicted in Figure 3.1:

— **Cache attacks**: The attacker monitors the cache accesses made by a victim in a shared physical system (a virtualized environment or a cloud service) to learn information about the victim (e.g. spy on a victim).

— **Timing attacks**: The attacker measures the execution time of various operations to learn some information about the victim (e.g. discover the length of the password of the victim).

— **Electromagnetic/Power attacks**: The attacker measures the electromagnetic emission/power consumption of the hardware during some operation to learn some information about the device (e.g. discover the key used by an encryption process on a device).

— **Acoustic attacks**: The attacker exploits the sound emitted by some device used by the victim to learn some information about him (e.g. discover the password of the victim with the sound emitted by his/her keyboard).

— **Heat attacks**: The attacker exploits the temperature variations of some devices to learn some information about the victim (e.g. discover the password of the victim after he/she typed it on his smartphone).

— **Photon attacks**: The attacker exploits the photon emission from the hardware to discover some information about the victim (e.g. obtain information on the victim's activity).

3.1.3 Attack Context

In this thesis, we will mainly focus our research on power and electromagnetic attacks. Depending on the context and the capability of the attacker to acquire a certain amount of traces, he can mount different Side-Channel Attacks (SCAs). First, we always consider that the attacker have an unlabelled dataset. An attacker with an unlabelled dataset can mount a non-profiled attack. Secondly, the attacker can have in addition to the unlabelled dataset, a labelled dataset. An attacker with a labelled dataset can mount profiled attacks. Profiled attacks are more

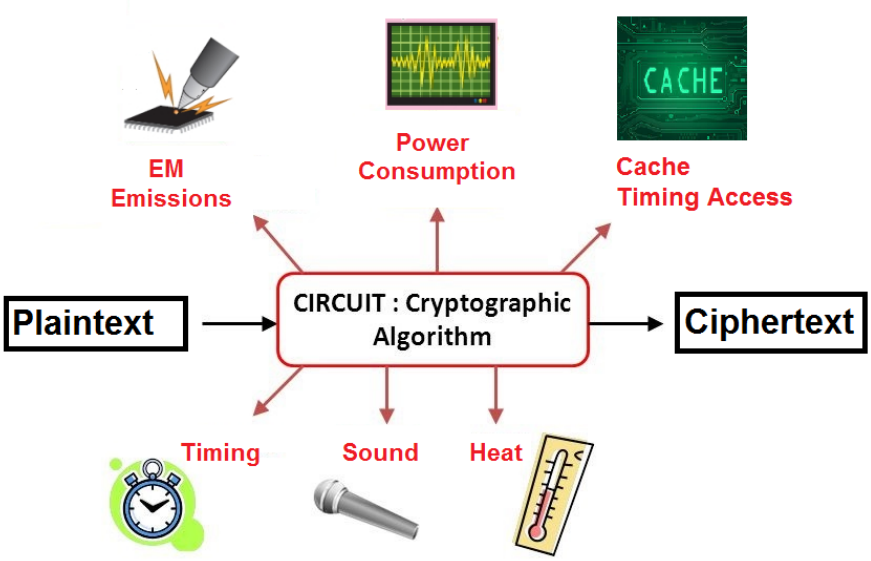


Figure 3.1 – The most use source of side channel attack

powerful because they require fewer traces than non-profiled attacks. However, profiled attacks required a labelled dataset, and in some situations having a labelled dataset is not possible.

Non-profiled attacks: In the context where an attacker have no access to a clone of the device, nor he can train by running the target device with some known keys, he only have an unlabeled dataset. This is the situation where one have to use non-profiled attacks. The attacker extracts electromagnetic or power traces from the target device, and directly analyzes these traces with a statistical tools to discover some secret used during the encryption process. The analysis made by the attacker requires to have some knowledge about the encryption process running on the target device.

Profiled attacks: This attack appears when the attacker have access to a device similar to the targeted one (also called a *clone device*), thus he can have a labelled dataset. The attacker have the opportunity to train a model on a similar device (with access to one or more know keys). Thus the attacker tries to model the leakage on a similar device, then he uses this model to discover the sensitive value used by the target device during the encryption process.

In this thesis, we explore deep learning techniques (presented in Section 3.6) with electromagnetic and power traces. While deep learning training usually uses a labelled dataset from a clone device (and thus is considered a profiled technique), some of our contributions are using deep learning using labelled datasets from another experimental setup (different source, probe, device ...). In these later cases, the use of deep learning may be considered non-profiled.

3.2 Non-profiled Side-Channel Attacks

In non-profiled side-channel attacks, an attacker called A have only access to the device to target denoted by D_{target}. We assume that an encryption process denoted by E runs on D_{target}. The encryption process E takes a couple composed of a plaintext p and a key k as input and returns a ciphertext c. The attacker knows the plaintext p used by the encryption process but he does not know the key k. The encryption process outputs the ciphertext $c = E(p, k)$.

The attacker extracts a list of N_a measurements called *attack set*, denoted by $x_1,...,x_{N_a} \in X_{attack}$. The *attack set* is always an unlabelled dataset. All traces in X_{attack} are extracted from the device D_{target}. Once the attacker have collected enough data, he can use *classical attacks* such as Differential Power Analysis or Correlation Power Analysis to reveal the potential value of the secret key k.

Differential Power Analysis (DPA) is introduced by Kocher *et al.* in [38]. DPA is a statistical method for analyzing sets of measurements to identify data-dependent similarity. This method involves partitioning a set of traces into two subsets based on a key candidate and a generic leakage model. Then, the difference between the averages of these subsets is computed. If the partitioning is sound (i.e. the key candidate is correct and the used generic model makes sense), then the average will approach a non-zero number. Otherwise, the average will approach zero as the number of traces increases.

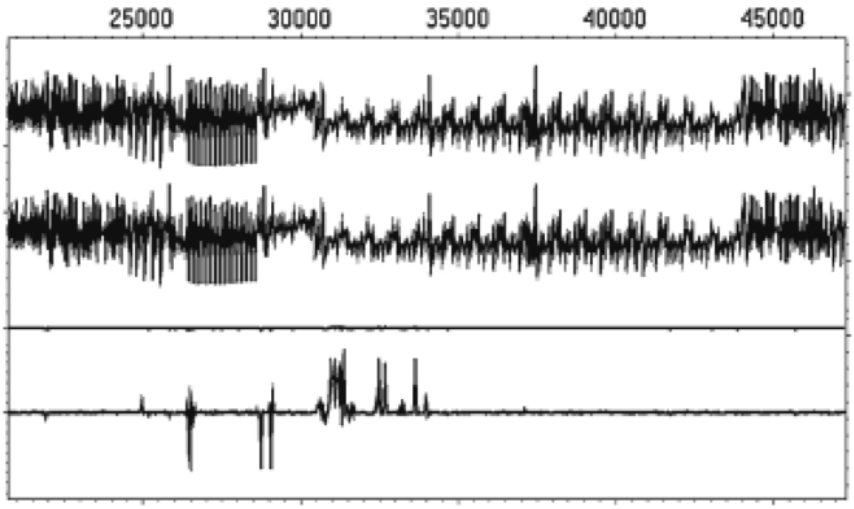


Figure 3.2 – Typical DPA results

Figure 3.2 illustrates a typical DPA result. The first two lines show two sets of traces. The difference between these two sets is shown on the third line. The fourth line shows the difference

between these two sets but is amplified by a factor of 15. We can observe that the partitioning is sound because, for some time slots, the difference is clearly non-zero.

Correlation Power Analysis (CPA) is another statistical method that uses the Pearson correlation coefficient for SCA. This method have been suggested in various paper [6, 11, 47]. In practice, the CPA is composed of four steps:

1. The attacker selects a generic model, denoted by $M(p, k)$ for the power consumption of the target device. This model will estimate how much power will be used for some part of the encryption of a plaintext, denoted by p, with a specific key, denoted by k.
2. The attacker captures traces from the target device. The traces represent the power consumption during the encryption process with different random plaintexts and an unknown key.
3. Then, the attacker makes several guesses, one for each possible key value. He obtains the corresponding estimations of the power consumption values, denoted by E_i returned by the model M, for each guess i.
4. Finally, the attacker computes the correlation between each estimation E_i and the measurement (captured from the target device). The highest correlation corresponds to the correct key value hypothesis.

3.3 Profiled Side-Channel Attacks

In profiled side-channel attacks, an attacker called A have access to the device to target denoted by D_{target}, and can have access to a clone of the device to target too, denoted by D_{clone}. We assume that the same encryption process E runs on D_{target} and D_{clone}. The attacker has full control of the device D_{clone}, so he can choose the plaintext p and the key k on this device. However the attacker does not have full access to the device D_{target}, so he can only choose the plaintext p (the key on the device D_{target} denoted by $k*$). The key k used on the device D_{clone} is variable, chosen by the attacker, whereas the key $k*$ on the device D_{target} is fixed and unknown. The goal of the attacker is to recover the secret key $k*$ on the device D_{target}.

If the attacker have only access to D_{target}, he can mount profiled side-channel attack if he knows some information about this device, for example, if the algorithm uses several keys and the attacker knows one of these keys. In a classic way, the attacker have access to D_{clone} in addition to D_{target} to mount the profiled side-channel attacks.

The profiled side-channel attacks are composed of two steps: *the profiling step* and *the attack step*. The goal of the profiling step is to estimate the leakage model with known key measurements and the attack step aims to use this model to guess key with unknown key measurements. But

to lead a successful profiled side-channel attack, the difference between the two datasets needs to be as minimal as possible because the leakage model needs to be the same for each device.

Before estimating the leakage model, the attacker must have a list of N_p measurements called *a profiling set*, denoted by x_1,...,$x_{N_p} \in X_{profiling}$. Each measurement x_i into the profiling set corresponds to the physical leakage occurring during the encryption process $c_i = E(p_i, k_i)$ on the device D_{clone}. The *profiling set* is always a labelled dataset.

Once the attacker have enough measurements, he can perform a profiled side-channel technique (such as *Template Attacks*) or Machine Learning (ML) algorithms to estimate the leakage model. The leakage model represents the dependency between each trace x_i in the profiling set $X_{profiling_set}$ and the intermediate values Y_i of the process. More specifically, the intermediate variable is not directly the secret key k, but it depends on the secret key k and corresponds to an intermediate state during the encryption process E.

Once the attacker determines the leakage model, he needs to have a list of N_a measurements called *attack set*, denoted by x_1,...,$x_{N_a} \in X_{attack}$. An *attack set* is always an unlabelled dataset. Each measurement x_i into the attack set corresponds to the physical leakage that occurred during the encryption process $c_i = E(p_i, k*)$ on the device D_{target}. The attacker uses the leakage model and the attack set to predict/discover all intermediate value Y_i used on the D_{target}.

Template Attack (TA) is first introduced by Chari *et al.* [9]. TA is considered as a powerful profiled side-channel attack. The principle of TA is based on the same principle as the *generative model*. From a statistical point of view, a generative model is a model of the conditional probability of the observable X, given a target Y, symbolically, $P(X|Y = y)$. In TA the observable X represents the electromagnetic or power traces and the target Y represents all possible key values. Thus, we observe a certain amount of couples $(x_i, y_i)i, ..., N_p$. in order to well establish the conditional probability between traces and each possible key value.

Deep Learning (DL)-based side channel attacks with convolutional neural network is introduced by Prouff *et al.* [44]. The DL is considered to be the most powerful profiled side-channel attack because DL is still efficient even in the presence of some countermeasures (presented in Section 3.5). Different approaches of DL for side-channel attacks will be presented with more details in Section 3.6.

3.4 Metrics on Side-Channel Attack

3.4.1 Leakage Quantification

A single trace can contain a certain amount of Points of Interest (PoI). A large number of PoI plunges us into high-dimensional data. In the context of profiled attacks, it may cause some troubles when estimating the model. Indeed, modelling the leakage gets harder with the

dimension of the problem up to the point where it is untractable. Some techniques exist to reduce the dimension of the data. The main used in a side-channel attack is the Signal-to-Noise Ratio (SNR) metric. The SNR is the ratio between the deterministic data-dependent leakage and the remaining noise:

$$SNR = \frac{\mathrm{Var}(\mathrm{E}(X|Y))}{\mathrm{E}(\mathrm{Var}(X|Y))}. \tag{3.1}$$

In the equation 3.1:

— X represents the physical measurements, e.g. electromagnetic measurements

— Y represents the target intermediate variable.

3.4.2 Evaluation Metric

The attacker's goal is to recover the secret key with the fewest number of traces possible. To evaluate the efficiency of his attack, we use the Guessing Entropy (GE) metric. The GE represents the ranking of the secret key k^* in the list of possible key byte values. When the attacker directly recovers the correct key the mean rank equal to 0 because the correct key is at the first position in the list. Otherwise, if the mean rank is greater than 0, the attacker still have to perform a small brute-force step to recover the correct key.

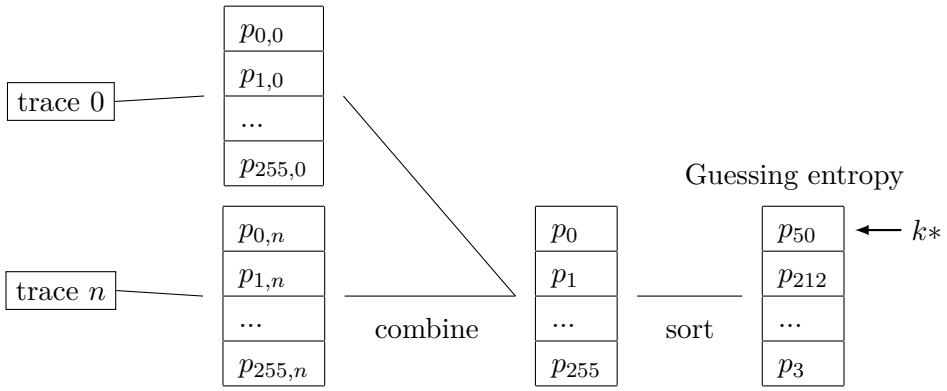


Figure 3.3 – Guessing entropy metric

Figure 3.3 describes an example of profiled attacks where the GE equals 0. In detail, we have:

— $p_{k,j}$ represents the probability that the key k is used in the trace j

— p_k represents the average probability that the key k is used in all traces.

— k^* represents the secret key.

3.5 Countermeasure in Side-Channel Attacks

There are many ways to protect cryptographic implementations against SCA. The main idea is to break the correlation between physical traces and sensitive data. The dependency between the traces and the sensitive data can be reduced by adding some noise, this effect reinforces the impact of countermeasure. Noisy traces are more difficult to attack than clear traces for an attacker. Thus, the number of traces required by the attacker to guess the correct key will increase. In the following section, we present the most common countermeasures from the state of the art.

3.5.1 Masking Countermeasure

The aim of the **masking countermeasure** is to delete the statistical link between some first moments of the measurements and the intermediate value. Masking countermeasure is the main countermeasure used in SCA [8, 12, 48]. The masking countermeasure boils down to divide the intermediate value into several parts by mixing it with random data. Those parts (called *shares*) are then used to perform an equivalent computation. All these shares are statistically independent from the intermediate value and are necessary to recover the final output of the operation.

3.5.2 Hiding Countermeasure

Another known countermeasure is the **hiding countermeasure**. The goal of the hiding countermeasure is to make the attacker's measurements too noisy to be exploitable [10, 15, 14]. The hiding countermeasure is often implemented by randomizing the power, or ElectroMagnetic (emission) (EM) traces. These countermeasures allow us to obtain a certain level of randomization, desynchronization or misalignment into the traces and thus hide the dependency between the intermediate value and the side channels. It is harder for an attacker to find the value of an intermediate variable when using desynchronized or misaligned traces instead of synchronized ones.

— **Gaussian Noise** is the most common type of noise that it found in SCA. The environment, the condition of measurement can introduce noise into the traces. Some researchers [68] implement Gaussian noise at the software level in order to simulate a noisy environment. The noise in this simulation is intentionally introduced by adding a uniformly distributed random value between a certain interval to each point of the trace.

— **Desynchronization** is different from the Gaussian Noise because it adds randomness into the time domain. The three most used desynchronization countermeasures are **Random-delay interrupt**, **Clock jitter** and **Shuffling**. Random-delay interrupt adds randomness

to the time domain locally. This kind of countermeasure is implemented at the software level. It inserts some fragments with no links to the computation and random duration. Clock jitter is the hardware version of random-delay interrupt [7] but affects all the clock cycles during the computation. Shuffling is a classical method to shuffle the traces by shuffling order of operations when possible.

3.6 Regular Deep Learning-based Attacks

The first work using ML techniques in SCA shows that Support Vector Machine (SVM) and Random Forest (RF) are effective profiled SCA [41, 30]. Indeed, TA is optimal from an information-theoretic point of view [28], however, when the set of measurements in the profiling phase is limited SVM can be more efficient due to the underlying estimation problem [27]. More recently, DL techniques are even more advantageous in several settings. Using the advantages of DL in SCA is becoming a very "interesting" topic, with many published works over the last few years.

The first work on SCA based on DL techniques is introduced by Maghrebi *et al* [44]. Like the template attack, a DL technique is based on estimating a *leakage model* for each possible value y_i of the targeted sensitive variable Y during the training step. During the attack step, these *leakage models* are involved to output the most likely key k^* used during the acquisition of the attack traces set. In SCA, an attacker is interested in the estimation of the probability of each possible value $\hat{y}_i$ deduced from a key hypothesis.

Maghrebi *et al.* [44] demonstrates that DL-based attacks are more efficient than template attacks when targeting either unprotected or protected cryptographic implementations (with masking countermeasures). The authors make the first experiment on unprotected hardware(FPGA) of the Advanced Encryption Standard (AES) implementation. They investigate four different DL-based attacks (AutoEncoder (AE), Convolutional Neural Network (CNN), Long Short-Term Memory (LSTM), MultiLayers Perceptron (MLP)). All DL-based attacks successfully recover the correct key, but some DL-based attacks require more traces than others. CNN and AE obtain the best performance and required half as many traces as the template attack to recover the correct key. Finally, the LSTM does not perform better compared to the other DL-based attack because the leakage of this device is not time-dependent.

MLP is investigated with and without Principal Component Analysis (PCA) as a preprocessing. The authors show that using MLP without PCA leads to better performances than with PCA. Authors assume that PCA probably removed some data components which are informative for the MLP. This assumption shows the interest of using the DL technique compared to a classical one that uses dimensionality reduction. In DL, during the learning process, the network will improve itself on the task of extracting the relevant data and classifying the correct key. A Deep Neural Network (DNN) will never remove data that is informative for the attack performed afterwards. On the opposite, a classical approach will use a dimensionality reduction technique to select some relevant features using some heuristic that will not perfectly fit the following attack (for instance based on SNR which is a mono-variate metric).

The authors make a second experiment on a software implementation of an unprotected AES implementation. They use the ChipWhispererCapture Rev2 board to acquire the traces.

As in their first experimentation, the best performance is obtained by CNNs and AEs. The AE requires only 20 traces on average to discover the correct key, whereas the template attack requires between 80 and 100 traces on average.

Eventually, the authors make a last experiment on a first-order masked AES implementation still using the ChipWhisperer-Capture Rev2 board. They show that DL-based attacks perform well against masked implementation because AE,CNN,MLP require between 500 and 1,000 traces to recover the key. The template attack on the masked implementation needs more traces to reach a mean rank equal to zero, the number of traces required by the template attack is more than 1,000.

3.6.1 ASCAD Network

Benadjila *et al.* [54] propose the first in-depth study of DL algorithms when applied to the context of SCA, and introduce a new database named **ASCAD**. The ASCAD database permits checking the efficiency and accuracy of ML and DL algorithms applied to SCA with different levels of countermeasure. The authors propose three variants of the ASCAD database:

— without jitter countermeasure,

— with a maximum jitter of 50 points [1],

— with a maximum jitter of 100 points.

For each variant of the ASCAD database, the authors give two datasets: a profiling set and an attack set. Each dataset contains the traces corresponding to the profiling step or the attack step, the labels, and some metadata (e.g. key, plaintext, ciphertext, mask, and desynch). The number of profiling and attack traces is the same for the different variants. The authors acquire 60,000 labelled traces and chose 50,000 for the profiling set and the remaining 10,000 for the attack set.

In addition to their in-depth study of DL algorithms, the authors investigate the question of hyper-parameters tuning for the class of MLP and CNN to design an optimal architecture of the network for SCA. The authors start to investigate the choice of the following hyper-parameters for MLP architecture: the number of layers, the number of units of each layer and the activation functions. The best MLP, denoted MLP_{best}, is obtained by using a number of layers equal to 6, a number of units of each layer equal to 200, and "ReLU" as the activation function. The full architecture of MLP_{best} is described in Figure 3.4.

For CNN architecture, the authors investigate a list of parameters and their choice. Table 3.1 describes the list of tested parameters and the best one. The best CNN, denoted CNN_{best}, is obtained by using a number of blocks equal to 5, a number of convolutional layers per block

1. The jitter countermeasure is artificially added by inserting a random offset.

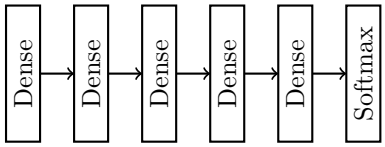


Figure 3.4 – Best MLP network for ASCAD

equal to 1, a number of filters equal to 64, 128, 256, 512, 512), a kernel size equal to 11, a number of fully connected layer equal to 2, "ReLU" as activation function, "Average" as pooling layer, and "same" as padding. The full architecture of CNN_{best} is described in Figure 3.5.

Parameter	**Range**	**Choice**
Epochs	{10,25,...,100,150}	up to 100
Batch Size	{50, 100, 200}	200
Blocks	[2..5]	5
Conv layers	[0..3]	1
Filters	$\{2^i; i \in [4..7]\}$	64
Kernel Size	{3,6,11}	11
Dense layers	[0..3]	2
Activation function	ReLU, Sigmoid, Tanh	ReLU
Pooling layer	{Max, Average, Stride}	Average
Padding	{Same, Valid}	Same

Table 3.1 – Benchmarks Summary in ASCAD

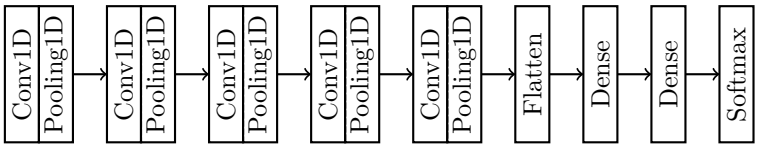


Figure 3.5 – Best CNN network for ASCAD

Finally, the authors evaluate four models, the two best models MLP_{best}, CNN_{best}, VGG-16 and a Template attack (combined with PCA). On synchronized traces, the performance of the models are similar, however, we notice that the template attack gives slightly better results than the others. Template attacks need less than 200 traces to obtain a mean rank equal to 0 whereas the other models require a bit more than 200. On the desynchronized traces, the model CNN_{best} outperforms all the other models. To reach optimal accuracy, only 75 epochs were needed. The model VGG-16 can obtain good performances with a higher number of epochs but it is somehow expected because it is designed for image classification and not for SCA.

Template attack combined with PCA provides similar results to CNN_{best} as long as the level of desynchronization remains low. MLP_{best} obtain good performances on synchronized traces but is very sensitive to jitter countermeasures.

3.6.2 Zaid Network

Zaid *et al.* [73] propose a first study of DL techniques based on some visualization techniques, such as *Weight visualization*, *Gradient visualization*, and *Heatmaps* introduced by Masure *et al* [45]. They propose a methodology for building efficient CNN architectures in terms of attack efficiency and network complexity, for multiple implementations of the AES on embedded devices. They evaluate their methodology by using public datasets with and without countermeasures. They demonstrate that their model, which we will refer to as the *Zaid* network, outperforms the best CNN model presented in the ASCAD paper [54], while significantly reducing the network complexity.

Authors use *Weight visualization* to evaluate the impact of performance related to the convolutional part. If the feature selection made by the convolutional layer is efficient, neurons of the first dense layer where information leaks will be evaluated with high weight during the training step. A weight with a large value means that the feature used by the network for its predictions is important. Thus, by visualizing the weight, they can understand which neurons have a positive impact on the classification. In addition to *Weight visualization*, they use *Heatmap* to understand which feature is selected by each filter. The *Heatmap* permits us to understand the role of each filter and determine which hyperparameter is the most suitable for feature selection.

The optimal architecture of CNN proposed by Zaid have one block composed of one convolutional layer, one "BatchNormalization" layer, and one "Average" pooling layer, follow by one "flatten" layer and three dense layers. The convolutional layer contains 4 filters, have a kernel size equal to 1, uses "SeLU" as the activation function, and the padding called "same". The padding known as "same" is a padding that distributes zeros uniformly to the input's left and right or up and down. Each dense layer is composed of 10 neurons, uses HE uniform [25] as the kernel initializer, and "SeLU" as the activation function (described in equation 2.12), except for the last dense layer which have a "softmax" activation function. The full architecture of the Zaid network is described in Figure 3.6.

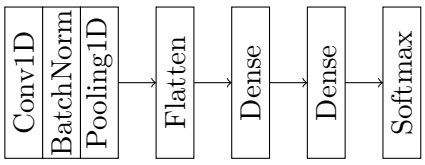


Figure 3.6 – Zaid network

First, to evaluate the performance of their model, the authors test it on synchronized traces for both unprotected and first-order masked AES. By visualizing the weights, authors notice that their network recognizes relevant information related to the mask and masked value. Their network is 3930 times less complex than the original paper [54], and it successfully guesses the correct key with 191 traces, whereas, in the original paper, 1146 traces were needed to obtain the same performance.

Then, the authors test their model on desynchronized traces with a certain level of random delay (50 and 100) and first-order masked AES. The heatmaps used by the authors permit to demonstrate that their network recognizes some influential patterns to predict the correct key, these patterns correspond to different leakages positions. For a random delay equal to 50, they success to reduce the number of traces from 5000 to 244, and the complexity of CNN is divided by 763, compared to the original paper [54]. For a random delay equal to 100, the Zaid network needs a number of traces equal to 270, whereas, in the original paper, it converges using 5000 traces.

3.6.3 NoConv1 Network

Wouters *et al.* [66] proposes a critical review of the paper by Zaid *et al.* [73]. They propose a new DNN architecture, called *NoConv1*, that has fewer parameters than the *Zaid* network while maintaining a similar performance, thus the number of parameters is reduced on average by 52%.

They demonstrate that the first convolutional layer from the *Zaid* network can be replaced by preprocessing techniques. They evaluate their network (where the convolutional layer is replaced by some normalization technique) and compare it to the *Zaid* network on synchronized and desynchronized traces. The normalization techniques used are feature standardization and feature scaling. For the synchronized traces, both networks give quite similar results, thus they prove that the convolutional layer can be replaced by a normalization technique. For the desynchronized traces, the *NoConv1* network performs slightly worse than the *Zaid* network but it is faster to train.

The *NoConv1* network is composed of one "Average" pooling, one "flatten" layer, and three dense layers. All dense layers have 10 neurons and "SeLU" as the activation function, except for the last dense layer which have a "softmax" activation function. The full architecture of the NoConv1 network is described in Figure 3.7.

Authors also provide a different interpretation from the one of Zaid *et al* [73] based on weight visualization. One conclusion make by Zaid *et al.* is that a model trained with a larger filter size leads to worse performances, but the results present by Wouters *et al.* suggest the opposite. On our side, we think that a low number of filters can lead to worse performances. The number of filters will determine how much information (pattern) will be extracted and used by the network,

and a network with little information will always have more difficulty to converge than a network with more. However, we also believe that an excessive number of filters can lead to a reduction in performance because in this case, the information extracted by the network could appear too specialized.

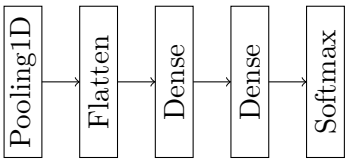


Figure 3.7 – NoConv1 network

Since the interest of DL [54, 73, 66] for the side-channel attack, the traditional attack has been less studied. Lichao *et al.* [70] claims that TA can continue to be efficient and to outperform DL-based side-channel attacks. They highlight the fact that the TA have a certain advantage compared to DL techniques, e.g. if not dimension reduction technique is applied on the input, TA does not have any hyperparameters to tune. They propose a new DL-assisted TA capable of breaking protected implementation. Their model is a combination of DL and TA. The DL part's aim is to extract relevant features from traces. Deep learning uses a triplet network [2]. The features (extracted by the DL) are then fed into the TA to discover the key. They evaluate their model on two publicly available datasets [3] and target two labels [4], and they compare the performance of their model with many networks fine-tuned for SCA [73, 69, 55, 52]. Also, they compare their model to TA with some dimensionality reduction techniques, such as PCA and AE. Their results show that TA assisted with the DL technique can outperform some networks fine-tuned for SCA. They also show that DL-assisted TA is better for feature engineering than PCA and AE.

Lichao *et al.* [70] presents interesting work because they use the advantage of the DL network to extract relevant features and improve the efficiency of TA. Their work demonstrates that the DL network can be used not only to discover the correct key, as a direct SCA but also to increase the efficiency of existing techniques.

2. The triplet network is evolved from the Siamese network, it is a stack of three networks that use shared weights and a triplet loss.
3. The two used datasets are the two versions of ASCAD dataset: `https://github.com/ANSSI-FR/ASCAD`.
4. The two labels are the intermediate value of the AES cipher and the Hamming weight.

3.7 Motivation of the thesis

In the previous section, we have seen the advantages to mount profiled attacks rather than non-profiled attacks because of the fewer attack trace requirements. Also, we have seen the advantage of using DL techniques instead of classical profiled attacks such as TA. Mostly, DNN is used with data that have been measured under ideal conditions, with traces coming from the same system as the targeted one with as little noise as possible. This assumption guarantees a small variation between the profiling set and the attack set and thus ensures that the performance of the DNN will be the same on the profiling set and the attack set.

In some real-world situations, it is not practical to use the same measurement equipment for both profiling and attacking. The assumption that profiling and attacking are carried out using the same measurement setup is assumed in the majority of published publications. We think that others sources of information from other devices or other probes can be beneficial for SCA because the few papers that have investigated this approach have shown an improvement over the traditional technique.

In this thesis, we investigate how deep learning can take profit from other sources of information through 3 contributions. Each contribution presents a new attacking method that is based on using the original but also some other available source of information. We refer to these three ways of attack as multi-channel attacks, transferred attacks, and translated attacks. Considering that these attacks are different, we will present them in individual chapters namely Chapter 5,6 and 7 and explain why our approach can be beneficial compared to the current related work. In the following Chapter 5, 6 and 7 we describe different attacks that we used in our contribution namely regular attacks, pre-trained attacks, multichannel attacks, transferred attacks and translated attacks. In addition to these three ways of attack, we present preliminary experiments using DL techniques to determine the optimal hyperparameters of our networks.

Chapter 4

Motivation and Implementation details

4.1 Implementation details

In Chapter 5, 6 and 7, we conduct our studies based on three existing networks from the state-of-the-art of Deep Learning (DL)-based side-channel attacks namely ASCAD [54], Zaid [73], and Noconv1 [66] presented in Chapter 3. We focus on the Advanced Encryption Standard (AES)-128 encryption code as the one from ASCAD [54]. The learning curves of these three networks are very different to have a generic working strategy for all our experiments. We cannot use an early stopping method, which means stopping the training of the network once its loss will stop decreasing because in Side-Channel Attack (SCA) we observe that even when the loss starts to increase the network sometimes still learns relevant things: it can be due to the fact that the loss is not perfectly suited to the SCA context. To determine the ideal number of epochs for each network, we conducted a preliminary analysis, which we provide in Section 4.3. Three criteria were used to determine the number of epochs: accuracy, loss value, and mean rank on the validation set. The batch size for the three networks has been set at 128 because it is the most adapted to the available computing hardware in the lab.

4.2 Measurement setup

4.2.1 AVR platform

The database called "DATABASE_AVR" was created in order to have a setup similar to the one used for producing the ASCAD database. However, we used an STK500 board instead of a smart card to earn time (the STK board already being part of the team platform). The target we used for experiments is a raw AtMega8515 microcontroller on the AVR STK500 platform [1]. We used the AES-128 encryption code as the one from ASCAD which is protected using a masking countermeasure [1], the compiler optimization flag was set to `-O0` but we did not embed

1. For ASCAD a smart-card embedding this microcontroller has been used.

the SOSSE operating system. The chip frequency was set to 3.686MHz. The measurements were obtained using different Langer near-field EM probes (two RF-B 0,3-3 and one RF-K 7-4) connected to 30dB amplifiers the overall having a bandwidth maximum frequency of 3GHz. The signal was then digitized by an RTO2014 oscilloscope from Rohde & Schwarz having a bandwidth of 1GHz (thus being the limiting element of the acquisition chain).

Algorithm 1: Secure AES Implementation with Table Recomputation

Input : a 16-byte plaintext $(p[1], \cdots, p[16])$,
an 18-byte mask vector $(r[1], \cdots, r[16], r_{\text{in}}, r_{\text{out}})$,
and a 16-byte master key $(mk_1, \cdots, mk_{16})$

Output : a 16-byte ciphertext $(c_1, \cdots, c_{16})$

steps] function **MaskedAES:**

// SBox recomputation

for $i = 0$ ***to*** 255 **do**

1] $\quad \text{sbox}^\star[i] \leftarrow \text{sbox}[i \oplus r_{\text{in}}] \oplus r_{\text{out}}$

// Initialization

for $i = 1$ ***to*** 16 **do**

2] $\quad \text{state0}[i] \leftarrow p[i] \oplus r[i]$

3] $\quad \text{state1}[i] \leftarrow r[i]$

4] $\quad \text{key}[i] \leftarrow mk_i$

// AES processing

for $round = 1$ ***to*** 10 **do**

/* Key scheduling */

5] $\quad (\text{key}[1], \cdots, \text{key}[16]) \leftarrow \text{KeyScheduling}(\text{key}[1], \cdots, \text{key}[16])$

for $i = 1$ ***to*** 16 **do**

/* AddRoundKey and SubBytes */

6] $\quad \quad \text{state0}[i] \leftarrow (\text{state0}[i] \oplus \text{key}[i] \oplus r_{\text{in}}) \oplus \text{state1}[i]$

7] $\quad \quad \text{state0}[i] \leftarrow \text{sbox}^\star[\text{state0}[i]]$

8] $\quad \quad \text{state0}[i] \leftarrow (\text{state0}[i] \oplus \text{state1}[i]) \oplus r_{\text{out}}$

/* ShiftRows */

9] $\quad (\text{state0}[1], \cdots, \text{state0}[16]) \leftarrow \text{ShiftRows}(\text{state0}[1], \cdots, \text{state0}[16])$

10] $\quad (\text{state1}[1], \cdots, \text{state1}[16]) \leftarrow \text{ShiftRows}(\text{state1}[1], \cdots, \text{state1}[16])$

/* MixColumns except for the last round */

if $round \neq 10$ **then**

11] $\quad \quad (\text{state0}[1], \cdots, \text{state0}[16]) \leftarrow \text{MixColumns}(\text{state0}[1], \cdots, \text{state0}[16])$

12] $\quad \quad (\text{state1}[1], \cdots, \text{state1}[16]) \leftarrow \text{MixColumns}(\text{state1}[1], \cdots, \text{state1}[16])$

// Last AddRoundKey

13] $(\text{key}[1], \cdots, \text{key}[16]) \leftarrow \text{KeyScheduling}(\text{key}[1], \cdots, \text{key}[16])$

for $i = 1$ ***to*** 16 **do**

14] $\quad c_i \leftarrow (\text{state0}[i] \oplus \text{key}[i]) \oplus \text{state1}[i]$

// Return the ciphertext

return $(c_1, \ldots, c_{16})$

Figure 4.1 – First-Order masked AES [54]

We focused on the first AES round with a sampling frequency of 1Gs per second and obtained traces containing 20K samples. Figure 4.1 describes, at the algorithmic level, the targeted AES

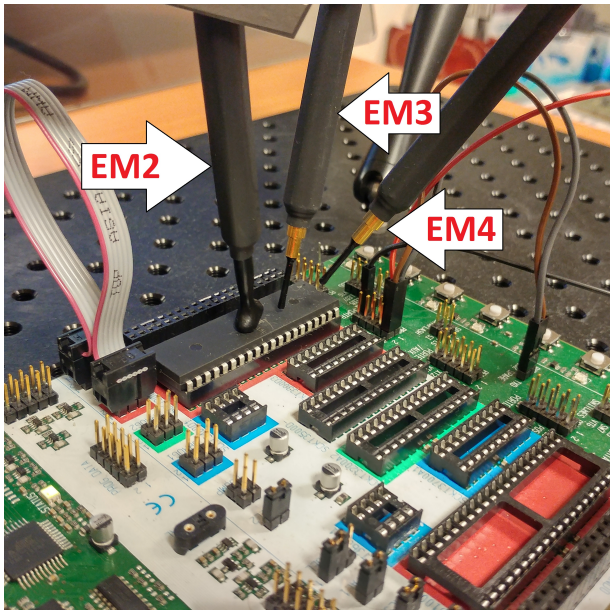


Figure 4.2 – Experiment setup on STK500 board

implementation. In our contribution we recover an intermediate value called snr4 [54], this operation corresponds to the masked sbox output (described in table 4.1).

Name target value	**Description**	**Definition**
$snr1$	unmasked sbox output	sbox(p[3] $\oplus$ k[3])
$snr2$	masked sbox output	sbox(p[3] $\oplus$ k[3]) $\oplus$ r_{out}
$snr3$	common sbox output mask	r_{out}
$snr4$	masked sbox output in linear parts	sbox(p[3] $\oplus$ k[3]) $\oplus$ r[3]
$snr5$	sbox output mask in linear parts	r[3]

Table 4.1 – List of target value

The main two differences between our setup and the one from [54] are

1. the fact that we use a raw microcontroller on a development board instead of a smart-card,
2. the measured quantity is EM radiation instead of power consumption.

We observed that the leakages we obtained have different behaviour than the one reported in [54], that is different time locations but also different signal amplitudes.

Figure 4.2 is a picture of the measurement setup on the STK500 board. We can see the three probes that capture ElectroMagnetic (emission) (EM). These three probes capture the following channels: EM2, EM3, and EM4. Channels EM2 and EM3 are close to the microcontroller while

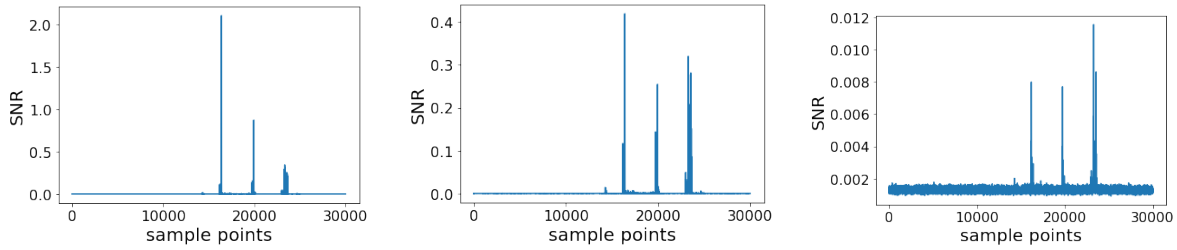


Figure 4.3 – SNR evaluation for each channel (from left to right: EM2, EM3, EM4) on AVR platform

EM4 is further from the microcontroller. It is intentional to have a probe that is further away from the microcontroller because it allows us to have noisier traces. We also add more variation between EM3 and EM4 by selecting two different kinds of probes, this permits us to add the variation between two probes in addition. Thus, we are able to treat in more detail how other sources of information can be used.

Figure 4.3 describes the Signal-to-Noise Ratio (SNR) evaluation for the channels EM2, EM3, and EM4. On the left, we have the SNR evaluation for EM2. In the middle, we have the SNR evaluation for EM3. On the right, we have the SNR evaluation for EM4. We observed that channel EM2 has the highest SNR value compared to channels EM3 and EM4. EM2 has a higher SNR value than EM4 because the probe is closer to the microcontroller, and the difference of SNR value between EM2 and EM3 may be due to the different type of probe used and/or the distance to the die that is small and located in the center. We also observed that channel EM3 has a higher SNR value than EM4 because it is also closer to the microcontroller compared to EM4.

4.2.2 ChipWhisperer platform

We decided to create another database called "DATABASE_STM32" to extend our experiment on several devices coming from the STM32 family. The database "DATABASE_STM32" contains several traces that come from different devices and different side-channels.

We used the chipwhisperer lite capture board combined with the CW308 UFO board on STM32Fx target devices. Similarly to the previous setup, we measured the first round of an AES-128 encryption, where we used the `TINYAES` implementation integrated in the chipwhisperer software. The chip frequency was set to 7.37MHz and the measurements are sampled at 4×7.37 Ms/s. Power consumption is collected through the measurement shunt on the CW308 UFO board. To capture EM signals, we used a Langer near-field EM probe (RF-U 5-2) connected to a 20dB amplifier.

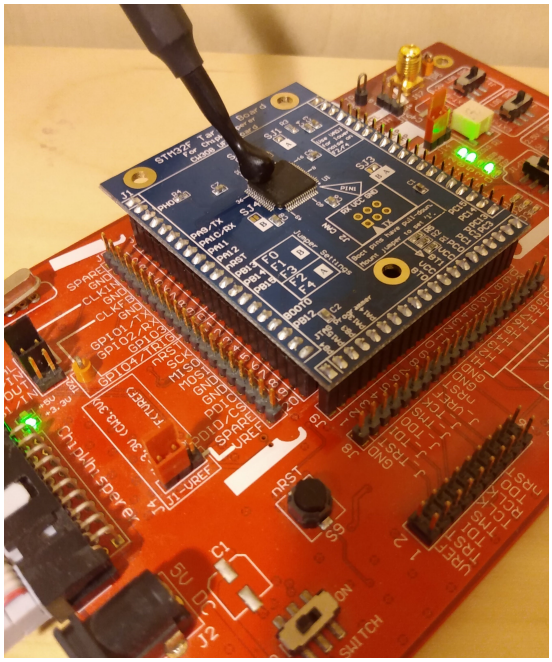


Figure 4.4 – Experiment setup for ChipWhisperer platform

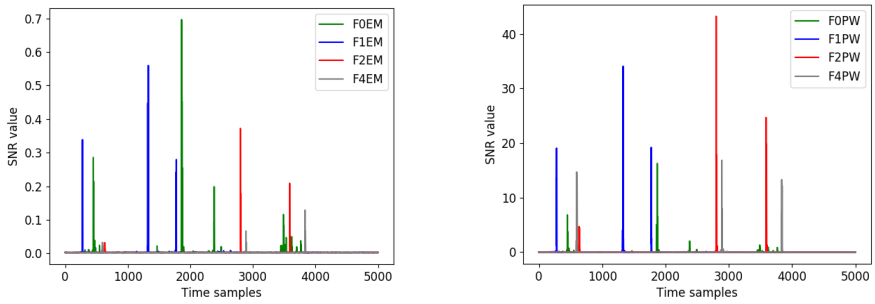


Figure 4.5 – SNR evaluation for each targeted device on STM32 platform

Figure 4.4 is a picture of the measurement setup on the ChipWhisperer (CW) platform. We can see a probe that captures EM for the different devices (F0, F1, F2, and F4). We also have a probe that captures the power consumption of these devices.

Figure 4.5 describes the SNR evaluation for the different devices (F0, F1, F2, F4) On the left, we have the SNR evaluation of these devices with EM. On the right, we have the SNR evaluation of these devices with power. We observed that for each device the leakage position is specific and also that we obtained different SNR values. Overall, we observed that power channels lead to larger SNR values than EM channels.

4.3 Preliminary analysis

We made a preliminary analysis in order to determine the optimal number of epochs that should be used for the three networks (ASCAD, Zaid, NoConv1) for our future contribution (in the next chapters). This analysis focuses on three networks because we wanted to show that our results can be applied to different networks. Concerning the choice of these networks, we chose the three most recurrent networks in the state-of-the-art at the time of our contributions.

The preliminary analysis is focused on the hyperparameter called "epoch", it is the parameter that define the number of iterations done by the network. The other hyperparameters are defined with constant values. The selected values are the ones respectively defined in the ASCAD [54], Zaid [73] and Noconv1 [66] seminal papers. Since "Epoch" was the most crucial hyperparameter for ensuring our neural networks' convergence, we made the choice to concentrate on it. The learning of the network is also affected by additional hyperparameters like "batch size," "learning rate," or "optimizer," however frequently their values are fixed. In order to be fair in our analysis, we prefer not to touch these parameters and to train as long as necessary on the new data sets. Table 4.2 describes the selected values of the other hyperparameters.

Hyperparameters	**ASCAD**	**NoConv1**	**Zaid**
Optimizers	RMSprop	Adam	Adam
learning rate	0,00001	0,00001	0,00001
batch size	128	128	128
Activation function	ReLU	SeLU	SeLU
Blocks	5	0	1
Filters	[64,128,256,512,512]	[2]	[4,2]
kernel initializer	Glorot uniform	Glorot uniform	He uniform
Kernel Size	11	0	1
Strides	2	2	2
Padding	Same	Valid	Same
Conv layer	5	0	1
Pooling layer	5	1	1
Dense layer	3	3	3
Units	4096	10	10

Table 4.2 – Summary of hyperparameters common to all sources

The preliminary analysis was done with three different architectures of neural networks: ASCAD [54], Zaid [73] and NoConv1 [66] (presented in Chapter 3). The number of epochs depends on the amount of useful information linked to the targeted label, and also depends on the architecture of the network. In a nutshell, a deeper architecture requires more data for the training, and more epoch to converge. The preliminary analysis takes into account the number

of epochs with three different metrics: the loss value (given by the loss function), the accuracy, and the mean rank on the validation set. Usually in the machine learning community, one just takes into account the loss value and the accuracy. However, for SCA we also consider the mean rank metric because it is more relevant than the former according to the objective that our neural network must reach.

We decided to choose these three metrics in order to have more information and more accuracy regarding the convergence of our networks. The loss allows us to give information about the summation of errors done by our model, and the accuracy provides information about the correct prediction done by our model. The loss and accuracy give global information but not enough information about the actual goal and in some situations, the learning of the network can stagnate. Picek *et al.* [53] demonstrated that loss and accuracy may not coincide with the SCA metric. The mean rank gives us specific information about the performance of the network to find the correct sub-key. The mean rank concerns only one sub-key (the intermediate value present in section 4.2.1) but in our case this is not a problem because we will target the same sub-key in each contribution. In case we would like to target another sub-key, it would be preferable to conduct another preliminary analysis, because another sub-key may require more epochs depending on the amount of data leakage. Thus this study is conducted to determine the ideal number of epochs for a given subkey. However, for a total attack on all the sub-keys (16 sub-keys) we could also use transfer learning to avoid relearning for all possible subkeys.

The preliminary analysis was done with

i) a reduced profiling set composed of 10,000 traces (resp. 2,000 traces) for the training set (resp. validation set),

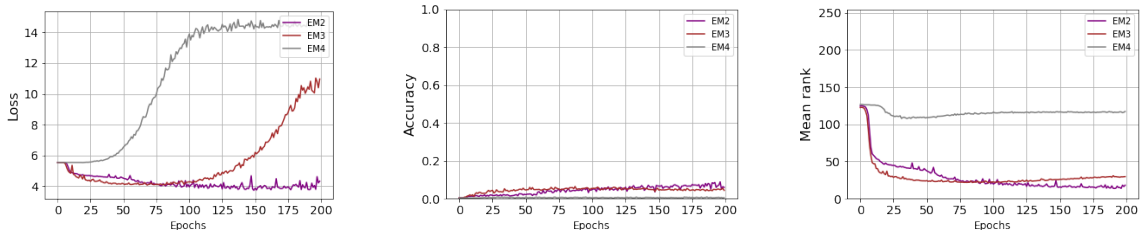
ii) a complete profiling set composed of 80,000 traces (resp. 20,000 traces) for the training set (resp. validation set),

In our contribution, we also wanted to study the performances of these networks with a limited amount of data, because in a more real context an attacker may find himself learning with a limited amount. So in addition to the 100% data acquired (which represents the complete amount of data), we also wanted to study the performance of these networks with 12% of data (which represents the limited amount of data). The interest is to see how in a concrete case the networks and these hyperparameters are impacted by this amount of data and to see how we can achieve/keep the same performance with this limited amount.

Finally, this preliminary analysis has been motivated by the need of ensuring that we selected the correct number of epochs for both databases used in our contributions.

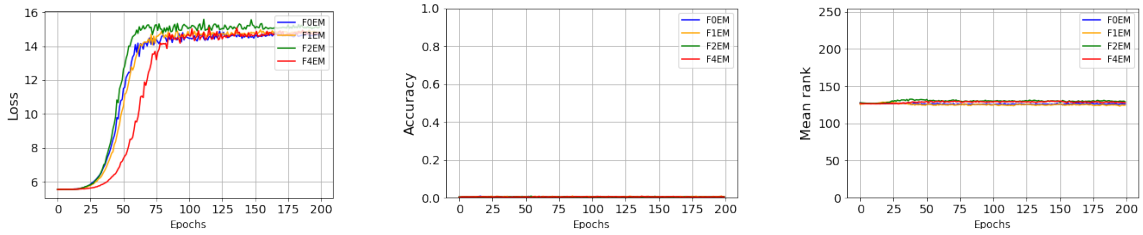
4.3.1 ASCAD network

Train with 12% of the database



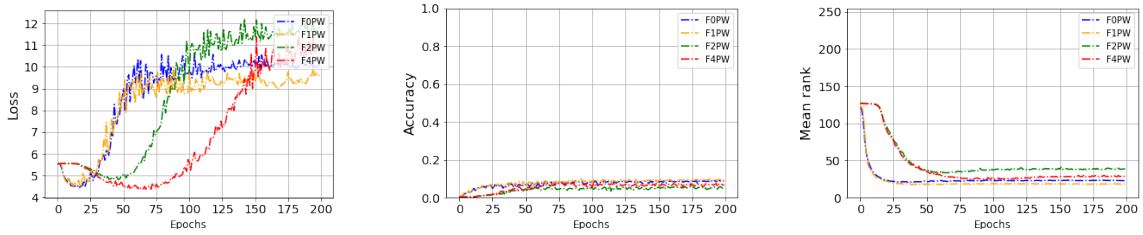
(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.6 – Training over 200 epochs using EMx



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.7 – Training over 200 epochs using FxEM



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.8 – Training over 200 epochs using FxPW and 12% of the database

We first made an analysis with the reduced profiling step (composed of 10,000 traces for training and 2,000 traces for validation). First, we define 200 epochs for all the sources of training (including the channels from the AVR platform and the side channel, and devices from the STM32 platform). Figure 4.6 shows the evolution of loss, accuracy, and mean rank on the validation set with 200 epochs for the AVR platform. Figure 4.7 and Figure 4.8 show the

evolution of loss, accuracy, and mean rank on the validation set with 200 epochs for the STM32 platform.

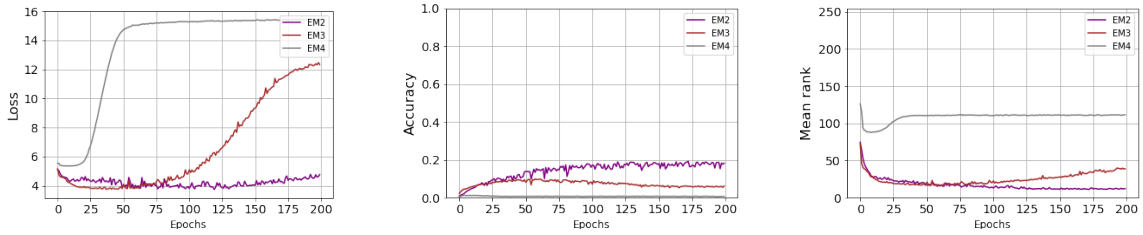
Figure 4.6a, Figure 4.7a, and Figure 4.8a show the evolution of loss on the validation set. We observe that the value of the loss increases at different times depending on the source used during the training. Generally, the value of the loss increases before 175 epochs which means the model is overfitting (and learning too well on the training set but not on the validation set). Figure 4.6b, Figure 4.7b, and Figure 4.8b show the evolution of accuracy on the validation set. For all experiments the accuracy increases but the number of epochs before reaching the best accuracy strongly depends on the source, e.g. for the source F1PW the best accuracy is reached with 25 epochs whereas, for the source F2PW, the best accuracy is reached with 50 epochs. We can see that the highest accuracy is at 15% which can be considered as weak. However, this accuracy is acceptable in side-channel attacks because our networks try to predict a subkey value among 256 possible values. In this case a random decision represents an accuracy of $1/256 = 0.0039$. So a network which obtains an accuracy higher than 0.0039 is a network which has acquired concepts and which will not give a random decision, in our case the 15% of accuracy confirms well that our network has acquired knowledge and converge towards a solution with a limited amount of traces. In other fields that use machine learning, we often end up with an accuracy of 80% to 99%. In the context of side-channel attacks, an accuracy of 80 to 99% would mean that our network can attack using only 1 or 2 traces. Currently, this accuracy is difficult to reach and we find rather accuracy orders ranging from 0.5% to 20% (the accuracy also depends on the target system and the countermeasures implemented on this system). Thus reaching an accuracy of 15% is quite acceptable.

Figure 4.6c, Figure 4.7c, and Figure 4.8c show the evolution of the mean rank on the validation set. We observed that depending on the source of the training, the mean rank can decrease or remain constant. Generally, we observe on the three metrics that the ASCAD network hardly converges on EM sources while good performances are obtained on power sources. We observed that 200 epochs are sufficient for the ASCAD network with most of the training sources and if we increase the number of epochs we risk that the model will be overfitting. In Figure 4.6c, we also distinguish two types of curve. Indeed the EM4 curve is different from EM2 and EM3, we see that EM2 and EM3 converge quickly while EM4 only very slightly decreases. This difference can be explained by the amount of information contained in each channel. Indeed the EM4 channel captures much more noise than EM2 and EM3 (because the probe is further from the microcontroller) which may explain the difficulty of the network to converge with this channel. Another difference is visible between the average ranks of FxEM in figure 4.7c and FxPW in figure 4.8b. This difference is also explained by the fact that the FxEM channels contain less information than the FxPW channels.

These figures are interesting because they confirm that the loss value is not the only metric

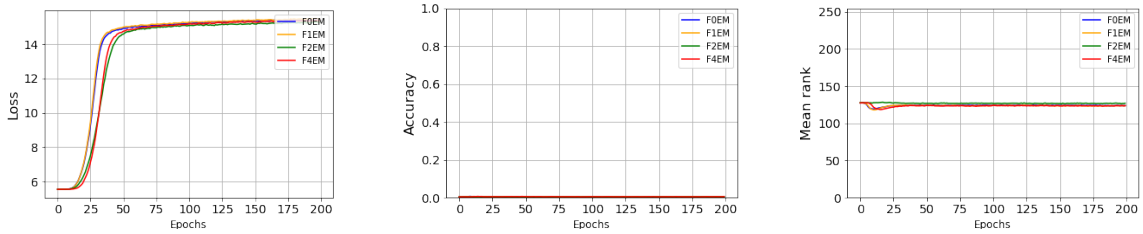
to take into account in SCA. Indeed, we see cases where the loss value starts to increase whereas the accuracy is still increasing and the mean rank is still decreasing.

Train with 100% of the database



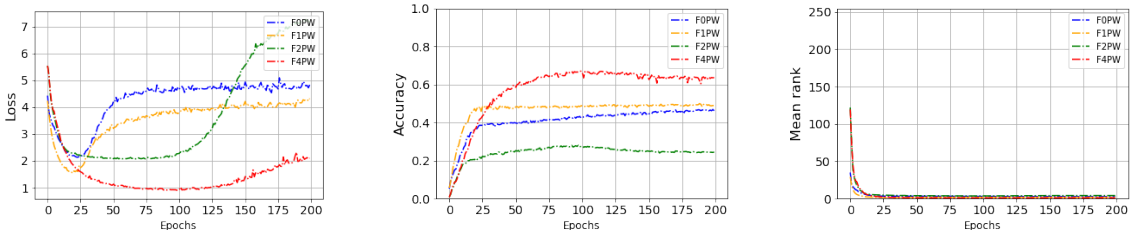
(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.9 – Training over 200 epochs using EMx



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.10 – Training over 200 epochs using FxEM



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.11 – Training over 200 epochs using FxPW and 100% of the database

We analyzed the ASCAD network using 80,000 traces for the training and 20,000 traces for the validation. As previously, we started by using 200 epochs for training all the sources. Fig 4.9, Fig 4.10, and Fig 4.11 show the evolution of loss, accuracy, and the mean rank on the validation set with 200 epochs using EM2, EM3, EM4, F0EM, F1EM, F2EM, F4EM, F0PW,

F1PW, F2PW, and F4PW. Figure 4.9a, Figure 4.10a, and Figure 4.11a show the evolution of the loss on the validation set. Figure 4.9b, Figure 4.10b, and Figure 4.11b show the evolution of the accuracy on the validation set. Figure 4.9c, Figure 4.10c, and Figure 4.11c show the evolution of the mean rank on the validation set.

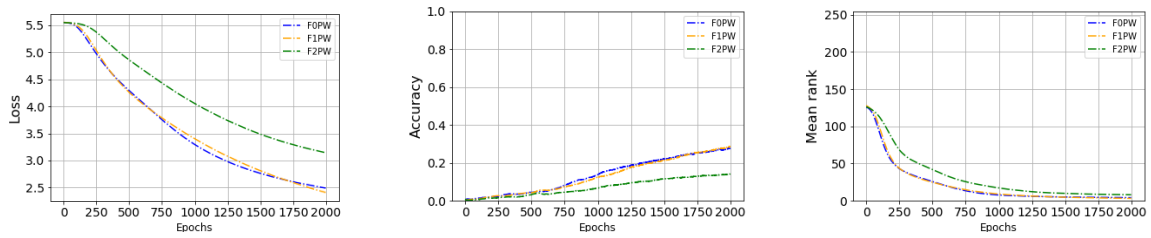
We observe a similar behaviour as in the reduced dataset experiments namely the ASCAD network converges well on power sources. We observe that the loss and accuracy increase at different time periods depending on the power source used during the training. We observe that the ASCAD network with a power source can reach a mean rank equal to 0 with less than 25 epochs. However, we observe that the loss increases quickly with EM source on F0, F1, F2 and F4. Generally, the value of the loss increases before 50 epochs which means the model is overfitting. Compared to the reduced dataset, we observe that the network also overfitted with a full dataset but it overfitted faster. We, therefore, believe that overfitting is not only related to the quantity of data but also to the quality of the data provided. We observe that the accuracy has difficulty increasing for EM4, and FxEM. For the channel EM4, we expected a low accuracy as with the reduced dataset because even using more traces, these traces still contain noise which makes learning still difficult. We observe that the mean rank hardly decreases for EM4, and FxEM. The assumptions stated above can explain this phenomenon. Nevertheless, an interesting point we noticed is that the mean rank of the FxEM channels decreases a little bit between epoch 0 and 25, while the loss and accuracy do not necessarily indicate an improvement. The mean rank allows us to see that the network has still acquired some knowledge at some point thanks to this slight decrease.

When we compared these metrics between EM and power, we can conclude that using power is more relevant than using EM.

4.3.2 NoConv1 network

We performed similar tests as the one from ASCAD but replacing the network by NoConv1.

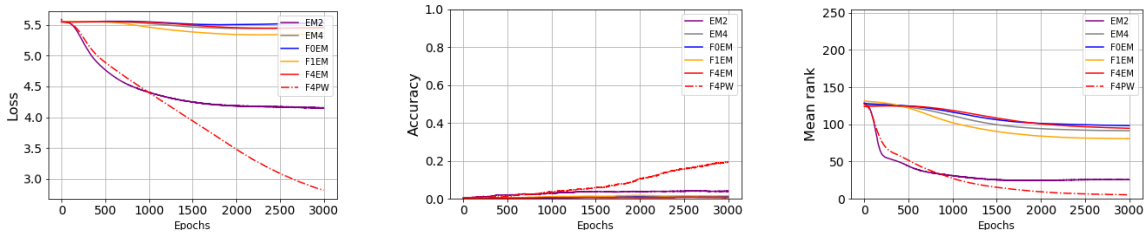
Train with 12% of the database



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.12 – Training over 2,000 epochs using F0PW, F1PW, F2PW and 12% of the database

We first used the reduced set (that is 10,000 traces for the training and 2,000 traces for the validation). First, we define 2,000 epochs for all the sources of training (including the channels from the AVR platform and the side channel, and devices from the STM32 platform). We defined more epochs than the ASCAD network as a starting point because the NoConv1 network is a more unstable network. Figure 4.12 shows the evolution of loss, accuracy, and mean rank on the validation set with 2,000 epochs using F0PW, F1PW, and F2PW. As previously, Figure 4.12 illustrates only the sources converging within 2,000 epochs (at maximum). Figure 4.12a shows that the loss value decreases since the beginning of the training. Figure 4.12b shows the same observation, that the accuracy grows since the beginning of the training. However, Figure 4.12c shows that the network quickly reaches a mean rank equal to 0 with at least 1,250 epochs. The evaluation of the mean rank on the validation set permits us to know that 2,000 epochs are sufficient for our network to converge to an optimal solution with F0PW, F1PW and F2PW.



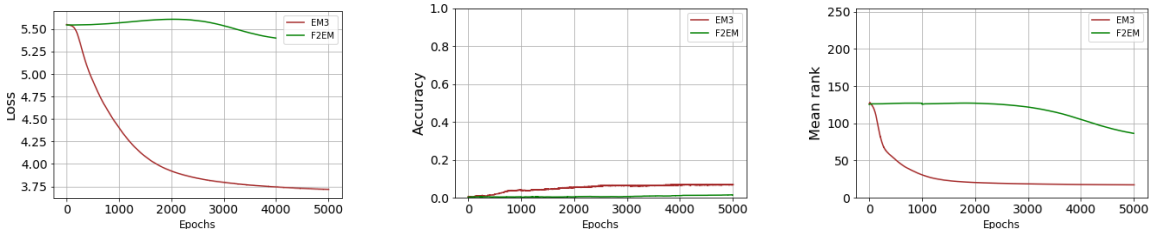
(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.13 – Training over 3,000 epochs using EMx, FxEM, FxPW and 12% of the database

Secondly, we extended the number of epochs to 3,000 for the other sources except for F2EM and EM3. Figure 4.13 shows the evolution of loss, accuracy, and mean rank on the validation set with 3,000 epochs using EM2, EM4, F0EM, F1EM, F4EM, and F4PW. Figure 4.13 illustrates only the source that converges with 3,000 epochs (at maximum). Figure 4.13a shows that the loss value decreases since the beginning of the training for source EM2 and F4PW, but hardly decreases for EM4, F0EM, F1EM, and F4EM. Figure 4.13b shows that the accuracy of F4PW is growing quickly, but for the other sources, the increase in accuracy is slower. For accuracy, we observe the same results as the ASCAD network, which is that the accuracy may seem low, but the network NoConv1 has acquired knowledge. These results are confirmed by the evolution of the mean rank.

Figure 4.13c shows that the mean rank decreased since the beginning of the training for source EM2 and F4PW, but decreases slower for EM4, F0EM, F1EM, and F4EM. The same observation applies to the loss value and the mean rank, however, the loss for the source F4PW is decreasing at 3,000 epochs but the mean rank remains constant and close to 0 at 3,000 epochs. The evaluation of the mean rank on the validation set permits us to know that 3,000 epochs are

sufficient for the noConv1 network to converge to an optimal solution with EM2 and F4PW. Similar to the previous experiments on ASCAD, we observe two types of curves for the mean rank. Like ASCAD, the NoConv1 network learns more easily with the data from FxPW than with the data from FxEM (for the same reason as stated for ASCAD).

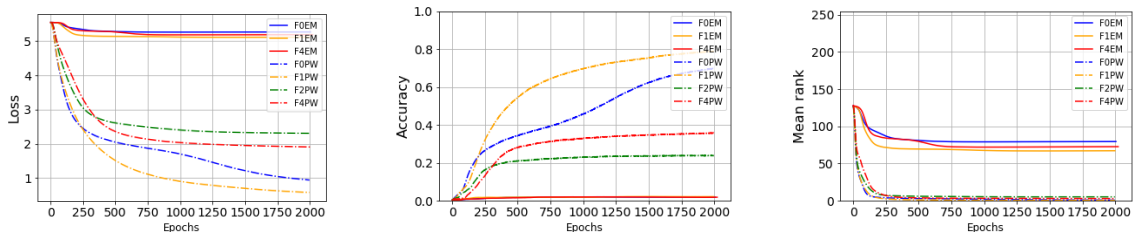


(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.14 – Training over 4,000 epochs using F2EM and 12% of the database

Third, we extended the number of epochs to 5,000 for the source EM3 and F2EM. Figure 4.14 shows the evolution of loss, accuracy, and the mean rank on the validation set with 5,000 epochs using EM3 and F2EM. Figures 4.14a and 4.14c show us some interesting things about source F2EM. We observed that the network takes at least 2,500 epochs before converging and decreases the loss value and the mean rank. In detail, we show that the loss value increases between 0 and 2,000 epochs, remains constant between 2,000 and 2,500 epochs, and finally decreases. Figure 4.14c shows that the mean rank remains constant between 0 and 2,500 epochs, and finally decreases. For EM3, Figure 4.14a shows that the loss value decreased between 0 and 4,000 epochs, and finally remains constant since 4,000 epochs. It shows that the accuracy increased between 0 and 3,0000 epochs and the mean rank decreased between 0 and 2,000 epochs to finally remains constant since 2,000 epochs.

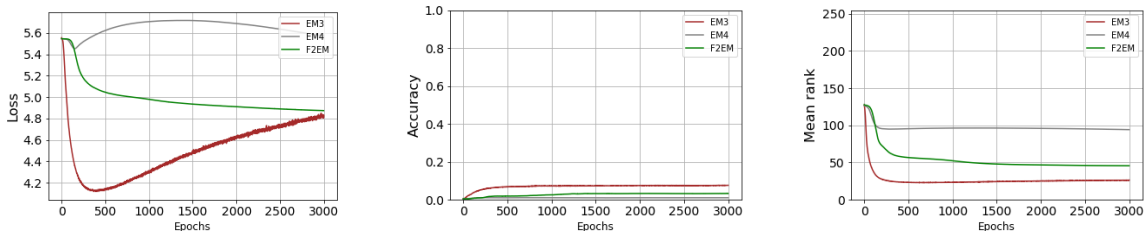
Train with 100% of the database



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.15 – Training over 2,000 epochs using FxEM, FxPW and 100% of the database

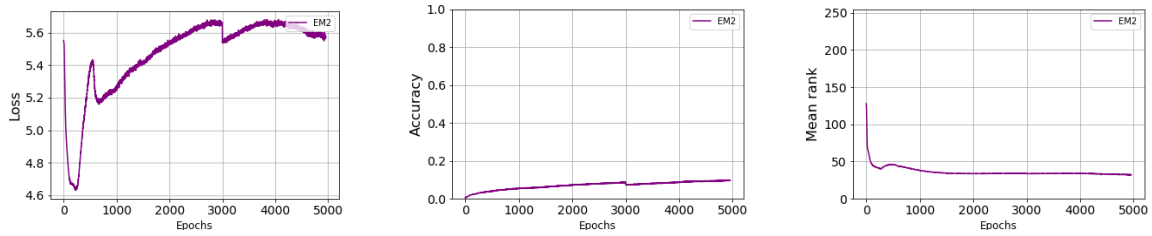
As in the previous analysis of the ASCAD network, we also analyzed the Noconv1 network by using 80,000 traces for the training and 20,000 traces for the validation. As the first analysis, we started our analysis with 2,000 epochs. Figure 4.15 shows the evolution of loss, accuracy, and the mean rank on the validation set with 2,000 epochs using F0EM, F1EM, F4EM, F0PW, F1PW, F2PW, and F4PW. Figure 4.15a shows that the loss value of the source F0PW, F1PW, F2PW, and F4PW decreased and remains stable since the beginning of the training, however, the loss value of the source F0EM, F1EM, and F4EM hardly decrease since the beginning. We had the same observation in Figure 4.15b, the accuracy for the source F0PW, F1PW, F2PW, and F4PW increases faster and is stronger than the accuracy for the source F0EM, F1EM, and F4EM. Figure 4.15c shows that the mean rank for the source F0PW, F1PW, F2PW, and F4PW converge to 0 with at least 250 epochs whereas the other sources (F0EM, F1EM, and F4EM) decrease a bit but do not reach a mean rank equal to 0.



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.16 – Training over 3,000 epochs using EM2, EM4, F2EM and 100% of the database

Next, we extended the number of epochs to 3,000 for the source EM2, EM4, and F2EM. Figure 4.16 shows the evolution of loss, accuracy, and the mean rank on the validation set with 3,000 epochs using EM3, EM4, and F2EM. Figure 4.16a shows a strange evolution of the loss value. For the evolution of the source EM3, we observed that the loss value decreases between 0 and 500 epochs, and grows between 500 and 3,000 epochs. For the evolution of the source EM4, we observed that the loss value decreases a bit between 0 and 200 epochs, growing between 200 and 1,500 epochs, then decreasing from 1,500 epochs. For the evolution of the source F2EM, we observed that the loss value decreases between 0 and 3,000 epochs. Figure 4.16a shows us how the variation between the channels and the devices can affect the loss during the training of the Noconv1 network. We observed that the NoConv1 network is sensitive to the issue of overfitting with the channels EM3 and EM4. Figure 4.16b shows that for all sources (EM3, EM4, and F2EM) the accuracy grew between 0 and 3,000 epochs. Figure 4.16c shows that for all sources the mean rank decrease between 50 and 500 epochs, and remains constant after 500 epochs.



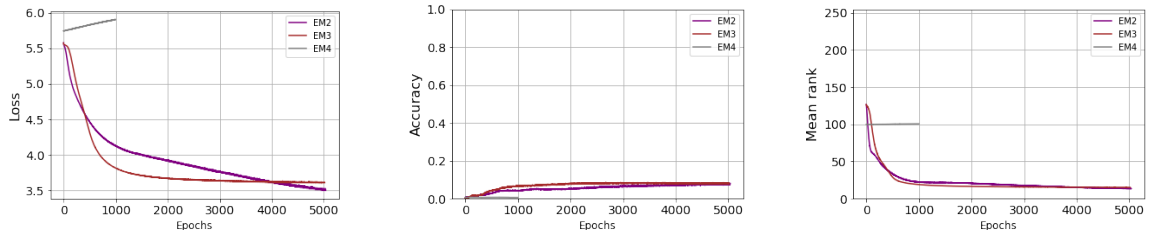
(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.17 – Training over 5,000 epochs using EM2 and 100% of the database

Finally, we extended the number of epochs to 5,000 for the source EM2. Figure 4.17 shows the evolution of loss, accuracy, and the mean rank on the validation set with 5,000 epochs using EM2. We observed that the loss for the source EM2 decreases between 0 and 100 epochs, and then grows to 5,000. We see the importance of determining the correct number of epochs in these results 4.17a because we can see that the network learns things before overfitting. We observed that the accuracy for the source EM2 increases between 0 and 5,000 epochs. We observed that the mean rank for the source EM2 decreases between 0 and 100, then increases a little bit, and finally decreases and remains constant until 5,000 epochs.

4.3.3 Zaid network

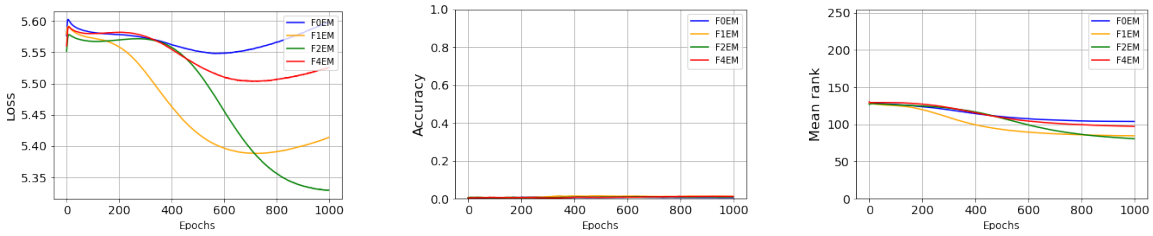
Train with 12% of the database



(a) Loss metric (b) Accuracy metric (c) Rank metric

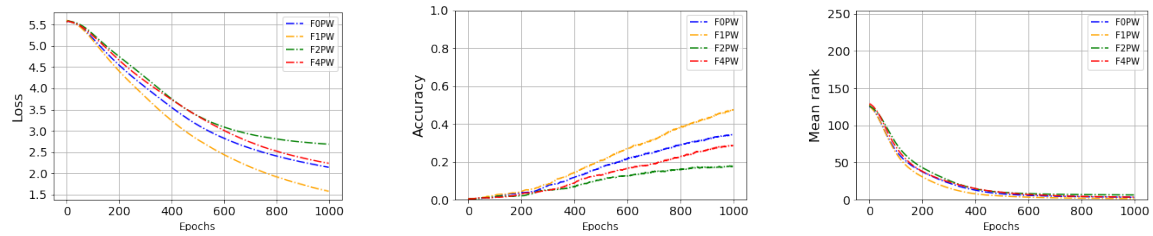
Figure 4.18 – Training over 1,000 epochs using EMx and 12% of the database

As a previous analysis with the ASCAD network and Noconv1 network, we made another analysis with the Zaid network by using 10,000 traces for the training and 2,000 traces for the validation. We define 1,000 epochs at the beginning of our analysis for all sources of training (including the channels from the AVR platform and the side channel, and devices from the STM32 platform). We defined more epochs than the ASCAD network such as the Noconv1 network because the Zaid network is also a more unstable network than the ASCAD network,



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.19 – Training over 1,000 epochs using FxEM and 12% of the database



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.20 – Training over 1,000 epochs using FxPW and 12% of the database

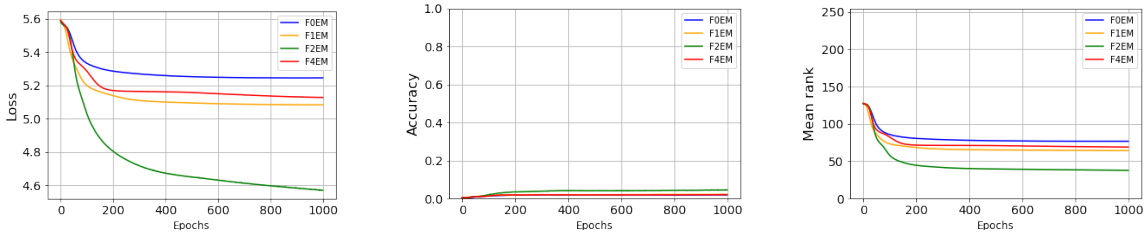
and thus required more epochs to converge. Figure 4.18, Figure 4.19, and Figure 4.20 show the evolution of loss, accuracy, and the mean rank on the validation set with 1,000 epochs using EM4, F0EM, F1EM, F2EM, F4EM, F0PW, F1PW, F2PW, and F4PW. These figures illustrate only the source that converges with 1,000 epochs (at maximum). In Figure 4.20a we observed that the loss value of all power channels decreases between 0 and 1,000 epochs. However, in Figure 4.18a the loss value of all EM channels remains constant.

Figure 4.20b shows that the accuracy of F0PW, F1PW, F2PW, and F4PW increases between 0 and 1,000. But in Figure 4.18b and Figure 4.19b the accuracy of EM4, F0EM, F1EM, F2EM, and F4EM do not increase a lot and remains constant. Figure 4.20c shows that the mean rank decreases and the Zaid network reach a mean rank equal to 0 for all source of power with 600 epochs. In Figure 4.18c we observed that the mean rank of the EM source decreases a little bit less than the power. We still observe these two groups of mean rank curves. Similarly to ASCAD and NoConv1, the Zaid network learns more easily on the FxPW data than on the FxEM data. We also see that despite the fact that the accuracy is not high, the mean rank is reduced which means that Zaid network has learned some concepts during the training.

Secondly, we have extended the number of epochs to 5,000 for the sources EM2 and EM3. Figure 4.18 shows the evolution of loss, accuracy, and the mean rank on the validation set with 5,000 epochs using EM2 and EM3. In Figure 4.18a, we observed that the loss value decreases

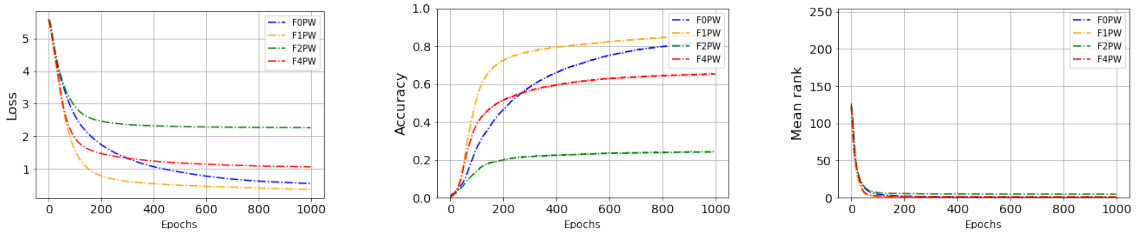
since the beginning of the training. The decrease in the loss value is correlated to the increase of the accuracy in figure 4.18b. Finally, we observed that the mean rank for source EM2 and EM3 decrease also since the beginning of the training. The mean rank did not reach a mean rank equal to 0, however, it decreased from 140 to 20.

Train with 100% of the database



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.21 – Training over 1,000 epochs using FxEM and 12% of the database

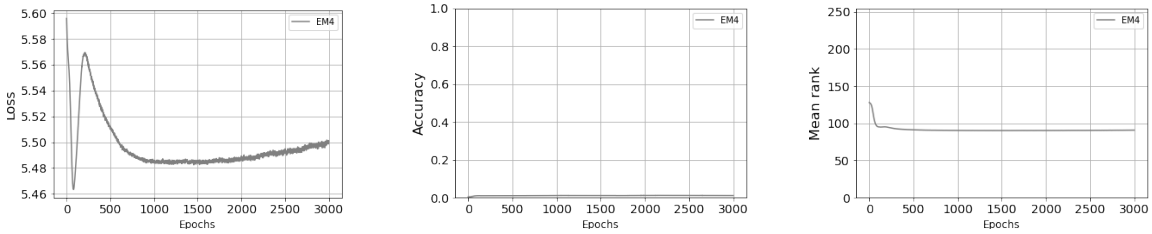


(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.22 – Training over 1,000 epochs using FxPW and 12% of the database

Like the two previous analyses with ASCAD and Noconv1 networks, we also analyzed the Zaid network by using 80,000 traces for the training and 20,000 traces for the validation. We started our analysis with 1,000 epochs. Figure 4.21 and Figure 4.22 show that the loss, accuracy, and mean rank on the validation set with 1,000 epochs using F0EM, F1EM, F2EM, F4EM, F0PW, F1PW, F2PW, and F4PW. In figure 4.21a and Figure 4.22a, we observed that the loss value decreased since the beginning of the training. However, the loss value for the F0EM, F1EM, F2EM, F4EM decrease more slowly than F0PW, F1PW, F2PW, and F4PW. In Figure 4.21b and Figure 4.22b, we observed that accuracy increased since the beginning of the training. We observed that the accuracy for the F0EM, F1EM, F2EM, and F4EM increase more slowly than F0PW, F1PW, F2PW, and F4PW. In Figure 4.21c and Figure 4.22c, we observed also that the mean rank for F0EM, F1EM, F2EM, and F4EM decreases more slowly than F0PW, F1PW,

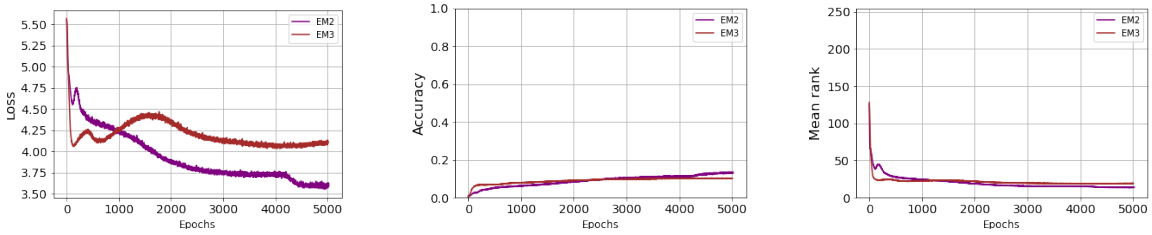
F2PW, and F4PW. Such as the ASCAD Network and NoConv1 network, we distinguish two groups of mean rank. The first group concerns all channels with electromagnetic emission and the second concerns all channels with power.



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.23 – Training over 3,000 epochs using EM4 and 100% of the database

Secondly, we extended the number of epochs to 3,000 for the source EM4. Figure 4.23 shows the loss, accuracy, and mean rank on the validation set with 3,000 epochs using EM4. We observed that the loss value reaches its minimal value at the beginning of the training between 0 and 100 epochs. We observed that the accuracy hardly increases. We observed that the mean rank decreases to 100 at the beginning of the training.



(a) Loss metric (b) Accuracy metric (c) Rank metric

Figure 4.24 – Training over 5,000 epochs using EM2, EM3 and 100% of the database

Finally, we extended the number of epochs to 5,000 for the sources EM2 and EM3. Figure 4.24 shows the loss, accuracy, and mean rank on the validation set with 5,000 epochs using EM2 and EM3. We observed that the loss value for source EM2 decreases a lot between 0 and 500 epochs and then remains constant but the loss value for source EM3 decreases until 5,000 epochs. We observed that the accuracy of the source EM2 and EM3 grow between 0 and 5,000 epochs. We observed that the mean rank for the source EM2 and EM3 also decreased between 0 and 5,000 epochs.

In Section 4.3, we made a preliminary analysis to determine the number of epochs that each network will use depending on the source of information used. This analysis was made with ASCAD, Zaid and NoConv1 networks because it's the most known network used in state-of-the-art of side channel attacks with deep learning techniques. We selected these networks in order to make our experiments and several networks and demonstrated that our contribution covers several deep learning techniques. As mentioned previously, we cannot use an early stopping method because some networks can take a certain number of epochs before they converge [2]. We have seen that depending on the source of information and the amount of data available, the number of epochs can vary. In all cases, using power as the source of information permitted the three neural networks to converge more quickly than the EM source. However, the number of epochs to converge until this mean rank depends on the network. The most important part of this analysis is the following two tables that summarize all hyperparameters information used in the following chapters. Table 4.3 summarises how many epochs have been defined for the training of ASCAD, Zaid, and NoConv1 networks according to a source of information with 100% of the database. Table 4.4 summarises how many epochs have been needed for the training of ASCAD, Zaid, and NoConv1 networks according to a source of information with 12% of the database. These epochs will be used in the following chapters 5, 6, 7. Table 4.2 summarizes all the other hyperparameters that we have used whatever the source of information for all networks, these hyperparameters are used in Chapters 5, 6, and 7.

Source of information	**ASCAD**	**NoConv1**	**Zaid**
EM2	200	5,000	5,000
EM3	200	3,000	5,000
EM4	200	3,000	3,000
F0EM	200	2,000	1,000
F1EM	200	2,000	1,000
F2EM	200	3,000	1,000
F4EM	200	2,000	1,000
F0PW	200	2,000	1,000
F1PW	200	2,000	1,000
F2PW	200	2,000	1,000
F4PW	200	2,000	1,000

Table 4.3 – Number of epochs defined for 100% database

2. for example the case of NoConv1 network with 12% of the dataset on F2EM

Source of information	**ASCAD**	**NoConv1**	**Zaid**
EM2	200	3,000	5,000
EM3	200	5,000	5,000
EM4	200	3,000	1,000
F0EM	200	3,000	1,000
F1EM	200	3,000	1,000
F2EM	200	5,000	1,000
F4EM	200	3,000	1,000
F0PW	200	2,000	1,000
F1PW	200	2,000	1,000
F2PW	200	2,000	1,000
F4PW	200	3,000	1,000

Table 4.4 – Number of epochs defined for 12% database

Chapter 5

Combining Sources of Side-channel Information

In most of the Side-Channel Attacks (SCAs), only a single source of the leakage is considered to attack a device. In the particular case of EM attacks, this source comes from a probe that has a supposed optimal position on the target. Few research papers investigate the combination of multiple leakage sources from different probe positions. In this thesis, the term "Multi-channel attacks" refers to an attack that combines several leakage sources from different probe positions and different probe types. In the following section, we present all related works about "Multi-channel attacks". These works are based on classical side channels and on Deep Learning (DL) side channel attacks.

5.1 Related works & Motivations

Standaert *et al.* [58] conduct several investigations to determine the impact of using multiple channels in SCA. These investigations allow answering three important questions.

1. Do electromagnetic leakages lead to more powerful attacks than power leakages [1]?
2. Does the combination of electromagnetic leakage and power consumption lead to a more powerful attack?
3. What is the effectiveness of data dimensionality reduction techniques for constructing subspace-based template attacks using multiple channels?

The authors quantify the amount of information leaked by these two side channels using two dimensionality reduction techniques namely Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA). An experimental comparison between both reduction methods shows that LDA leads to better information extraction than PCA. However, it is not trivial to chose the best dimensionality reduction technique because it does not take into account the efficiency of a model during the attack step. Their results show that the ElectroMagnetic (emission) (EM) channel is significantly more informative than the power channel when the

1. This point of interest is specific to the measurement setup used during the experimentation. The observations and conclusions made can be different for another measurement setup.

measurement probe is very close to the cryptographic chip. They also show that the combination of both channels (EM + power) leads to a more powerful attack compared to using only EM or power.

Agrawal *et al.* [2] proposes a formal adversarial model for multi-channel analysis using the power and multiple EM channels. They investigate several questions specific to the multi-channel attack, e.g. which side-channel signals should be collected? How should information from various channels be combined? These questions refer to the problem of signal detection in their paper. They present a formal adversarial model to overcome the problem of signal detection because it finds the two best side channels from a set of possibilities that maximize the attack efficiency. They provide experimental evidence that template attacks with EM + power outperform template attacks with EM or power. They make one strong assumption about the knowledge of the adversary. They take into account that the attacker has two crucial pieces of knowledge about the characteristics of the target device, the attacker knows the clock cycle and the noise distribution. Thus, he can perform attacks similar to template attacks on the device [2].

To avoid these pieces of knowledge about the characteristics of the device, they include other experiments with Differential Power Analysis (DPA), because DPA is simple and can be immediately applied to an unknown device. DPA does not require performing the attack on a similar device (such as A template attack). They experiment DPA with multi-channel by targeting different bit values. DPA with multi-channel is similar to a DPA with one channel because instead of taking a collection of traces from one channel, they take a collection of traces from several channels. However, for the multi-channel DPA to be effective, they select channels that have similar leakage characteristics. All experiments show that the combination of power and EM reduce the number of traces required to discover the correct key compared to using only power or EM.

Yang *et al.* [71] introduces another way to use multi-channel attacks because they do attacks but by merging in different ways multi-channel measurement. In their paper, they present different methods to "fusion" leakage information from multiple channels. They classify them into three groups depending on the level of the fusion: *data-level*, *feature-level*, and *decision-level* fusion attack. They construct six multi-channel fusion attacks and verify their performance in different scenarios. The *data-level* fusion attacks mean merging multi-channel measurements into one new leakage. The *feature-level* fusion attacks mean merging multi-channel measurements after applying dimension reduction techniques [3]. The *decision-level* fusion attacks mean combining the results of all mono-channel attacks. Their results show the efficiency of different

2. In a real-world scenario, the attacker often doesn't have knowledge about the characteristics of the device and can be difficult to mount.

3. The main dimension reduction techniques apply are: PCA and LDA

multi-channel attacks with the different groups of fusion against MCU [4] and FPGA with unprotected Advanced Encryption Standard (AES)-128 implementation. They show that data-level and feature-level fusion attacks are more efficient when the leakage position exists in different channels, whereas the feature-level and decision-level fusion attacks are better when we have few details about the cryptographic algorithm implementation.

The work of Standaert *et al.* [58] and Agrawal *et al.* [2] demonstrates the interest of multi-channel attacks with the combination of EM and power compared to the mono-channel attacks. These works were done on two different measurement setups but led to the same conclusion. In these two works, classical side-channel attacks with some dimensionality reduction techniques were used. However, it can be a difficult task to choose the best dimensionality reduction technique because it does not take into account the efficiency of a model during the attack step. We are motivated to use the DL technique with multi-channel attacks to be sure that the relevant feature will be useful for the attack step.

Hettwer *et al.* [26] proposes a new DL-based side channel attack that combines the leakage from different sources using a deep learning network to break protected hardware. The hardware to break is a modern System-on-Chip with 16nm fabrication technology running an AES with masking countermeasure. Authors present two techniques for fusion: *early fusion* and *late fusion*. The *early fusion* consists in combining the different channel inputs within the first layer of the Deep Neural Network (DNN). On the opposite, when performing a *late fusion*, the attacker extracts features from the input data of the individual channels and then combines and processes them in the further layers of the DNN. Experiments presented by the authors show that the *late fusion* always outperforms *early fusion*. During the experimental part, authors compare their models with several template attacks enhanced with PCA. Their results show that Template Attack (TA)-based attacks outperform the template attacks on decoupling capacitor measurements. Hettwer *et al.* [26] demonstrates the interest and advantage to use DL-based side channel attacks for multi-channel leakage. However, in their paper, they are limited to only two channels of power consumption. An interesting aspect that could be explored would be to add one more channel and see the efficiency of the attacks compared to only two channels. Also, in a real-world scenario, acquiring traces coming from power channels can be a difficult task because it requires that the attacker have direct access to the target device. To relax this assumption, it could be interesting to investigate multi-channel attacks using only EM leakages.

These works provide pieces of evidence that the consideration of multi-channels can be beneficial for side-channel analysis. Given the advantages of machine learning and deep learning techniques in SCA (see chapter 3), our approach consists in deriving multi-channel deep learning

4. In more details, it is an 8051 microcontroller

techniques. This allows us to make use of the information provided from multi-channels as well as the advantages given by machine/deep learning techniques. In this work, we are extending the approach of multi-channels to different EM locations, all approaches we used are described in the following section.

5.2 Approaches

The idea proposes in this Chapter is to combine different sources of information to improve the attack efficiency. This approach is enabled as soon as an attacker can measure several sources of information on a device at the same time. Indeed, the different sources of information must correspond to the same data used during the encryption process (same plaintext, key, and mask).

To smooth the description of the different training strategies let us introduce some notations:

— the **original attack dataset** contains the traces of one channel from which the attacker wants to extract a secret key;

— the **original training dataset** contains traces of one channel;

— the **combined attack dataset** is the concatenation of two (or three) original attack datasets from which the attacker wants to extract a secret key, and all original datasets come from different source of information;

— the **combined training dataset** is the concatenation of two (or three) original training datasets, and all original datasets come from different source of information.

The three training strategies are described in Fig 5.1, but some precision is given in the two following paragraphs.

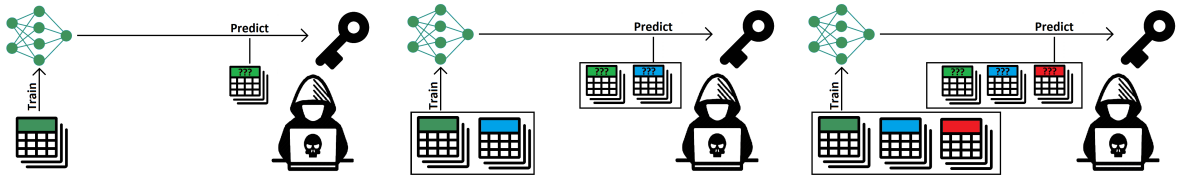
With these datasets in mind, the two attacks can be simply described in a very compact form.

Definition 1 (Regular attack) *The regular attacks is defined when:*

1. *Train a profiled side-channel attack on the original training dataset.*
2. *Attack the original attack dataset to recover the key using a profiled attack.*

Definition 2 (Multichannel attack) *The multichannel attack is defined when:*

1. *Train a profiled side-channel attack on the combined training dataset.*
2. *Attack the combined attack dataset to recover the key using a profiled attack.*



(a) Regular approach (b) Combine two source approach (c) Combine three source approach

Figure 5.1 – Training strategies for multichannel

5.3 Implementation details

We use the database called "DATABASE_AVR". This database is composed of two sets: the profiling set and the attack set. The profiling set contains three datasets (one for each probe), and each dataset is composed of 200,000 raw traces with a frame of 30,000 samples. The attack set contains three datasets, and each dataset is composed of 60,000 raw traces with a frame of 30,000 samples. Table 5.1 describes the data used in the profiling set and attack set. In both the profiling and the attack sets, we use the same randomness between each dataset so that the data is common across datasets (that is, trace 47 of EM2 has the same data as trace 47 of EM3). We cannot exploit directly the raw traces because the frame of the data is too large (with a frame equal to 30,000 samples). Thus, we reduce this frame by selecting the windows with the highest Signal-to-Noise Ratio (SNR) (that means we select the windows where the leakage is the most obvious). Table 5.2 describes the size of the original frame and the frame selected for our experiments. We select only 100,000 raw traces to have the same number of traces as the database "DATABASE_STM32" and thus allow the training of our network with the same amount of data from both databases. We exploit 100,000 raw traces with a frame of 700 samples for the profiling set and we exploit 60,000 raw traces with a frame of 700 samples for the attack set. We use 80% of the profiling set for the training of the DNN, and the last 20% for the validation of the DNN. Finally, we use 100% of the attack set for testing the DNN.

Data	**Profiling set**	**Attack set**
plaintext	random	random
key	random	changed every 2,000 traces

Table 5.1 – Composition of data into database "DATABASE_AVR"

Dataset	Original frame	Selected
Profiling set	30,000 samples	700 samples
Attack set	30,000 samples	700 samples

Table 5.2 – Composition of frame into database "DATABASE_AVR"

5.4 Contribution

5.4.1 Multi-channel with EM2

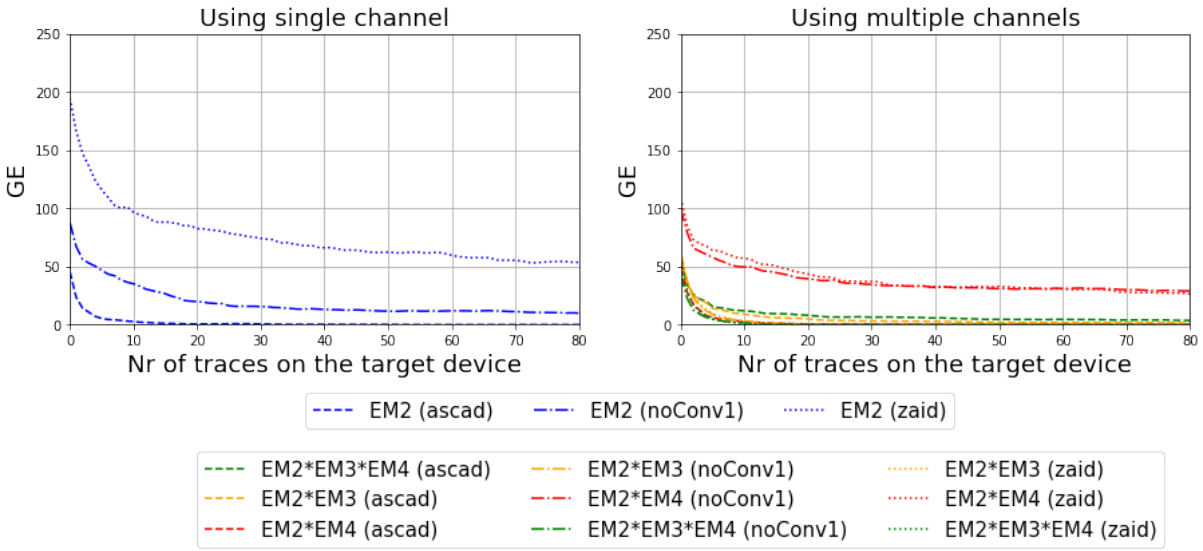


Figure 5.2 – Guessing entropy when targeting EM2

Figure 6.5 shows the Guessing Entropy (GE) when targeting EM2, which is the channel with the highest SNR value. On the left side, we observe the performances of the ASCAD network, Zaid network, and NoConv1 network when using a single channel called EM2 (described in chapter 4). We observe that the ASCAD network quickly converges towards zero, and reaches a mean rank equal to zero with 25 traces. The NoConv1 and Zaid networks require more traces than the ASCAD network. For the NoConv1 network, it reaches a mean rank equal to 6 with 200 traces. For the Zaid network, it reaches a mean rank equal to 35 with 200 traces. On the right side, we observe the performances of the ASCAD network, Zaid network, and NoConv1 network when using multiple channels. We observe that the ASCAD network with the channels "EM2*EM3*EM4" reaches a mean rank equal to 3 with 200 traces, with the channel "EM2*EM3", it reaches a mean rank equal to 0 with 200 traces, and with the channel "EM2*EM4" it converges towards zero with 50 traces. We observe that the NoConv1 network with the channels

"EM2*EM3" and the channel "EM2*EM3*EM4" converges towards zero with 30 traces, and with the channel "EM2*EM4" it reaches a mean rank equal to 25 with 200 traces. We observe that the Zaid network with the channels "EM2*EM3" and the channel "EM2*EM3*EM4" converges towards zero with 25 traces, and with the channel "EM2*EM4" it reaches a mean rank equal to 18 with 200 traces.

We recall that we are in the context of multi-channel adding only noisier sources (EM2*EM3, EM2*EM4, EM2*EM3*EM4) to EM2. We notice that all multi-channels (except for the combination of EM2*EM4 that give the same performance as the single-channel) decrease the performance of the ASCAD network. This degradation of performance can be justified by the fact that only noise is added to the network. Concerning the NoConv1 network, we notice that adding a channel with a certain level of noise but also a certain level of leakage can improve the performance of the network because we need fewer traces to converge until zero. Indeed, only 30 traces are required compared to more than 200 traces when using a single channel. However, we also notice that adding a channel with too much noise can decrease the performance of the network. Concerning the Zaid network, we notice the same thing as the NoConv1 network. We observe that adding a channel with a certain level of noise can improve the efficiency of the attack, however, a channel with too much noise can also decrease the efficiency of the attack.

5.4.2 Multi-channel with EM3

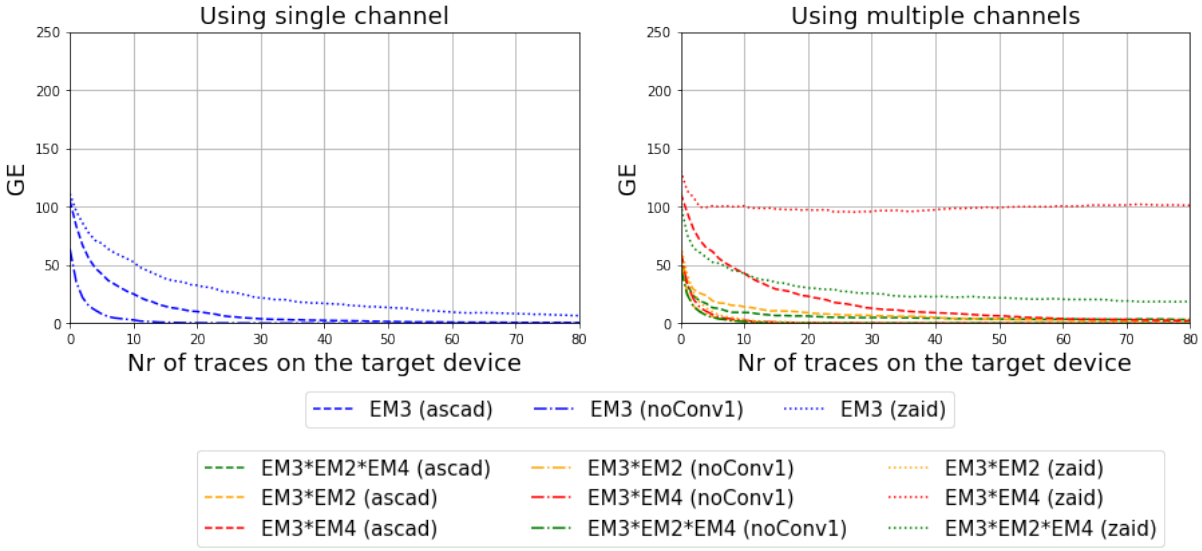


Figure 5.3 – Guessing entropy when targeting EM3

Figure 6.6 shows the GE when targeting EM3, which is the second channel with the highest SNR value. On the left side, we observe the performances of the three studied networks when

using the single channel EM3. We observe that the ASCAD network reaches a mean rank equal to zero with 150 traces and the Zaid network reaches a mean rank equal to 0 with 200 traces. However, the NoConv1 network reaches a mean rank equal to zero with 40 traces.

On the right side, we plot the performances of the ASCAD network, Zaid network, and NoConv1 network when using multiple channels. We observe that the ASCAD network using the combination of channels "EM3*EM2*EM4" or channels "EM3*EM2" reach a mean rank equal to 2 with 200 traces, and with the channel "EM3*EM4" it converges towards zero with 175 traces. We observe that the NoConv1 network with the channels "EM3*EM2*EM4" and channels "EM3*EM2" reaches a mean rank equal to zero with 25 traces, and with the channel "EM3*EM4" it converges towards zero with 75 traces. Finally, we observe that the Zaid network with the channels "EM3*EM2" reaches a mean rank equal to zero with 25 traces, with the channel "EM3*EM2*EM4" the mean rank equal to 25 with 200 traces, and with the channel "EM3*EM4" the mean rank equal to 100 with 200 traces.

We recall that we are in the context of multi-channel adding clearer sources (EM3*EM2), noisier sources (EM3*EM4), and both (EM3*EM2*EM4) to EM3. For the ASCAD network, we notice that any multi-channel approach, no matter if it adds noise or a clearer source, decreases the performance of the network. At least worse, the mean rank increases from 150 to 175 for the ASCAD network. Concerning the NoConv1 network, we notice that a multi-channel approach adding a clearer source than the original leads to better performances since it converges towards zero with 25 traces compared to 30 traces. However, if in the multi-channel attack, we have a source that is noisier than the original, this leads to a decrease of the performance of the network because it then requires 75 traces to converge toward a mean rank of 0. Concerning the Zaid network, we notice that only multi-channel with the clearer source added to the original allows increasing the performance of the network because the number of traces required to converge towards zero is decreased from 200 to 25. Otherwise, all other multi-channel attacks decrease the performance of the Zaid network, in one case the mean rank increases from 1 to 25 with 200 traces, and in other cases, the mean rank increases from 1 to 100 with 200 traces.

5.4.3 Multi-channel with EM4

Figure 6.7 shows the GE when targeting EM4, which is the channel with the lowest SNR value. On the left side, we observe the performances of the ASCAD network, Zaid network, and NoConv1 network when using the single channel EM4. We observe that both the ASCAD network and the Zaid network reach a mean rank equal to zero with 200 traces while the NoConv1 network reaches a mean rank equal to 0 using the same amount of data.

We observe that the ASCAD network with the channels "EM4*EM2*EM3" reaches a mean rank equal to 0 with 200 traces. However the ASCAD network with the channels "EM4*EM2"

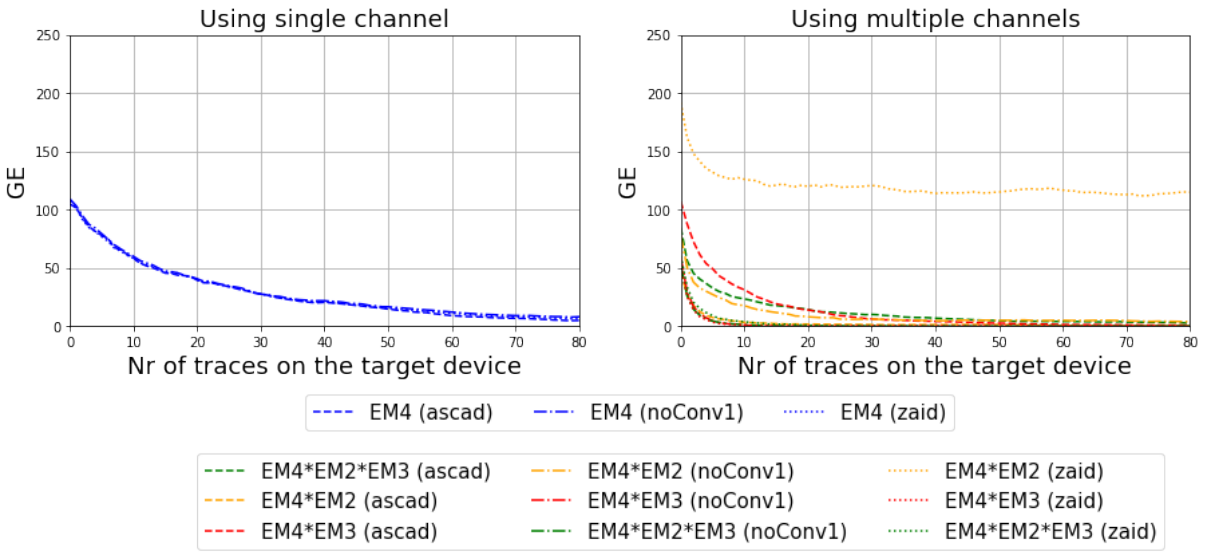


Figure 5.4 – Guessing entropy when targeting EM4

converges towards zero with as few as 100 traces. Using channels "EM4*EM3", it converges towards zero with 150 traces. Concerning the NoConv1 network, we see that using the channels "EM4*EM3" or the channels "EM4*EM2*EM3" leads to a quick convergence towards zero (only with 25 traces required). However, for the combination of channels "EM4*EM2" the mean rank is greater than 1 with 200 traces. We observe that the Zaid network with the channels "EM4*EM3" converges towards zero with 25 traces, and with the channels "EM4*EM2*EM3" it converges towards zero with 50 traces. As for the ASCAD and NoConv1 networks, for the channels "EM4*EM2" the mean rank is equal to 120 with 200 traces.

We recall that we are in the context of multi-channel adding only clearer sources (EM4*EM3, EM4*EM2, EM4*EM2*EM3) to EM4. For the ASCAD network, we notice that multi-channel with only one clearer source added allows increasing the performance of the ASCAD network because the number of traces required to converge until zero is reduced from 200 to 100. Multi-channel with more than one clearer does not affect the performance of the ASCAD network, its mean rank just increased by one. Concerning the NoConv1 and Zaid networks, we notice two things. First, adding EM3 to EM4 increases the performance of both networks. Indeed, they converge towards mean rank zero with 25 traces compared to 200 traces if we use a single channel. However, if we add EM2 to EM4 the performances of these two networks decrease because NoConv1 reaches a mean rank equal to 0 with 200 traces and Zaid reaches a mean rank equal to 120 with 200 traces. The hypothesis may be that the variation between EM2 and EM4 prevents the networks (Zaid and NoConv1) from converging.

5.4.4 Conclusion

In this work, we investigate multi-channel attacks compared to regular attacks. We have seen different contexts of multi-channel:

— Multi-channel adding only noisy sources compared to an original single channel.

— Multi-channel adding noisy and clean sources compared to an original single channel.

— Multi-channel adding only clean sources compared to an original single channel.

We investigate these three contexts with three state-of-the-art networks namely ASCAD, Zaid and NoConv1 networks. Concerning the ASCAD network, we see that it is sensible to the noisy channel. Indeed adding other channels which have a lower SNR value compared to the original channel only decreases the performance of the network. However, adding other channels which have a higher SNR value increases the performance of the ASCAD network. For the NoConv1 network and Zaid network, we observe that they are sensitive to the variations between the channels. Indeed, we observe that adding other channels that are too different compared to the original channel (regarding SNR) decreases their performances. This difference may be due to a sharper or noisier source. However, adding other channels that have an SNR close to the original channel can increase the performance of these networks.

This work led us to a second idea. Since the weights of the networks are randomly initialized, wouldn't it be more interesting to take advantage of weights already adjusted with another data set? Can this initialization improve the convergence of the neural network and its performance? This is the question we study in chapter 6.

Chapter 6

Train or Adapt a Deeply Learned Profile?

The work in the previous chapter has led to a question on the way other sources of information may be used in Side-Channel Attack (SCA). We ask ourselves if, instead of combining several sources of information, we can take the profit of other sources by using them to initialize the weights of a network in a context where the number of traces in the profiling set is limited. We assume that even if the sources of information are different from each other, with a certain level of variation between them, this can be beneficial to the neural networks. The term "beneficial" means that the network will not start from a random weight initialization, but from weights that are adjusted during the training on another source of information, thus we hope that the network already acquired some knowledge, understood some part of the problem, and will need fewer traces to train. We investigate Transfer learning in this situation and see how it can be beneficial for SCA. As mentioned in chapter 2, transfer learning is one of the methods used for weight initialization based on another dataset. In most of the papers about Deep Learning (DL) for SCA, the training of a Deep Neural Network (DNN) is directly performed on a device similar to the target one. There is no work that takes into account some other source of information that an attacker may have access to. In this thesis, we introduce a new attack called "transferred attack". The term "transferred attack" refers to an attack consisting in using another source of information to initialize the weight of a network. Then, once the network is pre-trained, its weights are adjusted using the limited profiling set from the target device. In the following section, we present all related works about the portability issue of profiled attacks and "transfer learning" techniques.

6.1 Related works & Motivations

Bhasin *et al.* [5] treats the portability issue that can occur in a real-world scenario for SCA. They evaluate the influence of portability on the efficiency of a machine learning-based attack. They analyze portability by using the normalized inter-class variance metric to characterize the difference in measurements between different devices/keys. They make an analysis and evaluate the performances of several Machine Learning (ML) techniques with 4 different scenarios. These

4 scenarios are:

1. same device and same key,
2. same device and different keys,
3. different devices and same key,
4. different devices and different keys.

The first scenario, where the attackers use the same device and the same key for the profiling and attack phases, is the common scenario studied in the SCA literature but this is not a realistic one. As mentioned in several works, all ML approaches are very good. The best model, an MultiLayers Perceptron (MLP) with 600 features trained with 10k traces, requires only 10 attack traces to discover the key. In the second considered scenario, attackers use the same device in the profiling phase and attack phase, but the keys between these two phases are different. Authors observe that this scenario is more difficult for ML techniques. The performance of some ML techniques that use 50 features (such as Naive Bayes, Random Forest, and MLP with 50 features) is decreased compared to the first scenario. However, some ML techniques keep the same performances as the first scenario such as MLP with 600 features, and Convolutional Neural Network (CNN). In the third scenario, attackers use the same key during the profiling phase and the attack phase but on distinct devices. This scenario is more realistic than the former, but it is also more difficult for an ML technique. All ML techniques using only 50 features require more than 100 attack traces to reach a mean rank equal to zero. The MLP with 600 features and CNN are the two best models because they require less than 20 attack traces to successfully discover the key. The last scenario considers that both the device and the key used by the attacker during the attack phase are different from the one used for profiling. This scenario can be considered the most realistic one and raises the portability issue of SCA. For the random forest, a mean rank equal to less than 90 is reached with 100 attack traces. For the Naive Bayes and MLP with 150 features, a mean rank less than 15 is obtained using 100 attack traces. For both the MLP with 600 features and the CNN, using 60 attack traces, the attacker obtains a mean rank equal to zero.

Wang *et al.* [64] deal with the problem of inter-chip variation and its impact on the performance of DNN to correctly predict the correct key. The authors propose a solution by training a DNN on multiple chips instead of one. This solution can overcome the problem of inter-chip variation and improve the efficiency of DL-based attacks. Their first experiment was performed on different chips while their second experience was done on two different devices namely the XMEGA1 and the XMEGA2. The results of their first experience show that training a DNN on nine chips rather than just one can increase its probability to recover the correct key. The probability to recover the key from a single trace increases from 40% to 86%. Their results also

show that training a DNN on two different types of boards rather than just one board can also increase its probability of successfully discovering the correct key. In that case, the probability to recover the key from a single trace indeed increases from 13% to 55%.

Thapar *et al.* [59] publishes an independent work but on a similar topic as our research. In their works, they also propose a DL side-channel attack based on the Transfer Learning (TL) to address the issue of insufficient data to train a model. They use the same approach as us (presented in chapter 4) and having the same positive outcomes on the use of TL when few training traces are available, but with a different measurement setup and a different architecture of the neural network. The approach they used [1] is the following. First, the attacker acquires a large number of profiled power traces from any device of his choice in order to make the first training and create a *base* model. Secondly, using a clone of the target device, the attacker acquires a limited set of profiled power traces in order to fine-tune the *base* model thanks to a transfer learning method. Third, the attacker uses the fine-tuned model to attack the target device. The architecture used by Thapar *et al.* [59] is described in Figure 6.1, the green colour represents the layer that is updated during the fine-tuning step, and the white colour represents the layer which is frozen (the weight of the layer is not updated) during the fine-tuning step. The authors make several experiments with different FPGA families. First, they investigate the efficiency of transfer learning with unprotected Advanced Encryption Standard (AES) implementation. They show that the transfer learning strategy can successfully find the correct key rank with 500 attack traces on average. They show that TL permits to attack a new device from the knowledge of another device even if the number of traces from the new device is limited. Secondly, they investigate the efficiency of transfer learning with a first-order masked AES implementation. They show that the transfer learning strategy successfully finds the correct key rank with 200 attack traces on average.

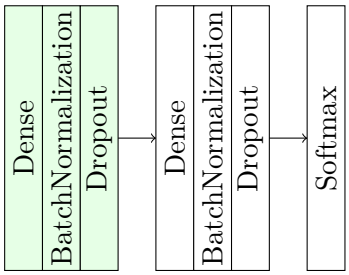


Figure 6.1 – TranSCA network

These works provide pieces of evidence that DL has a certain capacity of portability even if we have a certain variation between each source of information. Thapar *et al.* [59] have shown

1. and which we also used

that TL can be used in SCA, and in the context where we have a limited profiling set. However, in their studies, they just used TL between different FPGA families with only power as the source of information. We proposed to use TL with DNN in several other scenarios. Thus we investigate "transferred attack" between different channels, different side-channels, and also different devices but from the STM32 family.

6.2 Approaches

The idea proposes in this Chapter is to use transfer learning to improve the attack efficiency in the context of insufficient training data. The goal is to use transfer learning in the initialization of the weights of the network from a network trained on another source of information rather than using random initialization. This approach is of interest as soon as an attacker has only a few traces to mount a profiled attack. The goal is to determine if weight initialization from different sources can help to improve the attack efficiency. To smooth the description of the different training strategies let us introduce some notation:

— the **original attack dataset A** contains the traces from which the attacker wants to extract a secret key (thus from source A);

— the **original training dataset A** contains a small amount of traces from source A;

— the **original training dataset B** contains traces from source B [2].

The three training strategies are described in Fig 6.2, but we provide here two compact definitions of the two newly introduced attacks.

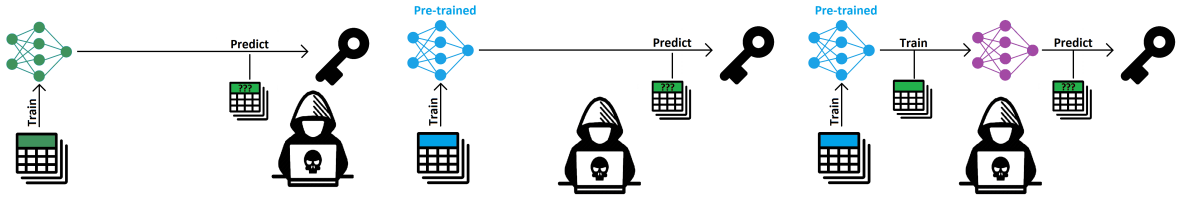
Definition 3 (Pretrained attack) *The pretrained attack is defined when:*

1. *Train a profiled side-channel attack on the original training dataset B.*
2. *Attack the original attack dataset A to recover the key using a profiled attack.*

Definition 4 (Transferred attack) *The transferred attack is defined when:*

1. *Train a profiled side-channel attack on the original training dataset B.*
2. *Fine-tune a profiled side-channel attack on the original training dataset A.*
3. *Attack the original attack dataset A to recover the key using a profiled attack.*

2. We assume that the number of traces from source B is higher than source A



(a) Regular attack (b) Pretrained attack (c) Transferred attack

Figure 6.2 – Training strategies for transfer learning

6.3 Implementation details

We include another database (in addition to the previous database called "DATABASE_AVR") namely "DATABASE_STM32". This last one is also composed of two sets: a profiling set and an attack set. The profiling set contains eight datasets (each device has a dataset for power and a dataset for ElectroMagnetic (emission) (EM)), and each dataset contains 100,000 raw traces with a frame of 5,000 samples. In the attack set, each of the eight datasets contains 32,000 raw traces with a frame of 5,000 samples. The inputs (plaintext and key) used in the profiling set are variable and uniformly distributed but remain the same over datasets. The data used in the attack set are fixed [3] and remain the same between each dataset. Table 6.1 describes the data used in the profiling and attack sets. As for the first database, we cannot directly exploit the frame since it is too large. Thus, we reduced it from 5,000 to 700 points (with the same technique as for the first database). Table 6.2 gives the sizes of the originally selected frames. Thus, we exploited 100,000 raw traces with a frame of 700 samples for the profiling set and 32,000 raw traces with a frame of 700 samples for the attack set. For the pre-trained approach, we used 80% of the profiling set for the training of the DNN, and the last 20% for the validation of the DNN (with both databases) to simulate the fact that we already had a pre-trained network. For the transfer learning and regular approach, we used 10% of the profiling set for the transfer learning of the DNN, 2.5% of the profiling set for the validation of the DNN to simulate the fact that the attacker only gets a small profiling dataset. We used 100% of the attack set for the test of the DNN (for both databases).

Data	**Profiling set**	**Attack set**
plaintext	random	random
key	random	changed every 1,000 traces

Table 6.1 – Composition of data into database "DATABASE_STM32"

3. The key has been changed every 1000 traces

Dataset	Original frame	Selected
Profiling set	5,000 samples	700 samples
Attack set	5,000 samples	700 samples

Table 6.2 – Composition of frame into database "DATABASE_STM32"

6.4 Contribution

6.4.1 Details on Transfer Learning

For the transferred attack, we used transfer learning on ASCAD, Zaid, and NoConv1 network architectures. Different strategies using transfer learning are possible.

For example, an attacker could have only trained dense layers (thus freezing convolutional layers) or could have reset some specific layers while keeping previous parameters for other layers.

Those strategies have three advantages:

— decrease the amount of data needed to train the neural network.

— faster convergence because the initialized weights from the first training may be better suited than a random initialization.

— reduce the number of parameters to update in the neural network, which may increase the training speed.

But those strategies are one drawback:

— potentially sub optimal solution however if frozen layers already have good parameters it could be ok.

We applied two strategies during some initial experiments: re-training the complete network during the second step and freezing the convolutional layers (if any). The choice of freezing convolutional layers is based on the assumption that feature extraction should be similar from one context to another (only decision changes). Since both gave similar results on our first runs, we decided to focus on retraining the full network, since then we do not make any hypothesis (that could end up being false in some cases). Investigating deeply different techniques is clearly an interesting extension of this work.

Figure 6.3, 6.4a, and 6.4b describe the layers that will be fine-tuned during the transfer learning. We used the previously introduced databases "DATABASE_AVR" and "DATABASE_STM32".

We used a similar amount of data across experiments. The choices have been made so that i) most of the direct approaches lead to working attacks ii) the data requirement drops is clearly

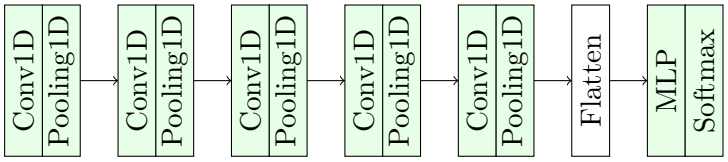


Figure 6.3 – Transfer learning on ASCAD network

(a) Zaid network (b) NoConv1 network

Figure 6.4 – Transfer learning on Zaid and NoConv1

visible when performing transfer learning and iii) the pre-trained models may be open-sourced (that is coming from another party that may have higher available resources). The number of traces used is hence:

— 100,000 traces for the first training (pre-trained attacks).

— 12,500 traces for regular attacks, and transferred attacks.

6.4.2 Transferring between EM probe positions and types

First, we compare the effectiveness of applying a "transferred attack" targeting EM measurements and having available pre-trained models on different channels. For these experiments, we used the "DATABASE_AVR" database presented in Chapter 4 with the three network architectures: ASCAD, Zaid and NoConv1.

Figure 6.5 shows the Guessing Entropy (GE) when targeting EM2, which is the channel with the highest Signal-to-Noise Ratio (SNR) value. On the left side, we see that all pre-trained models [4] and the pre-trained models of EM2 are converging quickly towards zero, with ASCAD and NoConv1 being slightly more effective. Pre-trained models, on EM3 and EM4, do not succeed to converge, even using a high amount of traces compared to the regular attack. On the right side, we see that using transfer learning permits the improvement of the efficiency of all pre-trained models with channels EM3 and EM4. We observe that the best performance between all attacks (regular attack, pre-trained attack, and transferred attack) is obtained by applying transfer learning on the NoConv1 network with EM4.

Next, Figure 6.6 shows the GE when targeting EM3, which has a medium SNR and is using the same probe type as EM4. We observe that the performances of the regular attacks with

4. the pre-trained models of EM2 is the regular attacks, so it only used 12,500 traces compared to other that used 100,000 traces

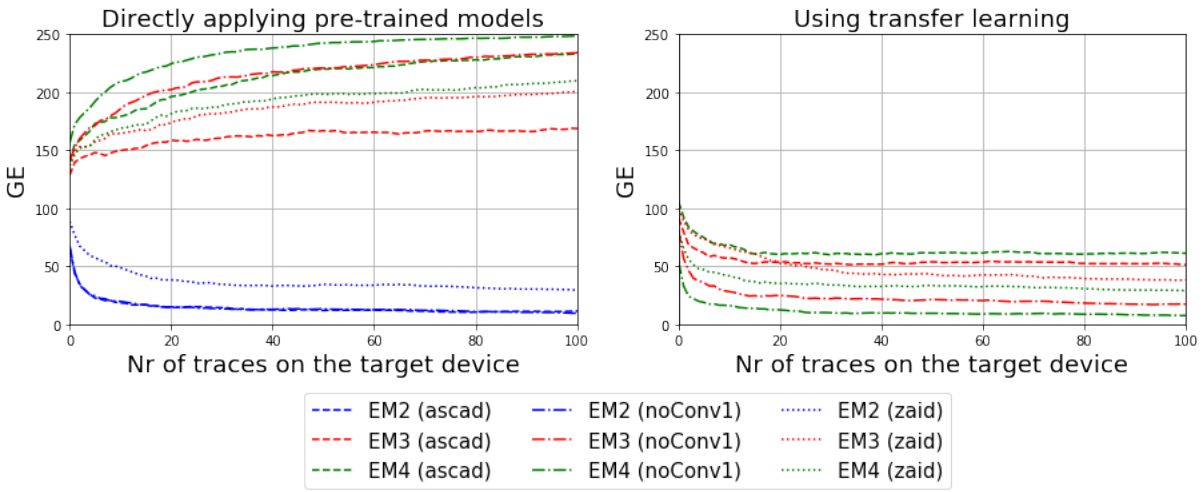


Figure 6.5 – Guessing entropy when targeting EM2

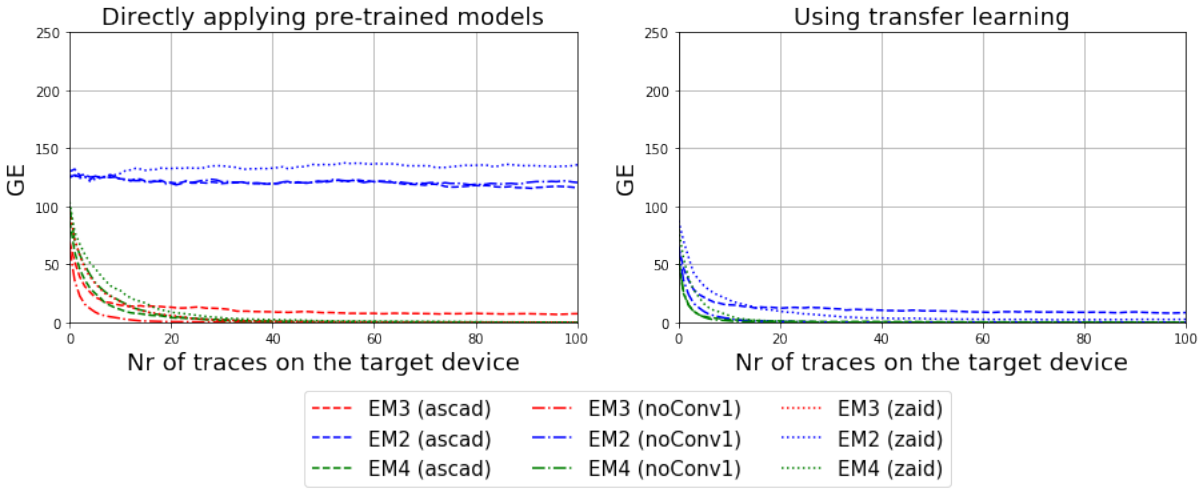


Figure 6.6 – Guessing entropy when targeting EM3

EM3 and the pre-trained attacks with EM4 are close. However, the regular attack is still more efficient than the pre-trained attack. Interestingly, a pre-trained model on EM4 is slightly more effective than using a pre-trained model on EM2 because we observe that EM4 converge to a mean rank equal to zero with 40 traces while EM2 does not converge at all. On the right, we see that using transfer learning permits to increase the performance of the pre-trained model with the channel EM2. We observe that the best performance between all attacks is obtained by applying transfer learning on the NoConv1 network with EM4.

In Figure 6.7 we show the GE targeting EM4 that has the lowest SNR from all three EM positions. Directly applying a pre-trained model from EM3 using NoConv1 works sufficiently well, followed by a pre-trained model on EM4 using NoConv1, and a Zaid network from EM4.

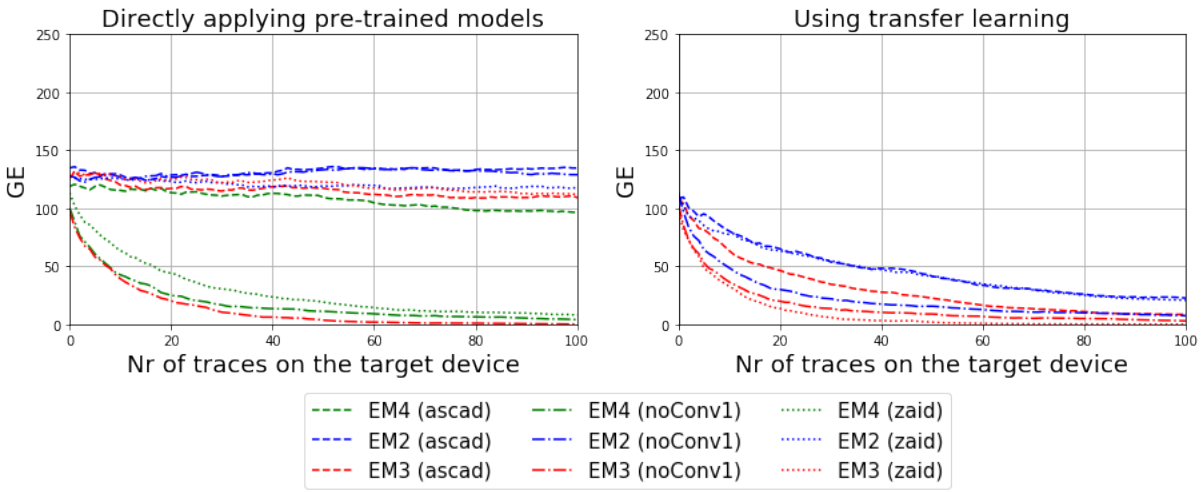


Figure 6.7 – Guessing entropy when targeting EM4

Interestingly, using directly the pre-trained model EM2, which is the channel containing the highest amount of information, does not perform better than less informative channels. We observe that using transfer learning improves the result for EM2+ASCAD, EM2+NoConv1, EM2+Zaid, EM3+Zaid, and EM3+ASCAD. We observe that the best performance between all attacks is obtained by applying transfer learning on the Zaid network with EM3.

6.4.3 Transferring between Power and EM

Secondly, we now compare the effectiveness of applying a "transferred attack" targeting EM measurements and having available pre-trained models on power consumption. For these experiments, we used the "DATABASE_STM32" database (refer to Chapter 4) and we considered the four previously introduced devices (namely, F0, F1, F2, and F4). We used ASCAD, Zaid and NoConv1 as neural network architectures.

In Figure 6.8 we plot the guessing entropy obtained when targeting device F0 with EM (F0em). On the left, we observe all pre-trained attacks and regular attacks. We observe that all pre-trained attacks do not converge within 1000 traces and only the regular attacks with NoConv1 and Zaid network converge. On the right, we observe all transferred attacks. We show that the efficiency of all pre-trained models (ASCAD, NoConv1, and Zaid with power) can be improved. We observe that the best performance between all attacks is obtained by applying transfer learning on the NoConv1 network with the power source.

Figure 6.9 presents the results when attacking F1 with an EM source. Similarly to previous experiments, when applying pre-trained networks directly with the power source, none of the networks seems to have a decreasing GE. But when applying pre-trained networks directly with EM sources, Zaid and NoConv1 reach a GE close to 0. However, when updating the pre-trained

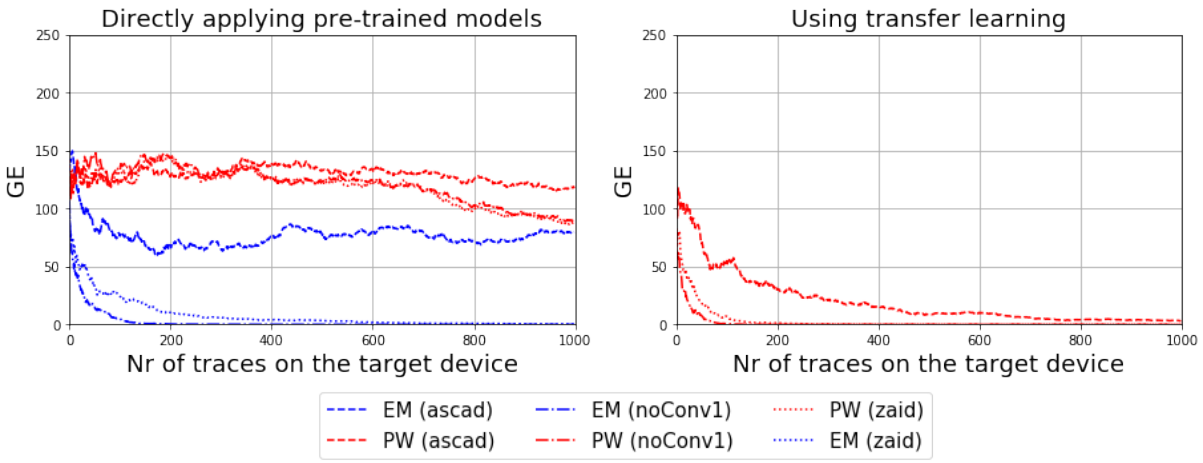


Figure 6.8 – Guessing entropy when targeting F0em

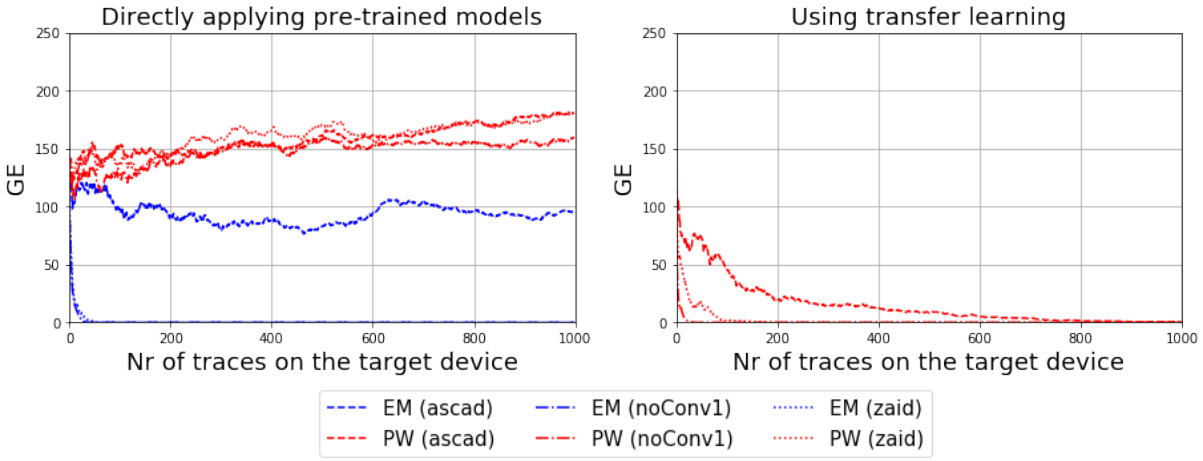


Figure 6.9 – Guessing entropy when targeting F1em

model with transfer learning, we observe that ASCAD, Zaid, and NoConv1 with the power channel can be improved and reach a mean rank close to 0. As previously, we observe that the best performance between all attacks is obtained by applying transfer learning on the NoConv1 network with the power source.

Figure 6.10 presents the results when attacking F2 with an EM source. We show that using a pre-trained model on power does not lead to convergence. We observe that only the Zaid network and the NoConv1 network converge with the EM channel. Interestingly, we see that in this scenario transfer learning is very effective on all three networks, while Zaid is the most effective one, reaching a GE of 0 with less than 600 traces. We observe that the best performance between all attacks is obtained by applying the transfer learning on the Zaid network with the power source.

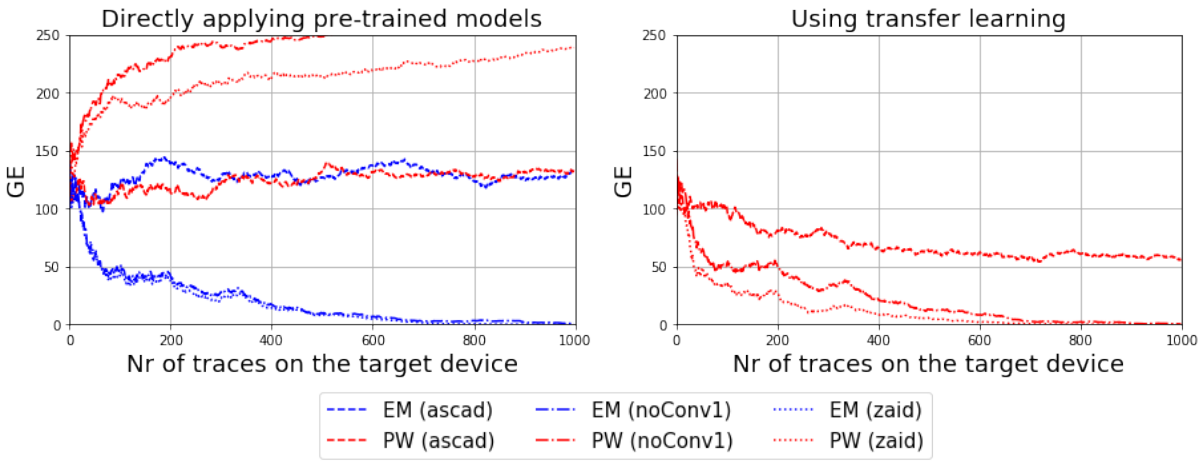


Figure 6.10 – Guessing entropy when targeting F2em

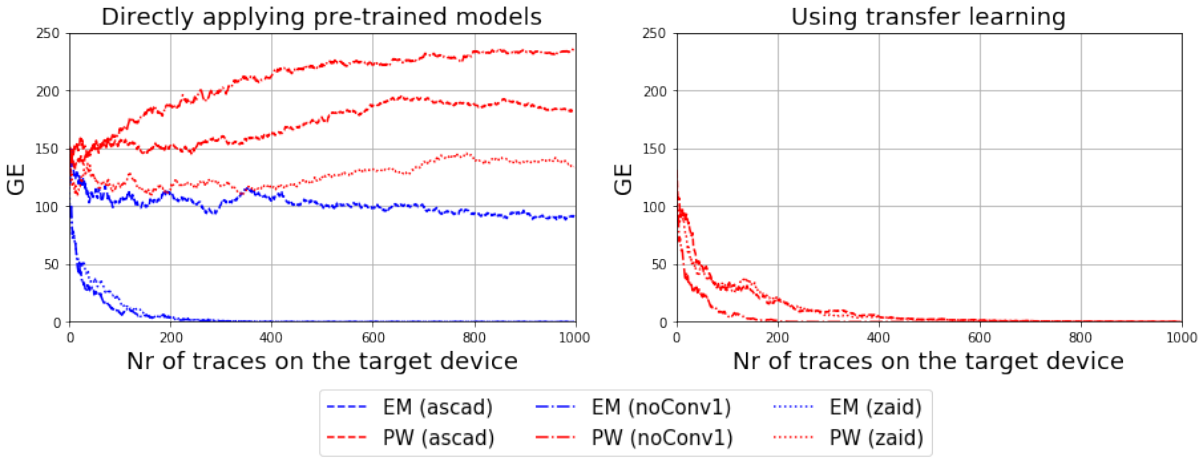


Figure 6.11 – Guessing entropy when targeting F4em

Figure 6.11 presents the results when attacking F4 with an EM source. When applying pre-trained networks directly with the power source, NoConv1 and Zaid are the only ones to obtain a mean rank equal to 0 with at least 250 traces. We can see that using transfer learning on all 3 neural networks pre-trained on power consumption permits us to obtain better performances than directly applying them. We observe that the best performance between all attacks is obtained by applying the transfer learning on the NoConv1 with the power source.

6.4.4 Transferring between different devices

Third, we investigate a "transferred attack" between different devices. For these experiments, we also used the database called "DATABASE_STM32" as the previous experiment with the three networks: ASCAD, Zaid, and NoConv1.

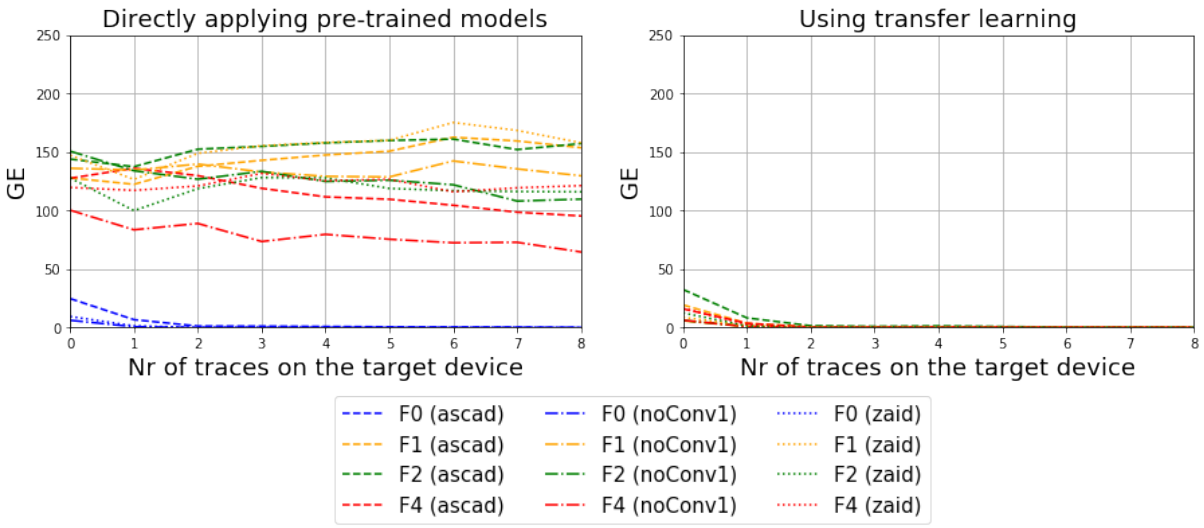


Figure 6.12 – Guessing entropy when targeting F0

Figure 6.12 presents the results when targeting F0 and pre-trained models are built from F0, F1, F2, and F4. We can see that when using directly pre-trained models, the model on F0 is the most effective, but not all pretrained models converge towards 0, and NoConv1 is the best among the three neural networks. When compared to transfer learning, all pre-trained models are improved. We observe that the best performance between all attacks is obtained by applying the transfer learning on the NoConv1 with the device F4.

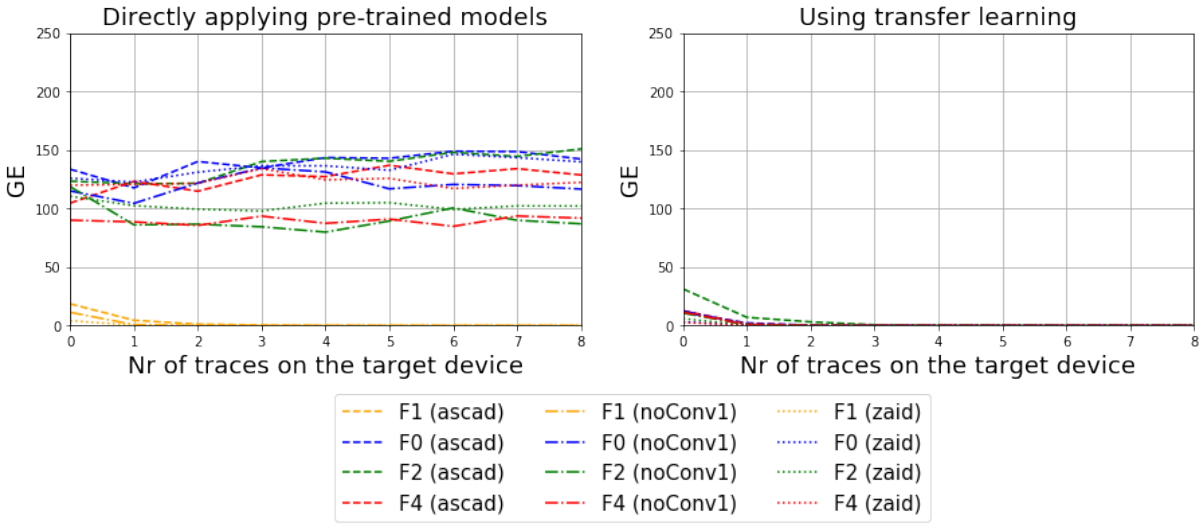


Figure 6.13 – Guessing entropy when targeting F1

Figure 6.13 presents the results when targeting F1 and pre-trained models are built from F1, F0, F2, and F4. The same conclusion can be seen in the figure above (when targeting F0).

We observe that transfer learning allows us to improve the efficiency of the pre-trained models. However, in this case, the best performance between all attacks is obtained by the regular attacks on the Zaid network with an F1 device.

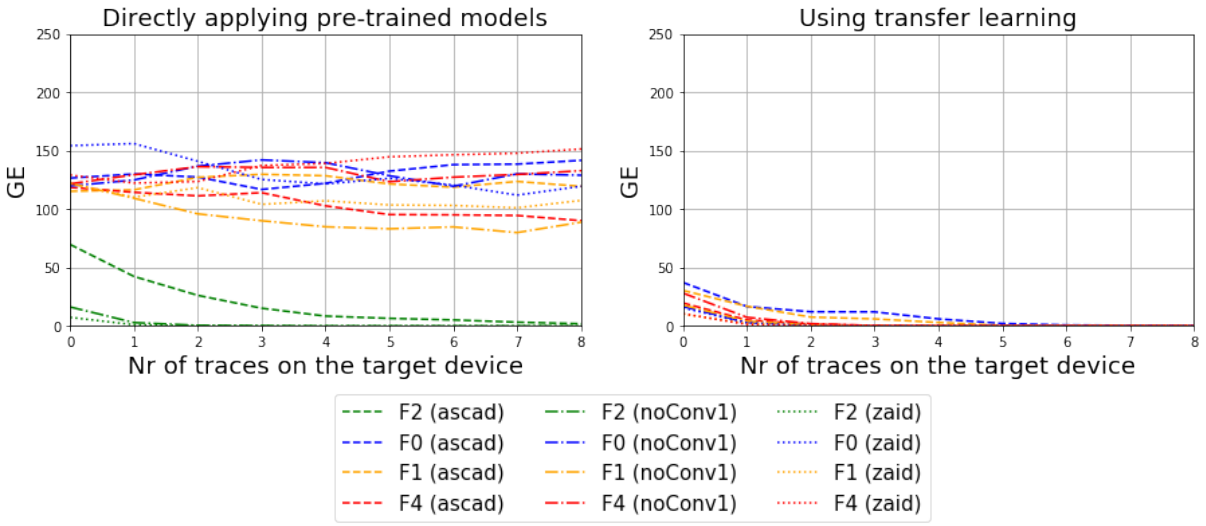


Figure 6.14 – Guessing entropy when targeting F2

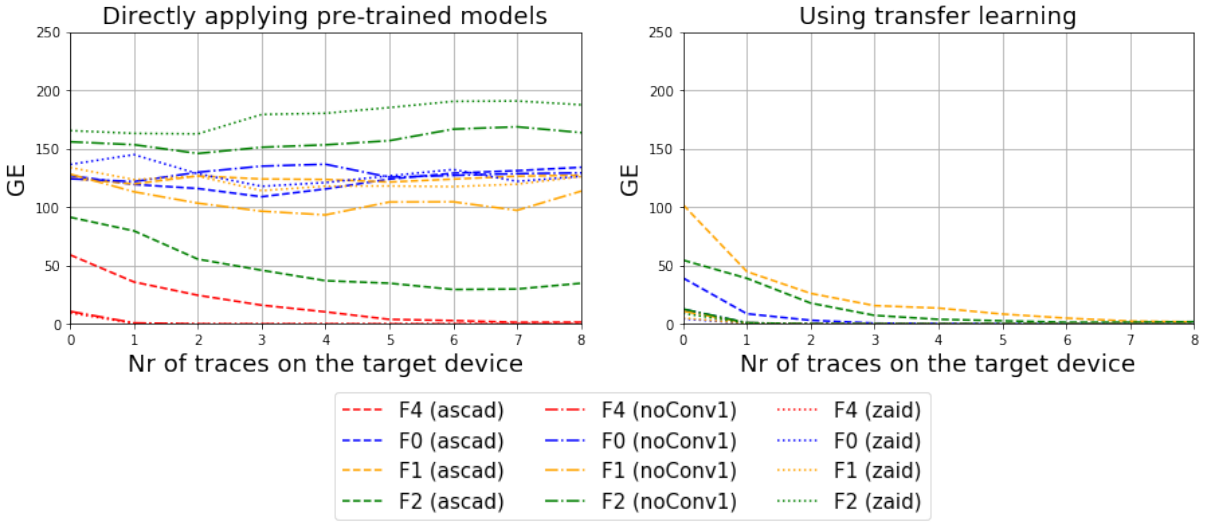


Figure 6.15 – Guessing entropy when targeting F4

Figure 6.14 presents the results when targeting F2 and pre-trained models are built from F2, F0, F1, and F4. Figure 6.15 presents the results when targeting F4 and pre-trained models are built from F4, F0, F1, and F2. As previously, only the regular attack converges toward a mean rank equal to zero. All pre-trained attacks not trained with the F2 device for Fig 6.14 and with

F4 for Fig 6.15 do not converge. The transfer learning allows for increasing the efficiency of all pre-trained attacks because all pre-trained attacks reach a mean rank equal to 0. For Fig 6.14 we observe that the best performance between all attacks is obtained by the regular attacks on the Zaid network with the F2 device. However, the performance of transferred attacks is close to the performance of the best regular attacks. For Fig 6.15 we observe that the best performance between all attacks is obtained by applying the transfer learning on Zaid with the F1 device.

6.4.5 Conclusion

In this work, we investigate transferred attacks compared to regular attacks with a limited profiling set. We have seen three different scenarios:

— Transferring between EM probe position and type

— Transferring between power and EM

— Transferring between different devices.

We compare the different attacks described in Chapter 4: regular attacks, pre-trained attacks, and transferred attacks. To show that our experiments can be done with different networks, we investigate the different scenarios with ASCAD, Zaid, and NoConv1 networks. In all scenarios, we have an example that shows that a transferred attack can improve the efficiency of the attack compared to a pre-trained attack or regular attack. For the scenarios about the transferring between EM probe position and type, in all cases transferred attack show better performance than regular attacks and pre-trained attacks. The same conclusion can be made for the transfer between power and EM. For the transferring between different devices, we find some cases where the regular attack gave better performance than transferred attacks but the difference was small.

As mentioned in section 6.1, Thapar *et al.* [59] published a similar topic as our research but it is independent work. Even if their work is done in a different measurement setup, they apply the same approach and they come to the same conclusion as us about the efficiency of TL in SCA. In future work, we would like to study other TL techniques and compare them with each other to determine what are the best TL techniques to use in SCA because in our work and the work of Thapar *et al* [59] we studied one specific technique.

During this work, we have seen that some channels and devices are easier to attack than others. This observation leads to our next contribution. In Chapter 7, we will investigate the possibility of using a neural network to translate traces from one source to another which is easier to attack.

Chapter 7

Trace-to-trace translation for SCA

In the previous work, we note that some sources of information (channels, side channels, and devices) are easier to attack than others. This observation leads us to wonder whether we could translate one source of information into another that is easier to attack. We expect that the translation will have some effect like reducing noise or representing information in a form that seems easier to interpret for the network. Thus, in some cases, the network could produce better performance than regular attacks.

We define **Domain A** as being the space of traces obtained from the traditional source of information and **Domain B** as being the space of traces obtained from another source of information (which is easier to attack and chosen by the attacker). To achieve the translation of traces, we use specific architectures of neural networks. These architectures are Generative Adversarial Networks (GANs) and were presented in Chapter 2. GANs are known for their potential to generate data but also for image translation. Image translation consists in translating an image into another one, for example, it allows to transfer of the style of an image to another or to produce a high-resolution image from a low-resolution image. We want to reproduce the same principle as image translation but with traces. In this thesis, we introduce a new attack called "translated attack". The term “translated attack" refers to an attack using a GAN that learns to translate traces from domain A to domain B. After the training of the GAN, we use a pre-trained network (trained on domain B) and the synthetic attack set (traces translated using the GAN) to attack the device. In the following section, we present all works related to the uses of GAN in Side-Channel Attack (SCA). One of these works is of particular interest since the authors investigated a similar concept to “translated attacks" but under different conditions. Our contribution shows how we fine-tune two existing GAN to translate traces. We demonstrated that GAN successfully translate traces in two scenarios: translation from ElectroMagnetic (emission) (EM) to power sources and translation between different devices.

7.1 Related works & Motivations

Wang *et al.* [65] proposes a new method to deal with the issue of insufficient data during the profiling step. They introduce the Conditional Generative Adversarial Network (CGAN) in the context of side-channel attacks to generate new artificial traces and thus increase the size of the profiling set. Their method allows to increase the accuracy of the network during the profiling step and to reduces by half the required number of traces during the attack step. They evaluate their model on various profiling sets with different sizes and various types of Advanced Encryption Standard (AES) implementation such as unprotected, first-order masked and random delay protected.

Their generator is composed of N blocks, each block is composed of one *dense* layer with *leakly relu* as activation function, followed by *batch normalization* layer. The output of their generator is a dense layer with *tanh* as activation function. Their discriminator is composed of N blocks, each block is composed of one *dense* layer with *leakly relu* as activation function. The output of their discriminator is composed of *dense* layers with *sigmoid* as activation function.

In their first experiment, they selected 500 traces out of 10K to simulate the problem of insufficient training data. With these 500 traces, they generated 400 additional traces. They used two techniques, Correlation Power Analysis (CPA) and Differential Power Analysis (DPA), to assess the quality of the generated traces. This first experiment shows that the location of the leakage points and the amplitude of leakage peaks with CPA using the generated traces are consistent with the one obtained on the original training set. They also evaluate the efficiency of the best MultiLayers Perceptron (MLP) network from ASCAD [54] on original traces, generated traces, and a combination of both original and generated traces. The best performance was obtained with the combination of original traces and generated traces.

Further experiments were conducted with different sizes of training sets and different numbers of generated traces to study how the amount of traces in the original training set affects the quality of generated traces. The following sizes are selected to generate 400 additional traces: 50, 200, and 500. The same conclusions as in the first experiment are drawn namely the correlation at the leakage points is similar between the original traces and the generated traces. They also note that increasing the original training set reduces the noise at the leakless positions. The size of the original traces affects the quality of generated traces, thus better generated traces are obtained when the CGAN is trained using 500 original traces rather than using only 50 original traces. The best performance is obtained for the ASCAD network using 500 original traces and 400 generated traces (using 500 original traces for GAN training).

Finally, they also evaluate the capability of their CGAN to generate traces from unprotected and protected original traces. They use unprotected traces from DPAcontest v4 dataset and ChipWhisperer platform. For the attack results on DPAcontest v4, the best performance is obtained with 1000 original traces and 1000 generated traces, they reached a mean rank equal 0

with 207 traces. Thus, by adding 1000 generated traces, they obtained a mean rank equal to 0 with around 350 fewer traces. For the attack results on ChipWhisperer unprotected implementation, the best performance was obtained with 100 original traces and 400 generated traces, they reached a mean rank equal to 0 with 853 traces. Thus, by adding 400 generated traces, they obtained a mean rank equal to 0 with around 1000 fewer traces. They also used protected traces from the AES_RD dataset (desynchronized traces). For the attack results on AES_RD, the best performance was obtained with 10K original traces and 10K generated traces, they reached a mean rank equal to 0 with 1369 traces. Thus, by adding 10K generated traces, they obtained a mean rank equal to 0 with around 2700 fewer traces.

Mukhtar *et al.* [50] proposes a novel method of data augmentation based on CGAN and the Siamese network [1] to deal with the issue of imbalanced datasets. They lead analyses to compare the raw traces (extracted from a dataset) and the synthetic traces (generated by their model). To demonstrate the capability of their model to generalise any leakage, they use different datasets containing both symmetric and public-key algorithm implementations. They also compare their model (Siamese-CGAN) with the CGAN model proposed by Wang *et al.* [65]. Finally, they evaluate the performance of the Siamese-CGAN data augmentation by applying the actual Machine Learning (ML)-based side-channel attack: MLP and two Convolutional Neural Network (CNN) for symmetric algorithm implementation and one CNN for public-key implementation. Their results were conducted with the MLP network described by Benadjila *et al.* [4], the CNN denoted as ASCAD-CNN1 and ASCAD-CNN2 described by Benadjila *et al.* [4]. The dataset used during their experimentation is the ASCAD dataset and the ECC dataset.

First, they compared the convergence of their model with the existing CGAN model proposed by Wang *et al.* [65]. They show that their model provides a better convergence in terms of the loss function. Their model converges after around 100-150 epochs on the ASCAD datasets, and 700-1000 epochs on the ECC datasets whereas the CGAN did not converge well within 1000 epochs. Secondly, then conduct further experiments using a Siameses-CGAN network. They compare different networks (MLP, ASCAD-CNN1, and ASCAD-CNN2) with two datasets: one only composed of real traces and the other containing both real and fake traces. The fake traces are previously generated by the Siamese network. They observe that ASCAD-CNN2 performs better than MLP to recover the key with real and fake traces.

Wu *et al.* [68] proposes a pre-processing method based on a neural network to reduce the effect of hiding countermeasures. They experiment with a Convolutional Auto-Encoder (CAE) network on six different types of noise generated with a single countermeasure, or a combination of several countermeasures. These noises are artificially added to the ASCAD database [54]: Gaussian noise, shuffling, random delay, and clock jitter. To denoise or reduce the effect of countermeasures,

1. the principe of Siamese network is described into the chapter 2

authors assume that the attacker can have access to the clean traces corresponding to the noisy ones.

First, the authors investigate the performance of their model to denoise traces with Gaussian noise countermeasure. Authors add normally-distributed random values with zero mean and variance of eight to each Points of Interest (PoI) to artificially add the effect of Gaussian noise. When the Gaussian noise countermeasure is applied to original traces, their results show that CNN and template attack with Principal Component Analysis (PCA) converge to a mean rank equal to zero, while template attack without PCA and MLP do not succeed to converge.

Secondly, the authors investigate the performance of their model to denoise desynchronized traces. Authors add randomness to the time domain to artificially add the effect of desynchronization. When the desynchronization is applied to original traces, their results show that CNN converges to a mean rank equal to zero with 9630 traces, whereas the template attack with and without PCA, and MLP does not converge to a mean rank equal to zero.

The authors compare two methods to re-synchronize traces: using static alignment or CAE. They show that attacks work better on the output of the CAE than on statically aligned traces. They observe that the number of traces required by the CNN to obtain a mean rank equal to zero is reduced from 1180 to 822. The template attack and MLP required from 8,905 to 7,168, and from 10,000 to 6,398, thus the performance of these two models with resynchronized traces is improved.

Third, the authors investigate the performance of their model to denoise traces altered by random delay interrupts. Authors simulate random delay interrupts based on the *Floating Mean method* [13]. This method induces more variance compared to the uniform random delay interrupts. When the random delay is applied to the original traces, their results show that CNN is not powerful enough to discover the key even using 10,000 traces. All other attacks such as template attacks (with and without PCA) and MLP also provide poor performances and do not succeed to discover the key. We can observe that the random delay countermeasure drastically increases the difficulty of the attack impacting both the classical and the deep learning approaches. Authors compared two methods to denoise traces with random delay: frequency analysis method and CAE. When they denoise traces with CAE, they observe that the performance of CNN was improved because it required only 1322 traces to reach a mean rank equal to 0. For the template attack, it required only 8952 traces and MLP required only 3398 traces to converge until a mean rank equal to 0. They observe that the effect of random delay has been reduced dramatically with the help of the CAE.

Fourth, the authors investigate the performance of their model to denoise traces with the clock-jitter countermeasure. The clock-jitter countermeasure increases the randomness for each point in the time domain. They simulate clock-jitter countermeasure by adding or removing points with a pre-defined range. When the clock-jitter countermeasure is used, their results

show that no approaches (CNN, template attack with PCA, template attack without PCA, and MLP) succeed in retrieving the key using 10,000 attack traces. The approach with the lowest guessing entropy is the approach with CNN. Similarly to the random delay case, the two methods to denoise the jittered traces are frequency analysis and CAE. They observe that CAE can successfully reduce the effect of clock-jitters countermeasures, thus the performance of all approaches can be improved. For example, the CNN required only 8,045 traces to obtain the correct key, and for the MLP, we can reach a mean rank close to zero with 10,000 attack traces.

Finally, the authors investigate the performance of their model to denoise traces with shuffling countermeasures. Authors simulate the shuffling countermeasure by randomizing the access to the S-box. When the shuffling is applied to the original traces, their results show that the best performance is obtained by template attack with PCA because it required only 9,885 attack traces to reach the correct key. All other approach (CNN, template attack without PCA, and MLP) doesn't reach any mean rank equal to zero with 10,000 attack traces. Authors successfully improve the performance of template attack, CNN, and MLP by adding 10,000 more traces denoised by the CAE. When they denoised with their CAE, they observed that the performance of CNN was improved because it required 7,754 traces to discover the correct key. For the template attack, the performance was improved and it reaches a mean rank equal to six after 10,000 attack traces. For the MLP, the performance was improved compared to the shuffling traces but slightly worse compared to the augmented attack traces (augmented attack traces contains 10,000 more traces than attack trace).

Authors also study the efficiency of CAE to denoise several countermeasure combinations. They combined all previous countermeasures in the following order: shuffling, desynchronization, random delay interrupt, clock-jitter, and Gaussian noise. When all countermeasures are applied, no approach is able to discover the correct key with 10,000 attack traces. However, when they apply their CAE, the performance of all approaches is improved. They show that their CAE is able to reduce the combined effect of noise and countermeasures. Indeed, the CNN (resp. MLP) reaches a mean rank equal to 27 (resp. 61) using 10,000 denoised attack traces.

In their work, Wu *et al.* shows the impact of different countermeasures on the performance of classical models and deep learning models [68]. They also investigate the fact that CAE can be used to reduce the effect of various individual countermeasures, and various combined countermeasures. The main drawback of the described approach is that in a real-world situation, the attacker might not be able to have access to those clean traces (i.e. traces without any countermeasure present). We extend this work by presenting a method to translate traces from one channel, side channel, and device to another channel, side channel and device with GAN. Our method does not require this access to a clean dataset.

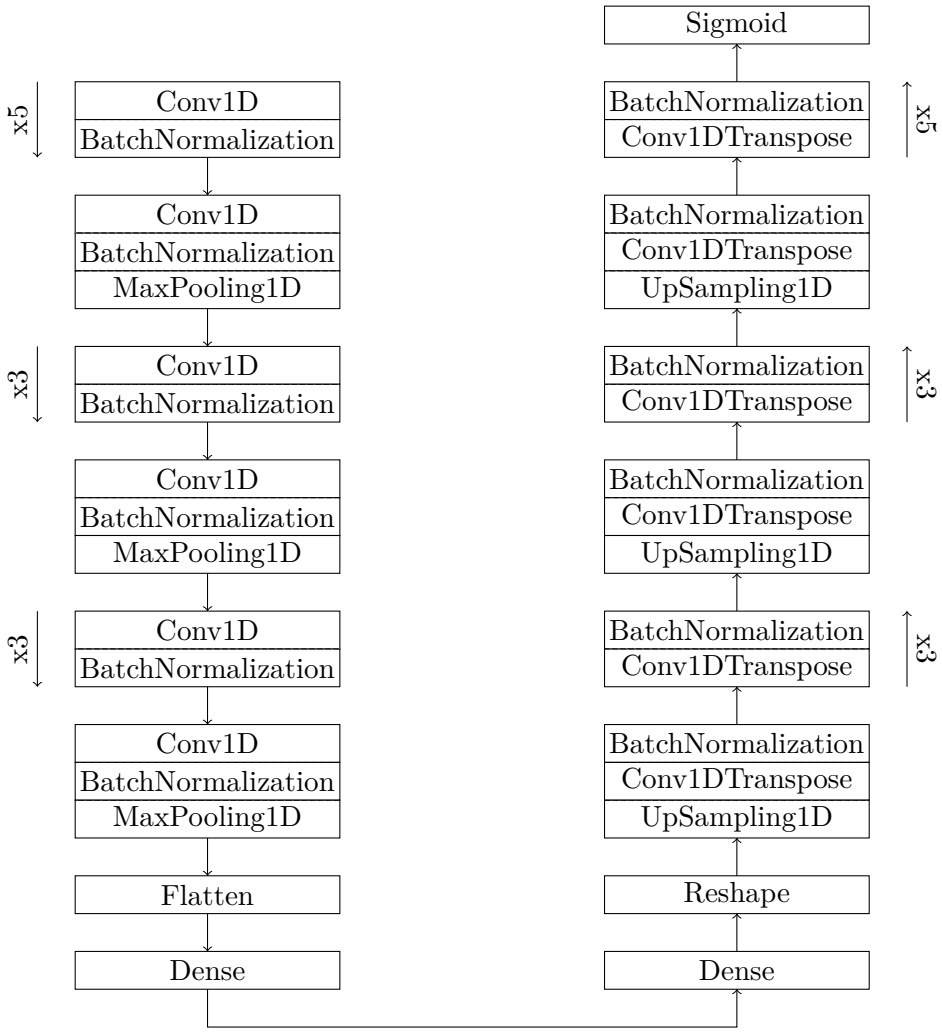


Figure 7.1 – Autoencoder

7.2 Approaches & Implementation details

The idea proposed in this Chapter is to use domain translation as a preprocessing technique to improve the quality of traces. This approach is enabled as soon as an attacker has access to paired datasets (that is, sharing the same intermediate target values/labels) from two different settings. Indeed, one of the settings must correspond to the attack dataset, and the other one corresponds to some settings for which the chosen attack performs better.

An attacker following the proposed approach will thus handle four different datasets. The goal is to improve the attack on domain A traces by translating them to domain B. To smooth the description of the technique let us introduce some notation.

— the **original attack dataset** contains the traces from which the attacker wants to extract

a secret key (thus from domain A);

— the **original training dataset A** contains traces from domain A;

— the **original training dataset B** contains traces from domain B that should be paired with the translation training dataset A;

— the **translated attack dataset** containing the translation from domain A to domain B of the *original attack dataset*.

Should this technique be used in a profiled setting, an additional fifth dataset would be added, namely

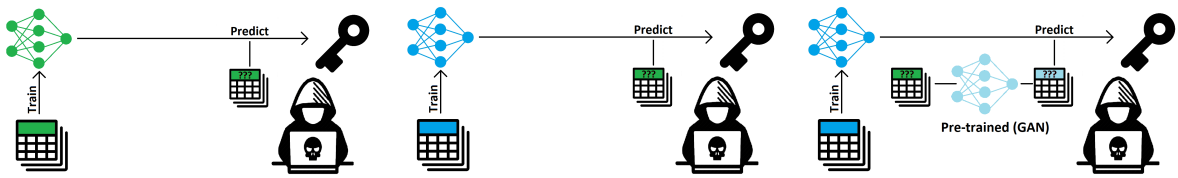
— the **training dataset** that contains training traces from domain B used to train the profiled attack.

We want to bring your attention to the fact that the only labelled dataset is the optional fifth one since translation training datasets only need to be paired (depending on the context, it may not necessarily imply being labelled).

The three training strategies are described in Fig 7.2, but we provide here a compact definition of the newly introduced attacks.

Definition 5 (Translated attack) *The translated attack is defined when:*

1. *Train a translator to translate traces from domain A to domain B using original training datasets A and B.*
2. *Use the trained translator to generate the translated* attack *dataset from the original attack dataset on domain A.*
3. *(optional) Train a profiled side-channel attack on the training dataset from domain B.*
4. *Attack the translated attack dataset to recover the key using an independent (un-)profiled attack.*



(a) Regular attack (b) Pretrained attack (c) Translated attack

Figure 7.2 – Training strategies for translated

We used the previously described database "DATABASE_STM32" with the same frame selection as in Chapter 6. For both the regular approach and the GAN approach, we used 80% of the profiling set for the training and the last 20% for the validation. We used 100% of the attack set for testing the Deep Neural Network (DNN) and the GAN.

7.3 Contribution

7.3.1 GAN architecture

We investigate two architectures of GAN: Pix2Pix and Speech Enhancement GAN (SEGAN). As mentioned in Chapter 2, Pix2Pix is well known for image translation, whereas SEGAN was designed to denoise audio waveforms. We tuned these architectures over a set of hyperparameters and selected the best-performing model.

		SEGAN	Pix2Pix
Hyperparameter	**Range**	**Selected**	**Selected**
Optimizer	{ Adam, RMSProp, SGD }	RMSProp	RMSProp
Activation function	{ Tanh, LeakLy ReLU, PReLU }	Tanh	Tanh
Batch size	{ 64, 128, 256 }	128	64 [2]
Epochs	{ 25, 50, 100, 200 }	200	200

Table 7.1 – Hyperparameter tuning

For the hyperparameter tuning, we use a paired dataset composed of 2 x 100,000 traces (100,000 for each domain). These traces were split into a training set composed of 80% of the paired dataset and a validation set composed of 20% of the paired dataset. We saved the best model over epochs based on the Signal-to-Noise Ratio (SNR) obtained on translated traces. Thus SNR is directly computed on the validation set during the training. We cannot directly compute the Guessing Entropy (GE) because of its computation time. Finally, we keep the model providing the highest SNR peaks for each set of hyperparameters. Our first analysis allows us to know that the best performance of Pix2Pix and SEGAN was obtained using Tanh as the activation function of the layers and RMSprop as the optimizer. We made a second analysis to confirm that the selected hyperparameters were also relevant from an attack perspective (and not only to generate the highest peak of SNR). In this second analysis, we computed the GE of all models with the translated attack set. As mentioned, the translate attack set is the translation of the attack set from domain A to domain B. Thus, the translated attack set is composed of 25,000 traces generated using a few fixed keys. This second analysis permits us to confirm that the best performance of Pix2Pix and SEGAN was obtained with the Tanh as activation function and RMSprop as the optimizer.

The SEGAN architecture we selected for our experiments (after fine-tuning it) is close to the architecture of the original SEGAN [51] that is used to a denoise audio waveform.

The generator takes as input a 700-point trace coming from the original domain and outputs a 700-point synthetic trace, which is the translation to the target domain. The generator is an auto-encoder that is composed of an encoding part and a decoding part. The output value of

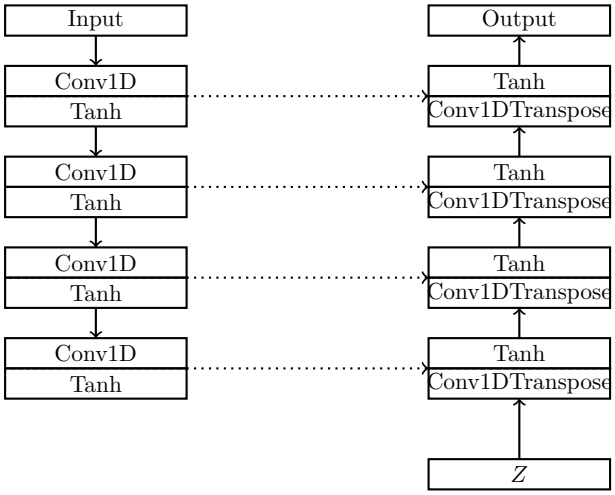


Figure 7.3 – Generator architecture (SEGAN)

the encoding part (or equivalently the input of the decoding part) lies in the so-called latent space. The latent representation of the trace has a shape equal to (2, 256) and this is where the random noise Z is added. The generator is illustrated in Figure 7.3. The encoder is composed of four blocks with one convolutional layer per block, a number of filters equal to (32, 64, 128, 256) with kernel size 31 (same padding), followed by a Tanh activation function. The decoding part is composed of four blocks and one transposed convolutional layer per block, a number of filters equal to (128, 64, 32, 1) with kernel size 31 (same padding), followed by a Tanh activation function. Each block of the decoding part is concatenated with the output of each encoding block, represented by a dotted arrow in Figure 7.3, it is the principle of the U-Net architecture [56].

The discriminator takes as input a combination of two 700-point traces: one trace coming from the original domain and one trace coming from the target one. This last trace may directly come from the target domain (real trace) or it could be a generated trace (fake trace). The discriminator is trained to distinguish between real and fake. More precisely, the discriminator outputs the probability that a trace from the target domain is a translation from the original domain. The discriminator is illustrated in Figure 7.4. It is composed of four blocks, and one convolutional layer per block, a number of filters equal to (32, 64, 128, 256) with a kernel size 31 (same padding), a Batch normalization layer, and a Tanh activation function. The discriminator has two final dense layers of 256 and 128 units.

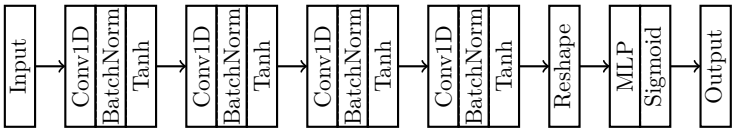


Figure 7.4 – Discriminator architecture (SEGAN)

As for SEGAN, we slightly adapted Pix2Pix to our datasets. The generator of the Pix2Pix takes as input a 700-point trace coming from the original domain and outputs a 700-point synthetic trace as for the SEGAN model. The generator is also an auto-encoder that is composed of an encoder part and decoder part. The Pix2Pix network has no latent representation between the encoder part and a decoder part, but just one convolutional layer composed of 512 filters. The encoder part is composed of seven blocks with one convolutional layer per block, a number of filters equal to $(32, 64, 128, 256, 256, 256, 256)$ with a kernel size of 11 (same padding), followed by a Batch Normalization layer and an activation function (see Table 7.1). The decoder part is composed of seven blocks and one transposed convolutional layer per block, a number of filters equal to (256, 256, 256, 256, 128, 64, 32) with a kernel size 11 (same padding), followed by a Batch Normalization layer and an activation function. The discriminator of the Pix2Pix is composed of five blocks with one convolutional layer per block, a number of filters equal to $(32, 64, 128, 256, 1)$ with a kernel size of 11 (same padding). In the original paper [32], the discriminator part of the Pix2Pix is implemented as PatchGAN, which means that the discriminator will classify 70×70 patches of the input image as real or fake. In our case, we adapt the output of the discriminator to output one single value.

During the tuning phase, Pix2Pix always provided lower performance (e.g., lower or fewer SNR peaks). Hence, SEGAN has been selected for our experiments on translation in the next sections.

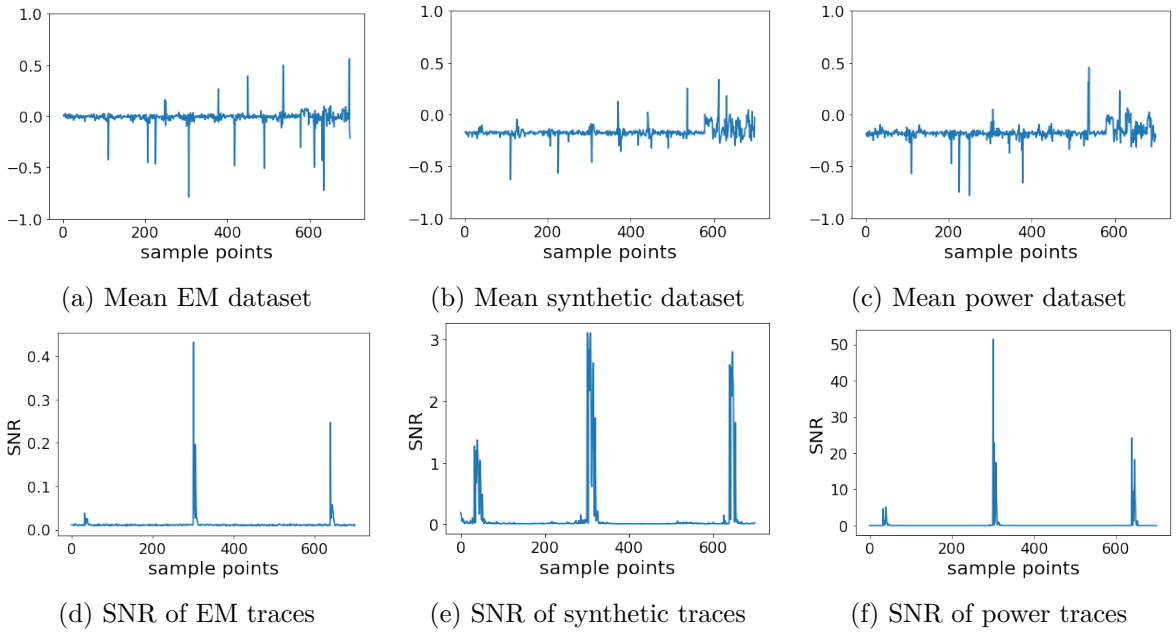
7.3.2 Translation from EM to Power

In this scenario, we investigate the capability of GAN to translate traces from EM radiations to power consumption on the same device. Depending on the device, we obtained various different SNR levels. For each device, we observed that the SNR corresponding to its EM and power traces have similar shapes while being of different magnitudes. From a quantitative point of view, the SNR value obtained from power traces is higher than from EM traces: in our setup EM measurements contain more noise.

For the experiments presented in this section, **Domain A** will correspond to EM traces (that are harder to attack) and **Domain B** to power ones.

STM32F2

Figure 7.5 shows the mean trace of the attack dataset (EM measurements), the synthetic dataset, and the power consumption, as well as their SNR levels. We observe that the SNR value is low (close to 0.4) for the EM channel, whereas the SNR value is high (close to 50) for the power channel, but their shapes are similar. The synthetic dataset corresponds to the set generated by GAN, which was trained to translate traces from EM to the power channel. First, We observe



(a) Mean EM dataset (b) Mean synthetic dataset (c) Mean power dataset

(d) SNR of EM traces (e) SNR of synthetic traces (f) SNR of power traces

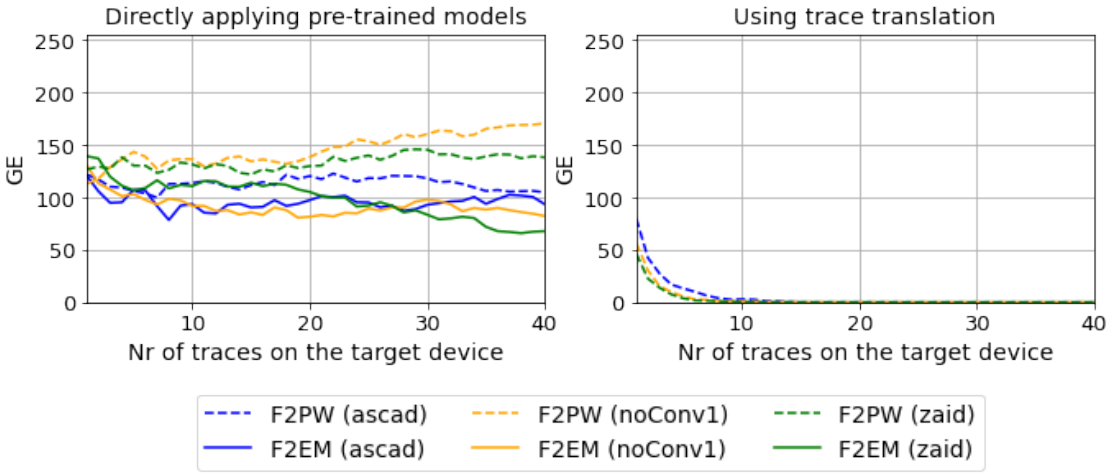
Figure 7.5 – Evaluation of STM32F2: (a) - (c) illustrates the mean trace and (d)-(f) the SNR of the attack dataset for the EM, power channel, and the translated synthetic trace dataset

that the shape of SNR is closer to the shape of EM and power and that the three main leakage position has been retrieved by the GAN. The SNR value of the synthetic dataset is close to 3, so the translation increases the magnitude and thus the amount of available information. Considering that the SNR value for the synthetic dataset is lower than for the power channel, which means that some information could not be reproduced.

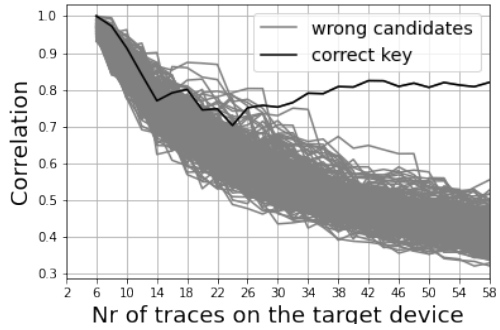
In Figure 7.6 (top) the GE in the profiled scenario when targeting the EM channel is plotted. First, we see that the performance is not specific to one network (ASCAD, Zaid, or noConv1). On the left side of the figure, one observes that when using directly the EM traces for attacking (labelled as F2EM), the attack does not succeed for any network. Next, similar performances can be observed when the network is trained on the power channel (labelled F2PW). The right plot shows that using the model trained on the power channel (F2PW) while attacking the translated synthetic traces, we observe that the attack rapidly converges towards a GE of 0 using less than 10 attack traces.

We further evaluate the outcome in the scenario of non-profiled attacks. Figure 7.6 (bottom) illustrates CPA using directly the EM dataset and when using the translated synthetic traces. Without translation, the correct key is found with approximately 30 traces, whereas with the translated synthetic dataset the key can be found using less than 15 traces.

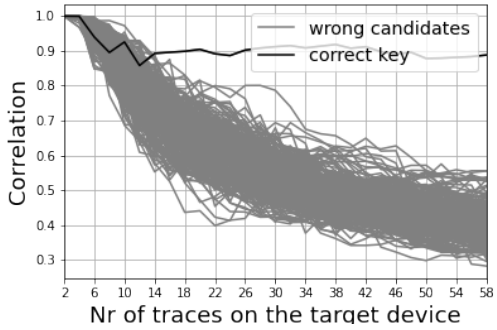
These results confirm that GAN is able to translate between the EM and power domain (i.e. F2PW not working on the EM dataset, but on the translated dataset for the three networks),



(a) GE for three networks; left: original EM dataset, right: translated synthetic dataset



(b) CPA on EM dataset



(c) CPA on synthetic dataset

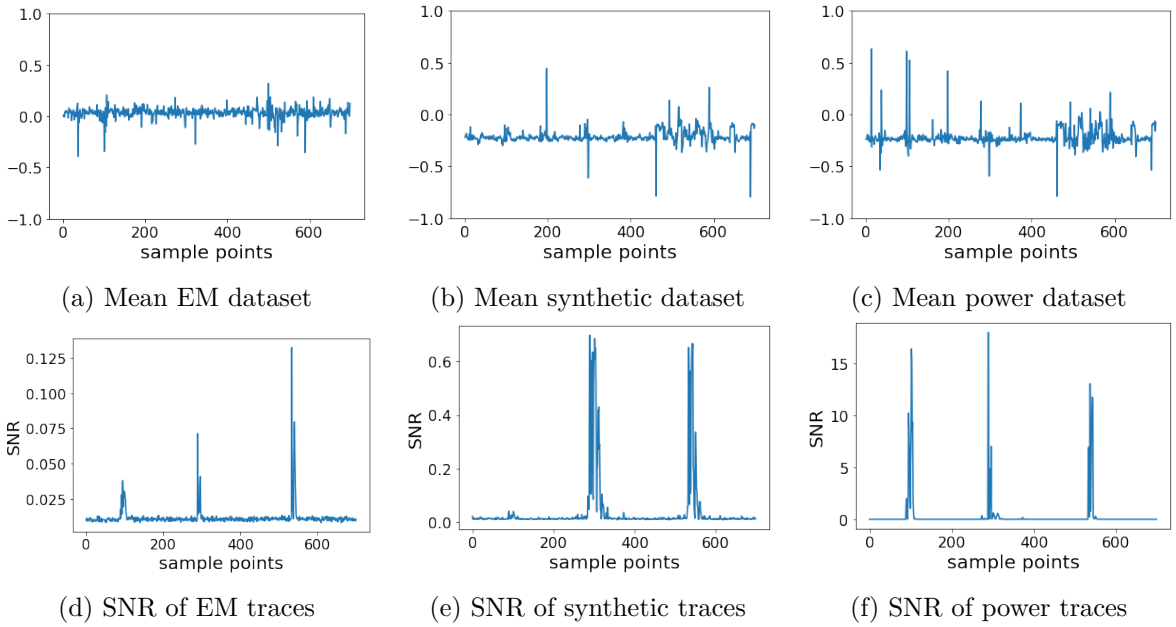
Figure 7.6 – Attack evaluation on STM32F2 EM (original and synthetic translated traces)

and further that the translation is increasing the exploitable side-channel information using a profiled DL-based attack or even when considering a classical non-profiled univariate attack.

STM32F4

Figure 7.7 (top) shows the mean trace obtained from STM32F4 of the dataset for EM/power measurements and the synthetic traces, where we visually observe a translation. At the bottom of the figure, we plot the SNR obtained on different attack datasets. The maximum SNR peak from the EM channel is close to 0.12, whereas it is close to 17.5 for the power channel, showing that the power channel is containing less noise. The GAN has at least retrieved the two main leakage positions. We observe that the SNR value of the synthetic attack set is improved by a factor of 5.

Figure 7.8 (top) shows the GE obtained when targeting device STM32F4 with EM for all three networks. As for STM32F2, we observe on the left that attacking the EM channel using



(a) Mean EM dataset (b) Mean synthetic dataset (c) Mean power dataset

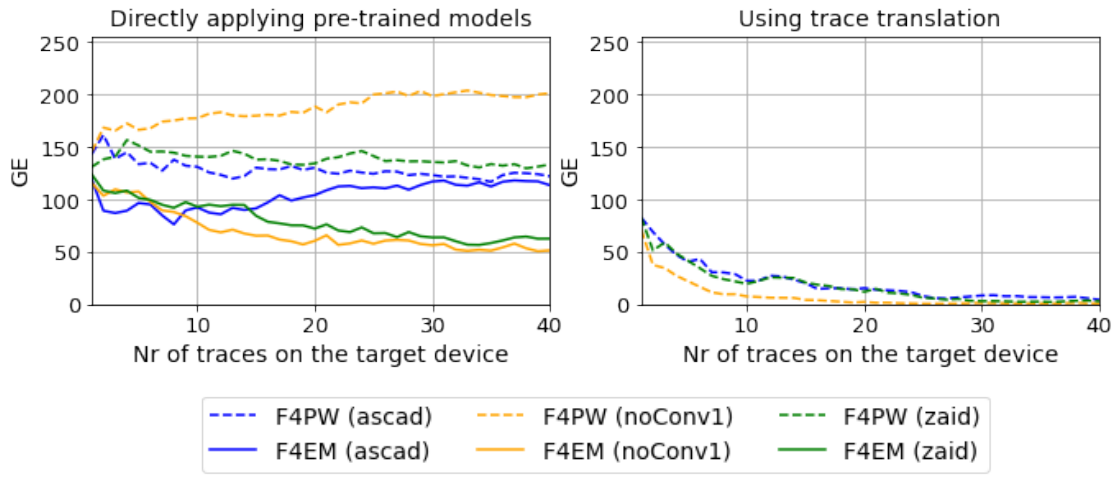
(d) SNR of EM traces (e) SNR of synthetic traces (f) SNR of power traces

Figure 7.7 – Evaluation of STM32F4: (a) - (c) illustrates the (mean) trace and (d)-(f) the SNR of the attack dataset for the EM, power channel, and the translated synthetic trace dataset

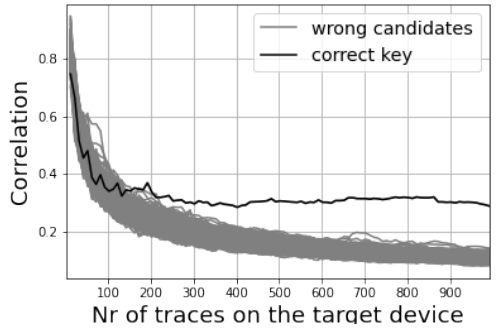
a model trained on EM (F4EM) or trained on power consumption (F4PW) does not converge within the given number of traces for any network. When using the translated synthetic traces and a model trained on the power channel (F4PW) GE reaches a mean rank equal to zero with only 20 traces for NoConv1 and around 30 traces for the other two networks. So, again by using the GAN translation, we could turn an unsuccessful attack into a successful one.

Figure 7.8 (bottom) shows that again the performance of CPA is improved as well. Using directly the EM dataset reveals the key using around 200 traces. On the other hand, when using the translated synthetic traces succeeds using around 50 traces.

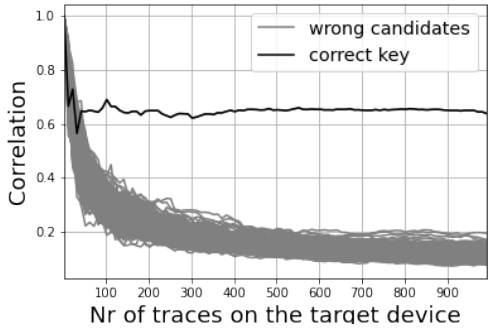
In this section, we demonstrated that a translation from EM to power consumption is possible and that it reduces the number of traces needed for a successful attack. On both datasets, the classical approach (attacking directly the noisy channel) with Deep Learning (DL) failed and we observe that directly using a network trained on the power channel (but without translating the attack dataset) leads to poor performances. Our results show that GANs can be used to translate traces from EM to the power channel, and the synthetic traces combined with a network trained on the power channel is successful. Additionally, our results show that the performance of CPA is greatly improved when attacking translated instead of the original traces.



(a) GE for three networks; left: original EM, right: translated synthetic dataset



(b) CPA on EM dataset



(c) CPA on synthetic dataset

Figure 7.8 – Attack evaluation on STM32F4 EM (original and synthetic translated traces)

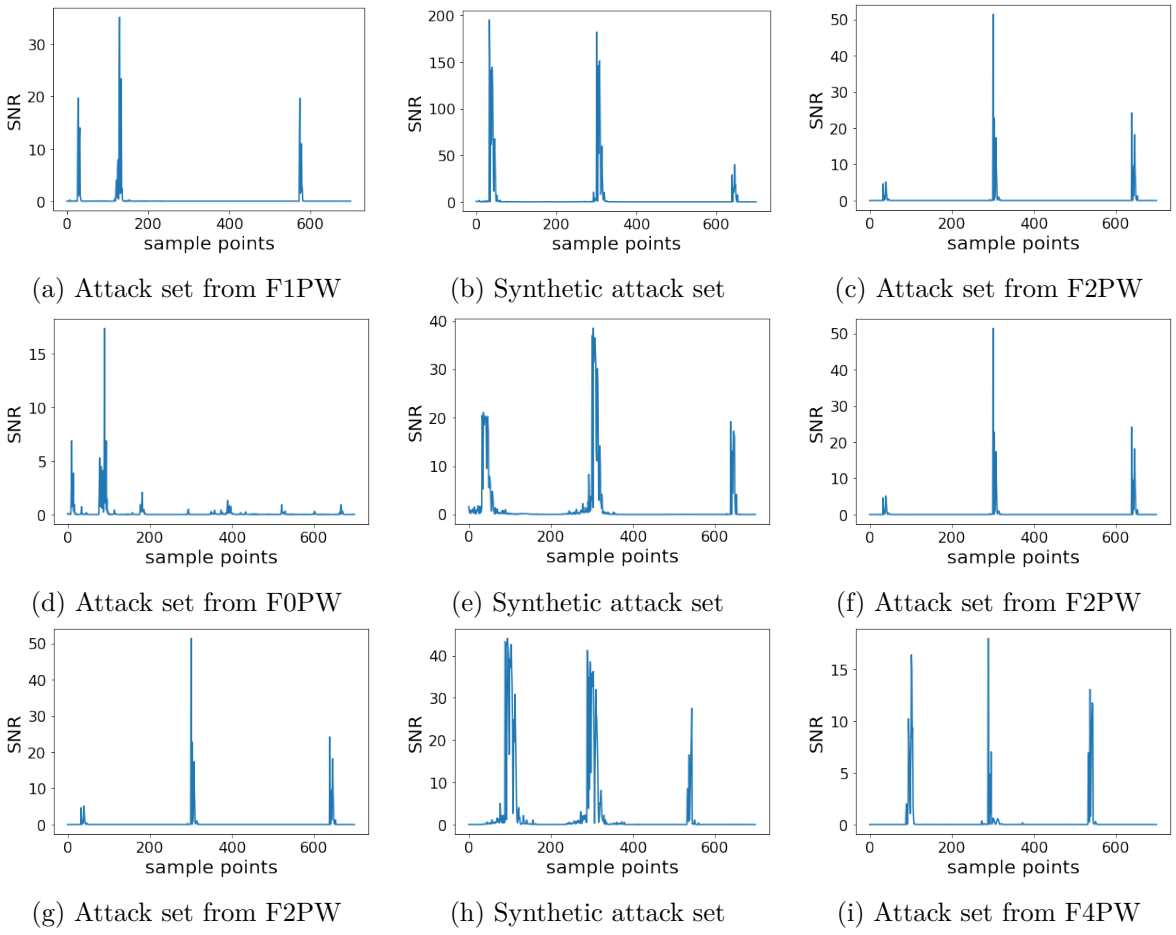
7.3.3 Cross-Device Translation

In this scenario, we investigate the capability of GAN to translate traces captured from one device to another. For the experiments presented in this section, **Domain A** will thus correspond to traces from one chip and **Domain B** to traces from another chip.

STM32F1 power to STM32F2 power

In Figure 7.9 (a)-(c), we plot the SNR evaluation obtained with different attack sets when translating F1 to F2 with power. The SNR value is close to 50 for F2, the synthetic attack set has an SNR value close to 200 (which is even larger than the target domain), and we can observe that GAN retrieved all three leakage positions from F2 which are at different time locations than F1.

In Figure 7.10 (top), we plot the GE obtained when we target F1 with power. Any of the DL-



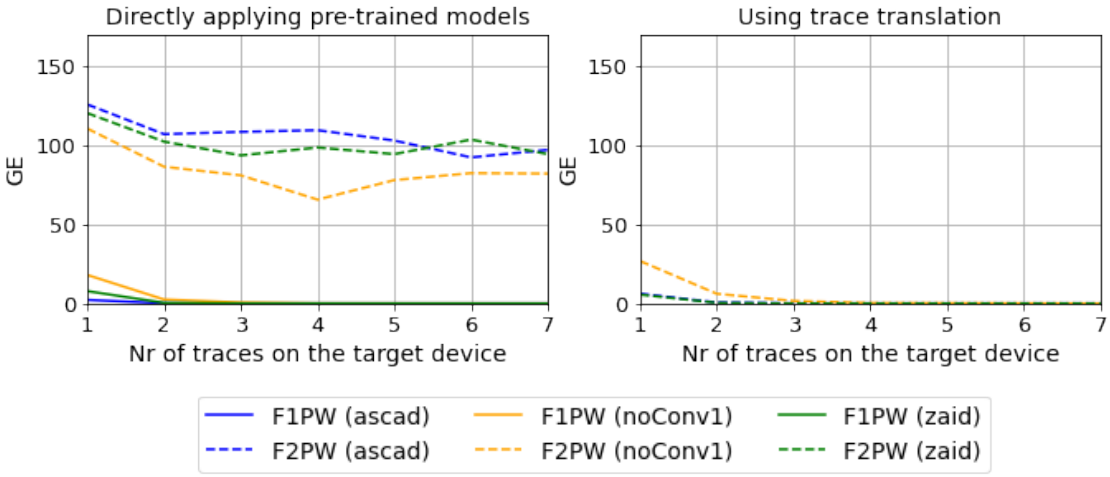
(a) Attack set from F1PW (b) Synthetic attack set (c) Attack set from F2PW

(d) Attack set from F0PW (e) Synthetic attack set (f) Attack set from F2PW

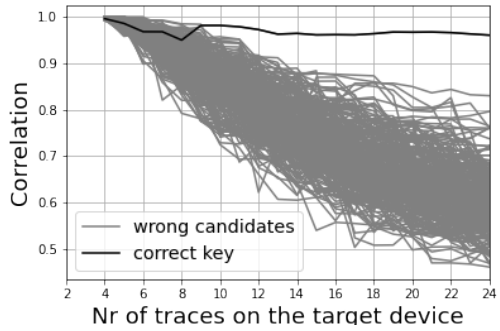
(g) Attack set from F2PW (h) Synthetic attack set (i) Attack set from F4PW

Figure 7.9 – SNR evaluation for each of scenarios considered: (a) - (c) F1PW translated to F2PW, (d)-(f) F0PW translated to F2PW, (g)-(i) F2PW translated to F4PW; left column domain A, middle column translated, right column domain B dataset.

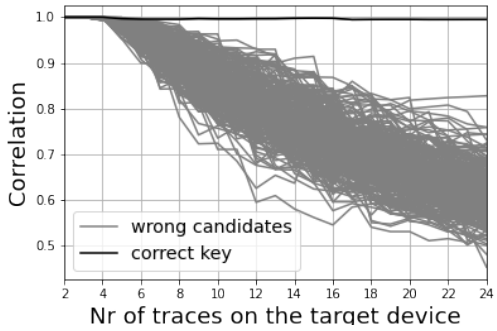
based attacks on the target domain (labelled F1PW) reaches a mean rank equal to zero below 2 traces. Using a model trained on F2 directly does not succeed, however, when translating to the domain F2, the ASCAD and Zaid networks succeed as well within 2 traces. Even though the performance using directly F2PW cannot be improved because of its high SNR by nature, this scenario shows that cross-device translation is possible. For CPA (given at the bottom of the figure) we see that the performance can be improved due to translation. Attacking the original traces is successful within 8 traces, whereas the correct key is found on the synthetic traces using 2 traces.



(a) GE; left: original dataset, right: translated synthetic dataset



(b) CPA on EM dataset



(c) CPA on synthetic dataset

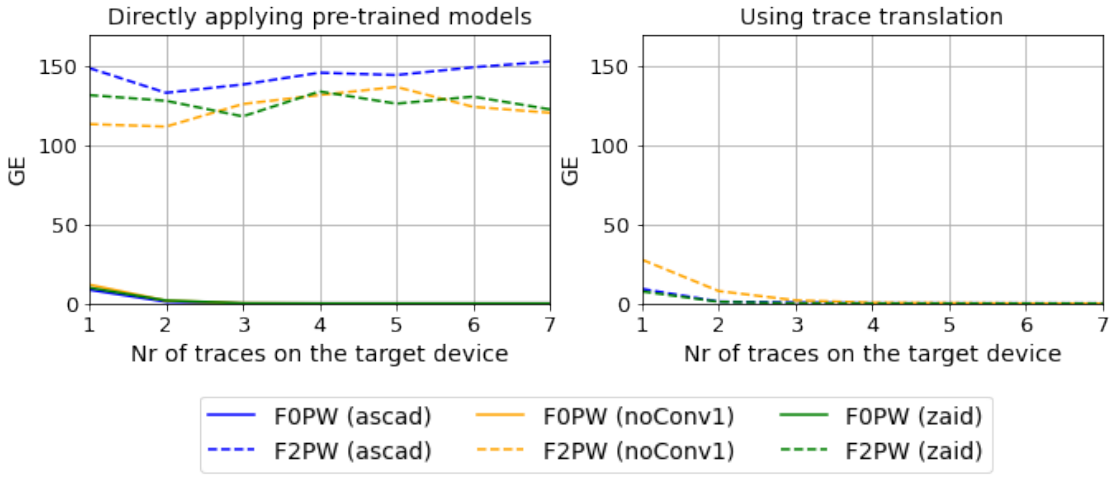
Figure 7.10 – Attack evaluation on STM32F1 (original and synthetic translated traces)

STM32F0 power to STM32F2 power

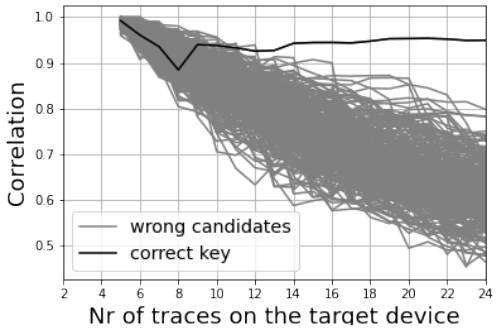
In Figure 7.9 (d)-(f), we plot the SNR evaluation when we translate from F0 to F2 device with power. The SNR value is close to 50 for the F2 device, and the synthetic attack set has an SNR close to 12 (which is smaller than the target domain), but again we can observe that GAN retrieved the leakage positions from F2, while being different in time and amount for F0 and F2.

In Figure 7.11 (top), we plot the GE obtained when we target F0 with power. As before, all DL-based attacks on the target domain (labelled F0PW) are already efficient (as the SNR is high enough), whereas the model trained on F2PW does not succeed on the original traces. Even though with a tiny difference, we observe that the best performance was obtained by applying GAN and the Zaid network.

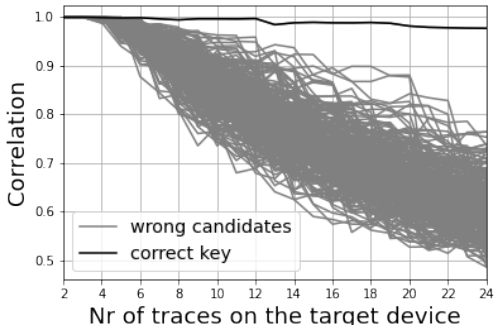
As in the previous scenario, we see that the translation improves the attackability with CPA. On the original dataset, the key can be found using around 12 traces, whereas on the translated



(a) GE; left: original dataset, right: translated synthetic dataset



(b) CPA on STM32F0



(c) CPA on synthetic dataset

Figure 7.11 – Attack evaluation on STM32F0 (original and synthetic translated traces)

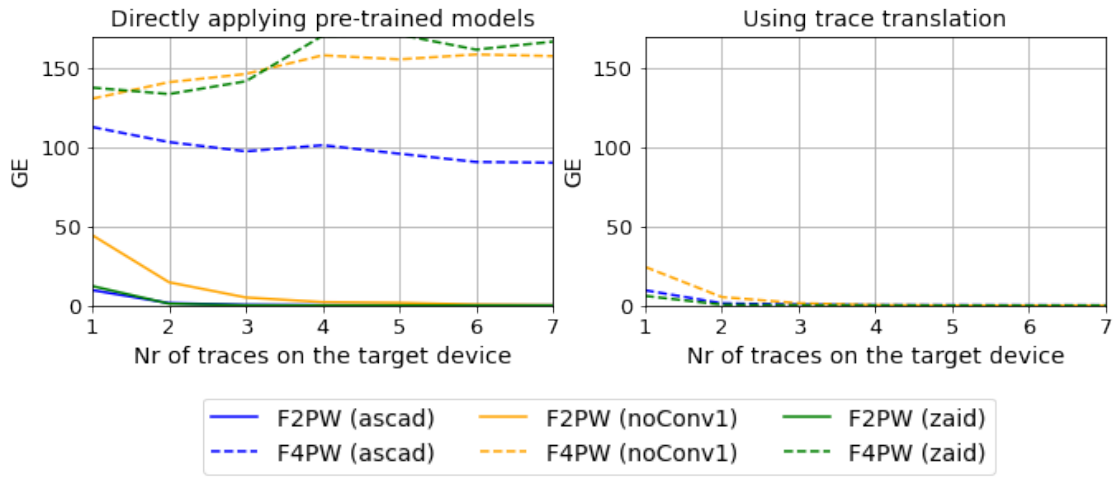
synthetic traces we see that the correct key can be found immediately using 2 traces.

STM32F2 power to STM32F4 power

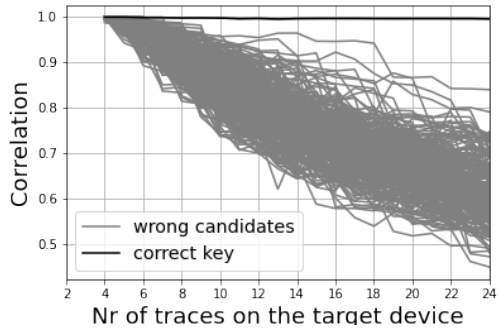
Unlike the previous scenarios, we now consider translation from a device with higher SNR to another one with lower SNR. In the situation where an attacker only has a dataset available (and the corresponding tuned network) for a device with less SNR, does translation between domains still successful? Could translation bring some improvement ?

In Figure 7.9 (g)-(i), we plot the SNR evaluation when we translate F2 to F4. The SNR value is close to 17.5 for the F4 device, the synthetic attack set has an SNR value close to 40 (which is larger than the target domain), and we can observe that GAN has recovered all three leakage positions at the same time positions as F4.

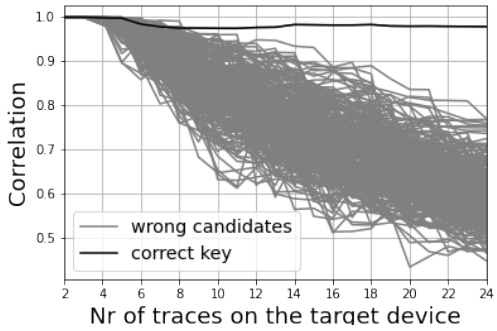
Figure 7.12 shows the GE obtained when targeting F2 with power. First, we see that indeed a translation to a higher noise domain is possible, as all networks trained on F4PW succeed on



(a) GE for three networks; left: original EM dataset, right: translated synthetic dataset



(b) CPA on STM32F2



(c) CPA on synthetic dataset

Figure 7.12 – Attack evaluation on STM32F2 (original and synthetic translated traces)

the translated dataset, but fail on the original dataset. Second, one can observe that the attack performance of the three neural networks is not degraded, but rather slightly improved. The increase may be explained by the fact, that even though F2 has a higher SNR peak, F4 contains three SNR peaks that are of a similar magnitude. We can see similar behaviour in the translated synthetic attack dataset, which shows three SNR peaks with comparable SNR levels. Indeed, when summing up all SNR values, the synthetic dataset achieves a higher value than F2.

The performance of CPA shows a degraded performance, which is expected as its a univariate attack, only considering one SNR peak at a time, where the magnitude of SNR is directly related to the success of CPA [18].

7.3.4 Conclusion

In this chapter, we want to investigate if GAN can be used to translate traces from one domain to another which is easier to attack. We see different scenarios of translation: translation from

EM to power and translation between different devices. First, our experiments show that the translation between EM and power is possible, and in all cases, it led to increased neural network efficiency (ASCAD, Zaid, NoConv1). Secondly, our experiments show that translation between different devices is also possible. We started to make the trace translation between devices from lower to higher SNR. In this case, the GAN successfully retrieves all leakage points and increase the SNR value. Then, we make the trace translation between devices from higher to lower SNR. In this case, the GAN also succeed to retrieve all leakage positions. We also apply CPA to validate that trace translation can be used with other attacks. For the translation between EM and power, trace translation permit significantly improves the CPA. For the translation between devices (from lower to higher SNR), trace translation permits also to increase in the efficiency of CPA. However, in the context of translation between devices from higher to lower SNR, trace translation has decreased the performance of CPA because CPA is a univariate attack, so it only considers one SNR peak at a time.

Conclusions and perspectives

The literature on SCA with DNN mostly considers a situation where the variation between the profiling set and attack set is small. The variation can be due to the difference between the probe position and type, the difference between side-channel, and the difference between devices. In a real-world scenario, this type of variation can occur, so it is necessary to study them in order to analyze their impact. In this thesis, we wanted to take into account these variations and study how they impact the performance of the DNN. To conduct this study, we created our own database to investigate different scenarios of variations in different settings.

Thus, we have investigated how other sources of information can be beneficial for a DNN through 3 contributions:

In chapter 5, we investigated the combination of different sources of information by using DL techniques. We defined "Multi-channel attacks" which refers to an attack that consists in combining several sources of information coming from different probe positions and types. We evaluate the efficiency of "Multi-channel attacks" with the ASCAD, Zaid, and the NoConv1 networks compared to the regular deep learning attacks. The evaluation was made in different contexts:

— Only adding noisy source compared to an original single channel.

— Adding noisy and clean source compared to an original single channel.

— Only adding clean source compared to an original single channel.

We show that depending on the context and the networks, the network is affected differently. The ASCAD network is more sensitive to noisy channels. We have seen that adding a channel that contains more noise decreases the performance of the ASCAD network, but adding clear channels allows for an increase in its performance. For the Zaid and NoConv1 network, we have seen that these two networks are sensitive to the variation between the channels. Their performance can be affected if we add other sources that have a high level of variations compared to the original channel. However, adding other sources that are close to the original channel, allows for an increase in the performance of these two networks.

In chapter 6, we investigate how other sources of information can be beneficial to initialize the weight of a DNN rather than random initialization. We defined "Transferred attacks" that refers to using another source of information to pre-trained a network before fine-tuning it with

the original sources of information. We evaluate the efficiency of "Transferred attacks" with ASCAD, Zaid, and the NoConv1 networks compared to the regular deep learning techniques with a limited dataset. The evaluation was made on different scenarios:

— Transferring between EM probe position and type

— Transferring between power and EM

— Transferring between different devices

For the scenarios about transfer between EM probe position and type, we show that "transferred attacks" outperform regular attacks and pre-trained attacks. The same conclusion is made concerning the transfer between EM and power. For the scenarios about transfer between devices, we show that sometimes the regular attacks are still better than transferred attacks, but the difference is small.

In chapter 7, we investigate how GAN can be used to translate traces from one source to another that is easier to attack. We defined "Translated attacks" which refers to using a GAN to translate an original attack set to a synthetic attack set. We assume that the synthetic attack set is easier to attack. Like the two first chapters 5 and 6, we evaluate the efficiency of "Translated attack" with the ASCAD, Zaid, and NoConv1 networks compared to the regular deep learning attacks. The evaluation was made in different contexts:

— Translation from EM to power.

— Translation between devices.

We show that GAN successfully translate traces from EM to power, and between devices. We have seen that the performance of the ASCAD, Zaid, and NoConv1 networks can be improved with a translation from EM to power. For the translation between devices, if the translation goes from noisy to a clear source of information, this can increase the performance of the three networks. However, if the translation goes from clear to noisy, the regular attack remains better.

We have shown that other sources of information can be beneficial for deep neural networks. In this thesis, we have mainly studied three new attacks. However, other attacks can be thought of with other information sources. This topic can open the door to other types of attacks, consequently, it is important to study them. We have been thinking about several areas of research. For future works about the combination of side-channel information. We thought of extending the combination to other auxiliary channels. For example, we can try to combine traces from power consumption with acoustic traces. We would be curious to know if the current countermeasures also hide information in the acoustic traces, and if they do not, how beneficial it would be to combine them for the neural network. For future work on the Transfer Learning (TL), we would like to extend our approach with different techniques of TL to determine if it is still suitable, and if they do, how much it can be beneficial for the neural network. Our work on

GANs remains the most optimistic. We would like to see in future work what the limits of these networks are. For example, regarding trace translation, what are the limits, can we translate all types of auxiliary channels?

In the future, I intend to continue to study the problem of generalization that we can have in Artificial Intelligence (AI), especially in the IT Security domain. I am convinced that we can make more robust and more general approaches based on AI because new technologies are appearing that make the algorithms of AI more efficient [62]. Another point I would like to study in AI is explainability. I think that an Explainable Artificial Intelligence (xai) algorithm would allow having more confidence in the decision-making and prediction made by the AI, and would allow solving more quickly the generalization and robustness problem of AI.

List of publications

[1] C. Genevey-Metat, B. Gérard, and A. Heuser, "Combining sources of side-channel information," in *C&ESAR 2019*, Rennes, France, Nov. 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-02456646

[2] C. Genevey-Metat, A. Heuser, and B. Gérard, "Train or adapt a deeply learned profile?" in *Progress in Cryptology - LATINCRYPT 2021 - 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6-8, 2021, Proceedings*, ser. Lecture Notes in Computer Science, P. Longa and C. Ràfols, Eds., vol. 12912. Springer, 2021, pp. 213–232. [Online]. Available: https://doi.org/10.1007/978-3-030-88238-9_11

[3] C. Genevey-Metat, A. Heuser, and B. Gérard, "Trace-to-trace translation for SCA," in *CARDIS 2021 - 20th Smart Card Research and Advanced Application Conference*, Luebeck, Germany, Nov. 2021. [Online]. Available: https://hal.inria.fr/hal-03553723

BIBLIOGRAPHY

[1] URL https://github.com/ANSSI-FR/secAES-ATmega8515.

[2] Dakshi Agrawal, Josyula R. Rao, and Pankaj Rohatgi. Multi-channel attacks. In Colin D. Walter, Çetin K. Koç, and Christof Paar, editors, *CHES 2003*, pages 2–16, 2003.

[3] Lejla Batina and Matthew Robshaw, editors. *Cryptographic Hardware and Embedded Systems - CHES 2014 - 16th International Workshop, Busan, South Korea, September 23-26, 2014. Proceedings*, volume 8731 of *Lecture Notes in Computer Science*, 2014. Springer. ISBN 978-3-662-44708-6. doi: 10.1007/978-3-662-44709-3. URL https://doi.org/10.1007/978-3-662-44709-3.

[4] Ryad Benadjila, Emmanuel Prouff, Rémi Strullu, Eleonora Cagli, and Cécile Dumas. Deep learning for side-channel analysis and introduction to ASCAD database. *Journal of Cryptographic Engineering*, 10(2):163–188, June 2020. doi: 10.1007/s13389-019-00220-8. URL https://hal.archives-ouvertes.fr/hal-02984494.

[5] Shivam Bhasin, Anupam Chattopadhyay, Annelie Heuser, Dirmanto Jap, Stjepan Picek, and Ritu Ranjan Shrivastwa. Mind the portability: A warriors guide through realistic profiled side-channel analysis. In *27th Annual Network and Distributed System Security Symposium, NDSS 2020*. The Internet Society, 2020.

[6] Eric Brier, Christophe Clavier, and Francis Olivier. Correlation power analysis with a leakage model. In Marc Joye and Jean-Jacques Quisquater, editors, *Cryptographic Hardware and Embedded Systems - CHES 2004*, pages 16–29, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg. ISBN 978-3-540-28632-5.

[7] Eleonora Cagli, Cécile Dumas, and Emmanuel Prouff. Convolutional Neural Networks with Data Augmentation against Jitter-Based Countermeasures. In *Cryptographic Hardware and Embedded Systems - CHES 2017 - 19th International Conference*, Taipei, Taiwan, September 2017. URL https://hal.archives-ouvertes.fr/hal-01661212.

[8] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, and Pankaj Rohatgi. Towards sound approaches to counteract power-analysis attacks. In Michael Wiener, editor, *Advances in Cryptology — CRYPTO' 99*, pages 398–412, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg. ISBN 978-3-540-48405-9.

[9] Suresh Chari, Josyula R. Rao, and Pankaj Rohatgi. Template attacks. In Burton S. Kaliski, çetin K. Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2002*, pages 13–28, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg. ISBN 978-3-540-36400-9.

[10] Christophe Clavier, Jean-Sébastien Coron, and Nora Dabbous. Differential power analysis in the presence of hardware countermeasures. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 252–263, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-44499-2.

[11] Jean-Sébasticn Coron, Paul Kocher, and David Naccache. Statistics and secret leakage. In Yair Frankel, editor, *Financial Cryptography*, pages 157–173, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-45472-4.

[12] Jean-Sébastien Coron and Louis Goubin. On boolean and arithmetic masking against differential power analysis. In Çetin K. Koç and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, pages 231–237, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg. ISBN 978-3-540-44499-2.

[13] Jean-Sébastien Coron and Ilya Kizhvatov. An efficient method for random delay generation in embedded software. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 156–170, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg. ISBN 978-3-642-04138-9.

[14] Jean-Sébastien Coron and Ilya Kizhvatov. Analysis and improvement of the random delay countermeasure of ches 2009. In Stefan Mangard and François-Xavier Standaert, editors, *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 95–109, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. ISBN 978-3-642-15031-9.

[15] Damien Couroussé, Thierno Barry, Bruno Robisson, Philippe Jaillon, Olivier Potin, and Jean-Louis Lanet. Runtime code polymorphism as a protection against side channel attacks. In Sara Foresti and Javier Lopez, editors, *Information Security Theory and Practice*, pages 136–152, Cham, 2016. Springer International Publishing. ISBN 978-3-319-45931-8.

[16] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael, 1999.

[17] Debayan Das, Anupam Golder, Josef Danial, Santosh Ghosh, Arijit Raychowdhury, and Shreyas Sen. X-deepsca: Cross-device deep learning side channel attack. In *DAC 2019*, pages 134:1–134:6, 2019.

[18] Yunsi Fei, Qiasi Luo, and A. Adam Ding. A statistical model for dpa with novel algorithmic confusion analysis. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic*

Hardware and Embedded Systems – CHES 2012, pages 233–250, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-33027-8.

[19] Christophe Genevey-Metat, Benoît Gérard, and Annelie Heuser. Combining sources of side-channel information. In *C&ESAR 2019*, Rennes, France, November 2019. URL `https://hal.archives-ouvertes.fr/hal-02456646`.

[20] Christophe Genevey-Metat, Annelie Heuser, and Benoît Gérard. Train or adapt a deeply learned profile? In Patrick Longa and Carla Ràfols, editors, *Progress in Cryptology - LATINCRYPT 2021 - 7th International Conference on Cryptology and Information Security in Latin America, Bogotá, Colombia, October 6-8, 2021, Proceedings*, volume 12912 of *Lecture Notes in Computer Science*, pages 213–232. Springer, 2021. doi: 10.1007/978-3-030-88238-9_11. URL `https://doi.org/10.1007/978-3-030-88238-9_11`.

[21] Christophe Genevey-Metat, Annelie Heuser, and Benoît Gérard. Trace-to-trace translation for SCA. In *CARDIS 2021 - 20th Smart Card Research and Advanced Application Conference*, Luebeck, Germany, November 2021. URL `https://hal.inria.fr/hal-03553723`.

[22] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. Stealing keys from pcs using a radio: Cheap electromagnetic attacks on windowed exponentiation. Cryptology ePrint Archive, Report 2015/170, 2015. `https://eprint.iacr.org/2015/170`.

[23] Daniel Genkin, Lev Pachmanov, Itamar Pipman, and Eran Tromer. Ecdh key-extraction via low-bandwidth electromagnetic attacks on pcs. In Kazue Sako, editor, *Topics in Cryptology - CT-RSA 2016*, pages 219–235, Cham, 2016. Springer International Publishing. ISBN 978-3-319-29485-8.

[24] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[26] Benjamin Hettwer, Daniel Fennes, Sebastien Leger, Jan Richter-Brockmann, Stefan Gehrer, and Tim Güneysu. Deep learning multi-channel fusion attack against side-channel protected hardware. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020. doi: 10.1109/DAC18072.2020.9218705.

[27] Annelie Heuser and Michael Zohner. Intelligent Machine Homicide - Breaking Cryptographic Devices Using Support Vector Machines. In Schindler and Huss [57], pages 249–

264. ISBN 978-3-642-29911-7. doi: 10.1007/978-3-642-29912-4. URL https://doi.org/10.1007/978-3-642-29912-4.

[28] Annelie Heuser, Olivier Rioul, and Sylvain Guilley. Good Is Not Good Enough - Deriving Optimal Distinguishers from Communication Theory. In Batina and Robshaw [3], pages 55–74. ISBN 978-3-662-44708-6. doi: 10.1007/978-3-662-44709-3_4. URL http://dx.doi.org/10.1007/978-3-662-44709-3_4.

[29] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Netw.*, 2(5):359–366, July 1989. ISSN 0893-6080.

[30] Gabriel Hospodar, Benedikt Gierlichs, Elke De Mulder, Ingrid Verbauwhede, and Joos Vandewalle. Machine learning in side-channel analysis: a first study. *J. Cryptographic Engineering*, 1(4):293–302, 2011. doi: 10.1007/s13389-011-0023-x. URL https://doi.org/10.1007/s13389-011-0023-x.

[31] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A Efros. Image-to-image translation with conditional adversarial networks. *CVPR*, 2017.

[32] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros. Image-to-image translation with conditional adversarial networks, 2018.

[33] Ramcharan Kakarla, Sundar Krishnan, and Sridhar Alla. *Supervised Learning Algorithms*, pages 143–203. Apress, Berkeley, CA, 2021. ISBN 978-1-4842-6500-0. doi: 10.1007/978-1-4842-6500-0_5. URL https://doi.org/10.1007/978-1-4842-6500-0_5.

[34] T. W. Kim, T. Kim, and Seokhie Hong. Breaking korea tansit card with side-channel analysis attack-unauthorized recharging -. 2017.

[35] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. *CoRR*, abs/1706.02515, 2017. URL http://arxiv.org/abs/1706.02515.

[36] Paul Kocher, Jann Horn, Anders Fogh, , Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *40th IEEE Symposium on Security and Privacy (S&P'19)*, 2019.

[37] Paul C. Kocher. Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems. In *Proceedings of the 16th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '96, page 104–113, Berlin, Heidelberg, 1996. Springer-Verlag. ISBN 3540615121.

[38] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *Proceedings of the 19th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '99, page 388–397, Berlin, Heidelberg, 1999. Springer-Verlag. ISBN 3540663479.

[39] Paul C. Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In *CRYPTO '99*, pages 388–397, 1999.

[40] Paul C Kocher, Joshua M Jaffe, Benjamin Jun, and Pankaj Rohatgi. Introduction to differential power analysis. *Journal of Cryptographic Engineering*, 1(1):5–27, 2011. ISSN 2190-8508. doi: 10.1007/s13389-011-0006-y.

[41] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. Side Channel Attack: an Approach Based on Machine Learning. In *Second International Workshop on Constructive SideChannel Analysis and Secure Design*, pages 29–41. Center for Advanced Security Research Darmstadt, 2011.

[42] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.

[43] Lu Lu. Dying relu and initialization: Theory and numerical examples. *Communications in Computational Physics*, 28(5):1671–1706, Jun 2020. ISSN 1991-7120. doi: 10.4208/cicp.oa-2020-0165. URL `http://dx.doi.org/10.4208/cicp.OA-2020-0165`.

[44] Houssem Maghrebi, Thibault Portigliatti, and Emmanuel Prouff. Breaking cryptographic implementations using deep learning techniques. In Claude Carlet, M. Anwar Hasan, and Vishal Saraswat, editors, *Security, Privacy, and Applied Cryptography Engineering*, pages 3–26, Cham, 2016. Springer International Publishing. ISBN 978-3-319-49445-6.

[45] Loïc Masure, Cécile Dumas, and Emmanuel Prouff. Gradient Visualization for General Characterization in Profiling Attacks. In *Constructive Side-Channel Analysis and Secure Design*, volume 11421 of *Constructive Side-Channel Analysis and Secure Design - 10th International Workshop, {COSADE} 2019, Darmstadt, Germany, April 3-5, 2019, Proceedings*, pages 145–167, Darmstadt, France, April 2019. Springer. doi: 10.1007/978-3-030-16350-1_9. URL `https://hal.archives-ouvertes.fr/hal-02087384`.

[46] Adam Matthews. Side-channel attacks on smartcards. *Network Security*, 2006(12):18–20, 2006. ISSN 1353-4858. doi: https://doi.org/10.1016/S1353-4858(06)70465-2. URL `https://www.sciencedirect.com/science/article/pii/S1353485806704652`.

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

[48] Thomas S. Messerges. Securing the aes finalists against power analysis attacks. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, and Bruce Schneier, editors, *Fast Software Encryption*, pages 150–164, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg. ISBN 978-3-540-44706-1.

[49] Mehdi Mirza and Simon Osindero. Conditional generative adversarial nets, 2014.

[50] Naila Mukhtar, Lejla Batina, Stjepan Picek, and Yinan Kong. Fake it till you make it: Data augmentation using generative adversarial networks for all the crypto you need on small devices. Cryptology ePrint Archive, Report 2021/991, 2021. `https://ia.cr/2021/991`.

[51] Santiago Pascual, Antonio Bonafonte, and Joan Serrà. Segan: Speech enhancement generative adversarial network, 2017.

[52] Guilherme Perin, L. Chmielewski, and S. Picek. Strength in numbers: Improving generalization with ensembles in machine learning-based profiled side-channel analysis. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020:337–364, 2020.

[53] 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 Transactions on Cryptographic Hardware and Embedded Systems*, 2019 (1):209–237, Nov. 2018. doi: 10.13154/tches.v2019.i1.209-237. URL `https://tches.iacr.org/index.php/TCHES/article/view/7339`.

[54] Emmanuel Prouff, Remi Strullu, Ryad Benadjila, Eleonora Cagli, and Cecile Dumas. Study of deep learning techniques for side-channel analysis and introduction to ascad database. Cryptology ePrint Archive, Report 2018/053, 2018. `https://eprint.iacr.org/2018/053`.

[55] Jorai Rijsdijk, Lichao Wu, Guilherme Perin, and Stjepan Picek. Reinforcement learning for hyperparameter tuning in deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2021/071, 2021. `https://ia.cr/2021/071`.

[56] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[57] Werner Schindler and Sorin A. Huss, editors. *Constructive Side-Channel Analysis and Secure Design - Third International Workshop, COSADE 2012, Darmstadt, Germany, May*

3-4, 2012. Proceedings, volume 7275 of *Lecture Notes in Computer Science*, 2012. Springer. ISBN 978-3-642-29911-7. doi: 10.1007/978-3-642-29912-4. URL https://doi.org/10.1007/978-3-642-29912-4.

[58] François-Xavier Standaert and Cedric Archambeau. Using subspace-based template attacks to compare and combine power and electromagnetic information leakages. In Elisabeth Oswald and Pankaj Rohatgi, editors, *CHES 2008*, pages 411–425, 2008.

[59] Dhruv Thapar, Manaar Alam, and Debdeep Mukhopadhyay. Transca: Cross-family profiled side-channel attacks using transfer learning on deep neural networks. *IACR Cryptol. ePrint Arch.*, 2020:1258, 2020.

[60] Dhruv Thapar, Manaar Alam, and Debdeep Mukhopadhyay. Deep learning assisted cross-family profiled side-channel attacks using transfer learning. In *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, pages 178–185, 2021.

[61] A. M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950. ISSN 00264423. URL http://www.jstor.org/stable/2251299.

[62] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

[63] Nicolas Veyrat-Charvillon, Marcel Medwed, Stéphanie Kerckhof, and François-Xavier Standaert. Shuffling against side-channel attacks: A comprehensive study with cautionary note. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology – ASIACRYPT 2012*, pages 740–757, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg. ISBN 978-3-642-34961-4.

[64] Huanyu Wang, Sebastian Forsmark, Martin Brisfors, and Elena Dubrova. Multi-source training deep-learning side-channel attacks. In *2020 IEEE 50th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 58–63, 2020. doi: 10.1109/ISMVL49045.2020.00-29.

[65] Ping Wang, Ping Chen, Zhimin Luo, Gaofeng Dong, Mengce Zheng, Nenghai Yu, and Honggang Hu. Enhancing the performance of practical profiling side-channel attacks using conditional generative adversarial networks, 2020.

[66] Lennert Wouters, Victor Arribas, Benedikt Gierlichs, and Bart Preneel. Revisiting a methodology for efficient cnn architectures in profiling attacks. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(3):147–168, Jun. 2020. doi: 10.13154/tches.v2020.i3.147-168. URL https://tches.iacr.org/index.php/TCHES/article/view/8586.

[67] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020(4):389–415, Aug. 2020. doi: 10.13154/tches.v2020.i4.389-415. URL https://tches.iacr.org/index.php/TCHES/article/view/8688.

[68] Lichao Wu and Stjepan Picek. Remove some noise: On pre-processing of side-channel measurements with autoencoders. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2020, Issue 4:389–415, 2020. doi: 10.13154/tches.v2020.i4.389-415. URL https://tches.iacr.org/index.php/TCHES/article/view/8688.

[69] Lichao Wu, Guilherme Perin, and Stjepan Picek. I choose you: Automated hyperparameter tuning for deep learning-based side-channel analysis. Cryptology ePrint Archive, Report 2020/1293, 2020. https://ia.cr/2020/1293.

[70] Lichao Wu, Guilherme Perin, and Stjepan Picek. The best of two worlds: Deep learning-assisted template attack. Cryptology ePrint Archive, Report 2021/959, 2021. https://ia.cr/2021/959.

[71] Wei Yang, Yongbin Zhou, Yuchen Cao, Hailong Zhang, Qian Zhang, and Huan Wang. Multi-channel fusion attacks. *Trans. Info. For. Sec.*, 12(8):1757–1771, August 2017. ISSN 1556-6013. doi: 10.1109/TIFS.2017.2672521. URL https://doi.org/10.1109/TIFS.2017.2672521.

[72] Yuval Yarom and Katrina Falkner. Flush+reload: A high resolution, low noise, l3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, San Diego, CA, August 2014. USENIX Association. ISBN 978-1-931971-15-7. URL https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/yarom.

[73] Gabriel Zaid, Lilian Bossuet, Amaury Habrard, and Alexandre Venelli. Methodology for efficient CNN architectures in profiling attacks. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2020(1):1–36, 2020. doi: 10.13154/tches.v2020.i1.1-36. URL https://doi.org/10.13154/tches.v2020.i1.1-36.

[74] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks, 2020.

COLLEGE MATHS, TELECOMS
DOCTORAL INFORMATIQUE, SIGNAL
BRETAGNE SYSTEMES, ELECTRONIQUE

Université de Rennes

Titre : Apprentissage automatique pour les attaques par canaux auxiliaires

Mot clés : Apprentissage automatique, apprentissage profond, attaques par canaux auxiliaire, réseaux GAN, apprentissage par transfert

Résumé : De nos jours, les systèmes embarqués sont plus nombreux et répandus dans la nature. Ces systèmes sont vulnérables aux attaques par canaux auxiliaires. Ces attaques consistent à utiliser des mesures physiques pour observer des fuites d'information sur des systèmes cryptographiques embarqués. Avec l'émergence de l'intelligence artificielle, de nouvelles attaques par canaux auxiliaires basées sur de l'apprentissage profond ont été développées. Dans un grand nombre de cas, ces nouvelles techniques d'attaque sont plus efficaces pour exploiter des fuites de données que des attaques classiques. Dans cette thèse, nous étudions les techniques d'apprentissage profond dans le contexte où nous avons plusieurs sources d'information disponibles. Dans un premier temps, nous avons cherché à savoir si la combinaison de plusieurs canaux d'observation provenant d'un même système mais de sondes différentes peut aider un attaquant à élaborer une attaque plus efficace qu'avec une source unique. Dans un deuxième temps, nous avons cherché à savoir si des ensembles de données supplémentaires qui n'ont pas été mesurés à partir de la même plateforme/sonde/système peuvent aider un attaquant à monter une attaque plus puissante ou bien à accélérer le temps d'une évaluation en facilitant la convergence d'un réseau. Nous avons utilisé la technique appelée apprentissage par transfert pour monter ce type d'attaque. Nous avons démontré que l'apprentissage par transfert permet en général de faciliter la convergence du réseau et, dans certains cas, d'améliorer son efficacité finale. Enfin, dans le contexte où nous avons deux ensembles de données avec des efficacités différentes du point de vue de l'attaque, nous avons étudié s'il était possible de traduire les traces d'un système qui est plus difficile à attaquer en d'autres traces d'un système qui est plus facile à attaquer.

Title: Machine learning for side-channel analysis

Keywords: Machine learning, deep learning, side channel attacks, GAN network, transfer learning

Abstract: Nowadays, embedded systems are more numerous and widespread in the real world. These systems are vulnerable to side channel attacks. Such attacks consist in exploiting information leaks from physical measures observed on embedded cryptographic systems. With the emergence of artificial intelligence, new side channel attacks based on deep learning techniques arose out. In many contexts, those techniques are more efficient in exploiting data leakages than classical ones. In this thesis, we study deep learning techniques in the context of having multiple sources of information available. In a first step, we investigated if the combination of multiple sources that are measured from the same setup/device could help an attacker to build a more effective attack compared to a single-source one. We have shown that it is not simple to combine sources to improve the efficiency of a neural network. In a second step, we investigated if the use of additional datasets that are not measured from the same setup/device could help an attacker to mount a stronger attack (or could decrease the evaluation time by helping the network convergence). We used a technique called transfer learning to mount this kind of attack. We have shown that transfer learning usually helps convergence and, in some cases, may improve the final efficiency of the neural network. Eventually, in the context where an attacker has access to two datasets leading to different attack efficiencies, we investigated whether it is possible to translate traces from the actual target (dataset that is harder to attack) into traces from the second dataset (which is easier to attack).