UPPSALA
UNIVERSITET

UPTEC F 22045
Examensarbete 30 hp
June 2022

An Efficiency Evaluation of Far-Field Electromagnetic Deep Learning Side-Channel Attacks in Controlled Environments

Gabriel Evensen

Civilingenjörsprogrammet i teknisk fysik

UPPSALA UNIVERSITET

An Efficiency Evaluation of Far Field Deep Learning Side-Channel Attacks in Controlled Environments

Gabriel Evensen

Abstract

As more and more modern systems and products use built-in microcontrollers, hardware security becomes more important to protect against cyber-attacks. Internet of things devices, like Bluetooth devices, usually use an encryption algorithm to keep data safe from hackers. Advanced Encryption Standard (AES) is a commonly used encryption algorithm. AES itself is hard to break. However, it is possible to utilize the information leaking from a system during the execution of encryption, called side-channel, to recover the key or part of the key used by the encryption algorithm. This kind of attack is called a side-channel attack (SCA).

In this study, two deep learning (DL) models are trained to attack the Bluetooth microcontroller unit Nordic nRF52 development kit equipped with an nRF52832 chip. The DL models are trained using the far-field electromagnetic emissions that the microcontroller unintentionally generates and transmits through the antenna while encrypting data. All encryptions are executed with a fixed key and random plaintext. The attack is conducted in two stages: the profiling and attack stages. In the profiling stage, where the attacker is assumed to have full system control, 100 000 traces holding encryption information are sampled and used to train the DL models to classify a sub-byte of the fixed key given a trace. In the attack stage, traces are captured in two different environments. The first is an entirely isolated environment, while the second adds a specific Wi-Fi access point and client connection that execute HTTP requests and responses in this isolated environment referred to as the system environment. Given traces obtained from one of the two attack environments, the performance of the trained models at classifying the correct sub-key is evaluated.

To summarize the results of this study, twelve SCAs are performed on six datasets captured in two different environments using two different DL models for each dataset. The correct key byte can be retrieved in three of these SCAs. All three successful attacks are made in an isolated environment without any interfering noise. The best performance is achieved with the multi-layer perceptron DL architecture, processing traces each composed of 10 averaged traces of the identical encryption, and the correct key-byte is recovered after 8198 traces.

Teknisk-naturvetenskapliga fakulteten

Uppsala universitet, Utgivningsort Uppsala

Handledare: Filip Björklund Ämnesgranskare: Roland Hostettler

Examinator: Tomas Nyberg

Populärvetenskaplig sammanfattning

I denna artikel implementeras ”far-field electromagnetic deep learning side-channel attacks” och effektiviteten i dessa attacker, som utförs i två olika miljöer, undersöks. Möjligheten att utföra denna typ av attack grundar sig på de fysiska elektromagnetiska vågor som bildas och läcker ut ur ett system medan data krypteras. Målet med en sido-kanalsattack är att hitta den hemliga nyckeln som används av algoritmen för att kryptera data. Angriparen använder en strategi som kallas ”dela och erövra” för att få fram delar av den fullständiga nyckel. Detta betyder att problemet delas upp i mindre delar, 16 delar i detta fall då dela nyckeln delas i 16 bitar. Under profileringsstadiet antas det att attackutövaren har full kontroll över enheten. I attackstadiet har motståndaren begränsad fysisk åtkomst till målenheten. Attackstegets insamlingsfas av det fysiska läckage av information, som uppstår då en enhet utför krypteringsalgoritmen, sker i två uppsättningar av miljöer. Den första är en helt isolerad miljö där endast enheten som utför krypteringen samt den utrustningen som används för att samla in data på de elektromagnetiska vågor som enhet läcker ut. I den andra miljön läggs en specifik Wi-Fi-åtkomstpunkt och klientanslutning till, där HTTP-förfrågningar och svar och sker kontinuerligt, i den tidigare isolerade miljö.
Med hjälp av en mjukvarudefinierad radio, som exempelvis går att beställa hem från Amazon, som heter HackRF One, samplas den sidokanal som läcker från profileringsenheten när AES-128-krypteringar utförs av enheten med jämna mellanrum. Den samplade signalen förbehandlas sedan och justeras så att varje segment av data insamlat representerar en liten del av den totala krypteringprocessen. Algoritmen körs med en slumpmässigt vald 128 bitars lång vektor av data, kallad ”plain-text” och med en slumpmässigt vald statisk hemlig nyckel, även den 128 bitar lång. Eftersom alla ”plain-texter” som används är kända för den som attackerar systemet, kan dessa vektorer av information användas för att bygga en ordnad datauppsättning. Denna datauppsättning används till att träna en djup-inlärningsmodell för att klassificera en del av den statiska nyckeln som används under profileringsfasen. Under attackfaserna tas insamlad data, som innehåller den information som läcker ut från enheten som krypterar data, från de två attackmiljöerna och ges till de tränade modellerna. En funktion används sedan för att ta reda på hur väl den tränade modellen kan klassificera rätt delnyckel från en uppsättning av ordnade data-vektorer som den aldrig tidigare har sett.
Endast i den isolerade attackmiljön kan modellerna framgångsrikt klassificera en del av nyckeln. När bakgrundsbrus läggs till i den andra miljön i attacksteget fungerar ”far-field electromagnetic deep learning side-channel attacks” mindre bra. Då nätverket inte har tränats med data som fångats i en brusig miljö blir det tydligt att det har stor effekt på hur väl sidokanals-attacker fungerar när de attackerar system i en brusig miljö. Om det skulle fungera att lägga till brus i ett system och använda detta som en möjlig motåtgärd till ”far-field electromagnetic deep learning side-channel attacks” eller om djup-inlärningsmodeller kan hantera detta är något som möjligtvis kan undersökas i framtida arbeten.

Acknowledgements

I am deeply grateful that I was able to do my thesis in collaboration with Knightec. I am grateful for the trust and dedication of the Knightec team.
I do like to express my sincere gratitude to Filip Björklund, my supervisor at Knightec. Your expertise, dedication and unwavering optimism have been invaluable throughout this project.
I do also like to thank Roland Hostettler, my subject reader from Uppsala University, who provided report writing advice and arranged access to the anechoic chamber.

Contents

1 Introduction

1.1 Background

Wireless protocol users are becoming increasingly familiar with the concept of data security and its growing demand. More and more sensitive data is being sent and received across various wireless protocols due to a comprehensive ongoing digital transformation. One of the most frequent ways to protect sensitive information is using encryption. There are a variety of encryption standards in place across the various wireless protocols to safeguard their users' privacy. However, new developing technologies and a broader understanding of cryptography have posed imminent threats to several previously considered safe encryption standards. A far field deep learning side-channel attack is the focus of this research.
Side-channel attacks (SCAs) exploit vulnerabilities in the implementation of algorithms on physical devices to extract its secret parameter, e.g. the secret key. This type of attack relies on information leaked by the device executing the algorithm, which we refer to as the side channel (SC). Some of the most common SCs used in SCAs are power consumption, timing, and electromagnetic (EM) emissions. SCA has become a powerful method for breaking cryptographic algorithms. In recent years many types of SCA exploiting different types of SCs have been utilized to break the cryptographic algorithm called Advanced Encryption Standard (AES) [1]. AES was announced by the U.S. National Institute of Standard and Technology in 2001 [2]. This algorithm is widely used in connected devices for data encryption and decryption. State of the art SCAs based on deep learning (DL) built on multi layer perceptron (MLP) and convolutional neural network (CNN) architecture have been proven to successfully extract secret key, or part of the key, from devices implementing the AES algorithm [3–8]. In a real world setting, the drawback of SCAs based on power traces or near field EM emission is that the acquisition of power traces or near field EM emissions requires the adversary to have close physical access to the targeted device. Conversely, an attack scenario, exploiting far field EM emissions, where the traces can be acquired at a distance becomes more realistic in a real world setting [6].
Recent research has demonstrated that several attacks against AES employing far-field EM emission as a SC can successfully recover the secret key, or a portion of it, from Bluetooth devices [4–6]. Although this does not necessarily guarantee the key's successful retrieval. However, new research indicates that the effectiveness of these types of attacks has improved [7, 8]. During the production of the ciphertext, the SC leakage arises as EM emissions created by the switching activity of the logic components of the targeted device's chip. The leaked information from the cryptographical process couples with the digital signal representing the ciphertext and transfers through the bus to the analog part, where it is converted to an analog signal by the digital to analog converter (DAC). The radio frequency (RF) block and voltage-controlled oscillator (VCO) modulate the analog signal to a frequency defined by the wireless transmission protocol and transmit it through the antenna, enabling an adversary to sample this leaked information at a greater distance [8].

1.2 Purpose

Any connected device executing a cryptographic operation may produce EM waves as a SC. Anyone with sufficient technological knowledge and the appropriate equipment may intercept this SC, providing a major risk of security breaches. DL has emerged as a powerful tool in the development of the most recent state-of-the-art SCAs. According to new findings, DL-SCAs require less traces from the target device than other types of attacks. Additionally, this approach is capable of bypassing a variety of current countermeasures such as jittering and masking [6].
Given that significant resources are being invested in DL, it is realistic to expect that these techniques will continue to develop and grow in power in the future. Therefore, raising awareness of DL-based SCAs is crucial in order to assist the development of relevant countermeasures.
The goal of this study is to determine if it is possible to conduct far field EM SCA on a system of connected devices. The system that will be investigated in this study is presented in Figure 1.

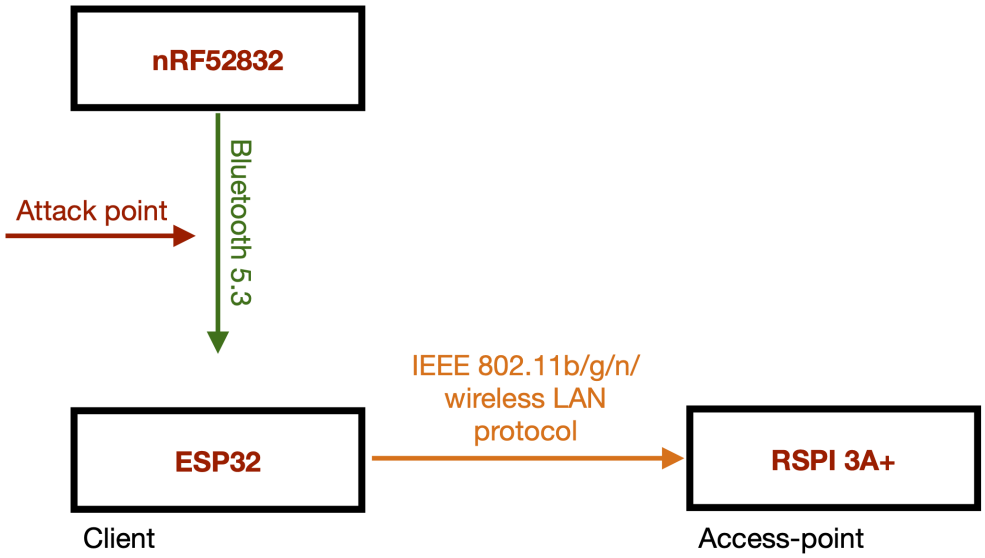


Figure 1: IoT-system setting equipped with Bluetooth 5.3 and IEEE 802.11b/g/n/ wireless LAN protocol. The target device is a nRF5283 MCU running AES.

The attack point in this system setting is the Bluetooth Low Energy (BLE) and Bluetooth mesh development kit equipped with the nRF52832 BLE chip by Nordic Semiconductor. The nRF52832 BLE chip will be running an implementation of AES with a fixed key and random plaintext as input. The antenna must be turned on to ensure that the SC is exposed.
The ESP32-DevKitC-32UE micro controller unit (MCU) will be connected to a Raspberry PI 3a+ acting as an access point and logging data over a IEEE 802.11b/g/n wireless protocol simultaneously as the Nordic MCU is performing the encryption. The Nordic

development kit will be implementing AES encryption and the information will leak out through the antenna.

As noted in Section 1.1, prior research has demonstrated that it is possible to retrieve the secret key from SCs created by MCUs employing a Bluetooth wireless protocol. However, there are fewer studies on the effect of EM noise on the efficiency of SCAs. In an office setting, for instance, there are typically multiple wireless protocols operating in the same area; nevertheless, this noise is not considered in earlier studies.
Given that Wi-Fi devices operate in the same frequency range as Bluetooth devices (2.4GHz), we want to determine if the Nordic MCU devices are similarly susceptible to SCAs when attacked in an environment of other connected devices, using the same techniques as the recently discovered Bluetooth far field EM emission attacks. The attack stage will be conducted in two phases; the first with an isolated Nordic MCU and the second with Nordic MCU in a system of connected devices. All experiments will be conducted in an anechoic chamber, creating a completely isolated environment devoid of any unidentified EM noise sources that could impact the resulting efficiency of the SCA.

1.3 Aim

The thesis address the following research question:

- How efficient are DL SCAs in retrieving a part of the secret key in a simple IoT-system setting equipped with Bluetooth 5.3 and IEEE 802.11b/g/n wireless LAN protocol?

The efficiency of an attack will be defined in this paper as the minimum size of the attack set traces needed on average to ensure that the attack succeeds in ranking the correct key in first place among the 256 likely ones. The metric used is the rank function and the efficiency evaluation methodology is presented in Section 2.5.5.

2 Theoretical Background

In this section, the theoretical aspects of this work are presented. First, AES and the algorithm that forms the basis of the encryption standard will be introduced as the first theoretical component. Secondly, the concept of SCAs and the various SCA types will be discussed. In addition, the specific type of SCA employed in this study and the mechanism by which EM emission arises in MCUs will be described. The theory underlying correlation analysis is then introduced. In the final section, the DL part of this study, the architectures that will be used, and the training and SCA efficiency metric will be described.

2.1 Advanced Encryption Standard

Encryption standards such as the block cipher, AES, are broadly applied in wireless encryption protocols. A block cipher is a function which maps plaintext bits of a certain block length into cipher-text of the same length and the function is parameterized by a key. The standard was announced by the National Institute of Standard and Technology in 2001 [2].
AES algorithm originates from the Rijndael algorithm, which was the winner in the contest for a new AES organized by National Institute for Standards and Technology (NIST) in the beginning of January 1997. When developing the winner of the new AES algorithm there were some modifications made to the Rijndael algorithm in order to define the final AES version. The difference between Rijndael and AES lies in the range and supported values for block length and cipher key length. Rijndael algorithm is design with both a variable block length and a variable block cipher key length. The variables are independently specified and restricted to be any multiple of 32 bits within the range of 128 bits and 256 bits. The AES algorithm was designed to be limited to 128-bits plain-text as input and encrypt it to a 128-bits cipher-text using a key of either 128, 192 or 256 bits length. Depending on the length of the key the algorithm will be refereed to as either AES-128, AES-192 or AES-256 [9].

2.1.1 Algorithm Specification

AES is designed to operate in steps called round transformations. In the rounds of the AES, the algorithm operates on an intermediate result refereed to as the state. Depending solely on the length of the key, since the block length is fixed, a number of rounds will be executed when creating the cipher. AES-128, AES-192 and AES-256 have 10, 12 and 14 rounds respectively. The round function is composed of byte-oriented transformations. In every round except the final one, where Mixcolumns is omitted, the intermediate state undergoes four byte-oriented transformations in the following order: **SubBytes**, **ShiftRow**, **MixColumns** and **AddRoundKey**. Each transformation of the state which will be explained in detail below.

Figure 2 illustrates the encryption scheme with the structure of the byte-wise transformation in the 10 rounds of AES-128 encryption:

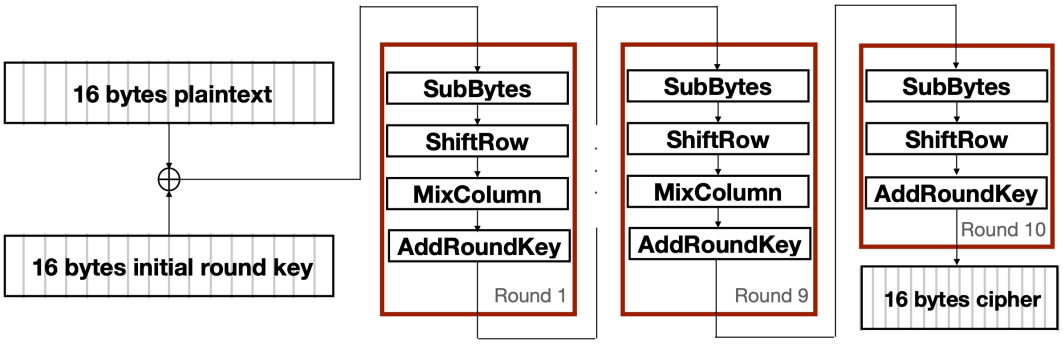


Figure 2: Demonstration of the 10 rounds of AES-128 encryption scheme.

Before the 10 rounds start, the AES algorithm performs a key expansion routine using the cipher key, denoted K, based on Rijndael's key schedule. The key expansion routine generates a linear array of 4-byte words with the length of 44 for AES-128, denoted w. This array holds the first 4-byte words of the initial round key for *AddRoundKey* called before the first round. The initial round key generated by Rijndael's key schedule is equal to the secret key that is given to the AES algorithm. The remaining 40 4-byte words hold the round key data of the following 14 rounds. With an intuitive understanding of the AES-128's general structure, we will further discuss the four byte-oriented transformations in AES presented in [2].

SubBytes is a non-linear independent substitution of each byte in the state array using a substitution table called the S-box. This invertible substitution table is constructed to substitute two transformations that occur in SubBytes. In matrix form, the affine byte transformations done by using the S-Box lookup table can be expressed as

$$\begin{bmatrix} y'_0 \\ y'_1 \\ y'_2 \\ y'_3 \\ y'_4 \\ y'_5 \\ y'_6 \\ y'_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \tag{1}$$

where y_i is the i_{th} bit of the byte for $0 \leq \mathrm{i} \leq 7$ and y'_i represents the updated byte. The S-box lookup table is used to speed up the transformation. As it is shown in Figure 3, the S-Box table is used to substitute each byte in the state array.

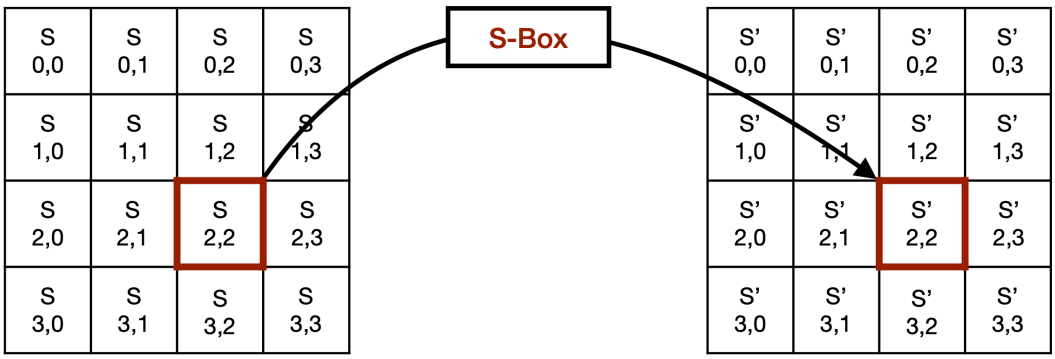


Figure 3: Illustrating how a S-Box lookup table function is used to update the state in SubBytes.

ShiftRow is a transformation where the last three rows in the 4×4 16-bytes matrix are shifted over a certain number of bytes called an offset. The aim of the transformation is to increase the diffusion of the cipher. The first row stays the same without modification. The second, third, and fourth rows are correspondingly shifted to the left by one, two, and three bytes. The cyclic shifts of the last three rows in the state are demonstrated in Figure 4.

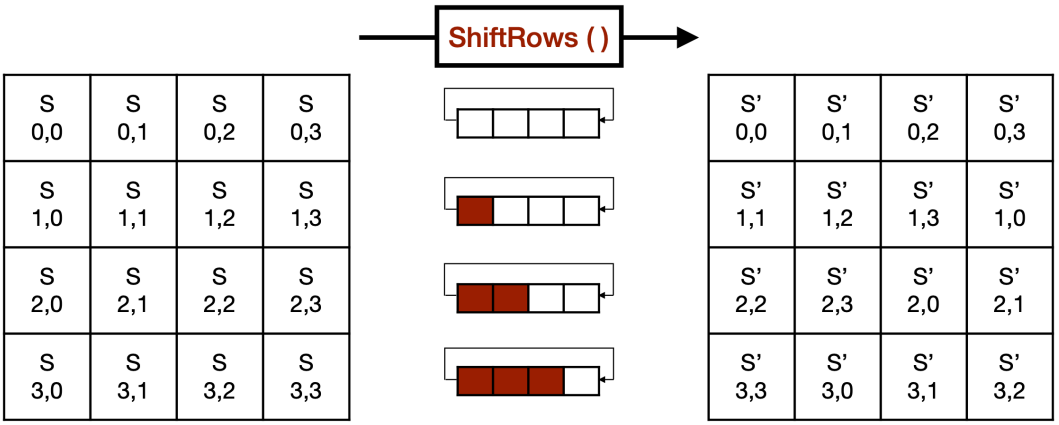


Figure 4: Illustrating how ShiftRow is vertically shifting the last three rows of the state matrix.

MixColumns, similarly to *ShiftRows*, increases the diffusion of the cipher. Diffusion implies that if a single character in the plaintext is modified, several characters in the ciphertext should also be modified. This transformation executes column wise, treating each column as a four term polynomial. Each column transformation can be expressed as a matrix multiplication

$$\begin{bmatrix} s'_{0,j} \\ s'_{1,j} \\ s'_{2,j} \\ s'_{3,j} \end{bmatrix} = \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix} \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix}, \quad 0 \leq j < 4, \tag{2}$$

where each column vector is multiplied over $GF(2^8)$ with a 4×4 matrix.
An illustration of the *MixColumns* operation can be seen in Figure 5, transforming the State column by column.

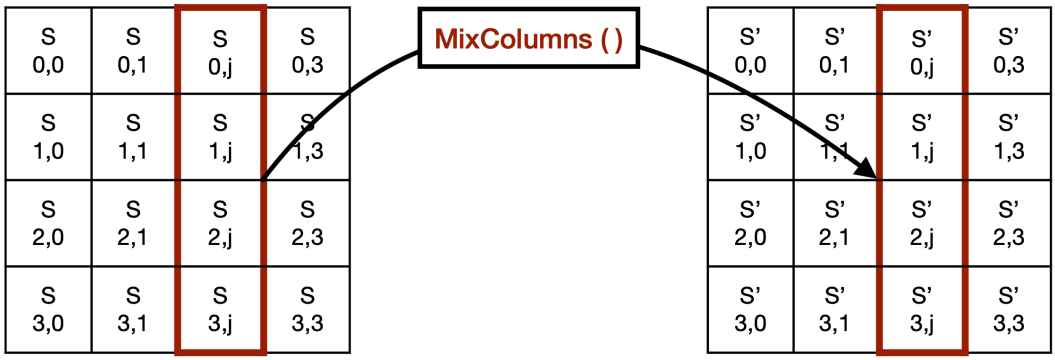


Figure 5: Illustrating how MixColumns executes column wise on the state matrix.

AddRoundKey is the last step in each round of the AES algorithm. This transformation consist of the bit-wise operation of every column in the State matrix with the specified part of the round key data. As mentioned earlier, the round key data generated by the key expansion routine consist of 44 4-byte words. The first AddRoundKey transformation occurs before the round function is applied, where the initial round key holds the 16 bytes of the secret key passed to the AES algorithm. In the following 10 rounds, AddRoundKey XORs each column of the state with a 4-byte word from the key schedule. The words from the key schedule are each added into the columns of the state according to

$$\begin{bmatrix} s'_{0,j} & s'_{1,j} & s'_{2,j} & s'_{3,j} \end{bmatrix} = \begin{bmatrix} s_{0,j} \\ s_{1,j} \\ s_{2,j} \\ s_{3,j} \end{bmatrix} \oplus \begin{bmatrix} w_{round*N_b+j} \end{bmatrix}, \quad 0 \leq j < 4, \quad k = round \cdot N_b, \tag{3}$$

where N_b represent the word from the key schedule and j the column in the state.
The illustration of *AddRoundKey* is presented in Figure 6, where w_i is the vector holding the 44 words created in the key expansion routine as mentioned in the beginning of this section.

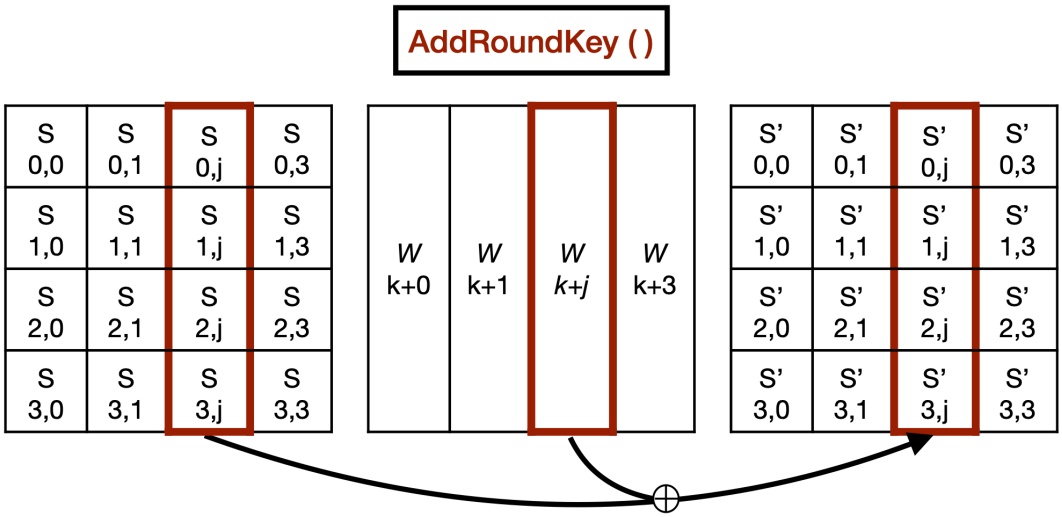


Figure 6: Illustrating how AddRoundKey operating column wise on the state matrix using the words generated in the key expansion routine.

The summarized AES-128 algorithm are presented in pseudo code in Algorithm 1 below.

Algorithm 1 AES-128 pseudo code

$N_r = 10$
$N_b = 4$

$state = plaintext$

$AddRoundKey(state, w[0, N_b - 1])$ ▷ $len(w) = 44$ words from key expansion routine.

for round $= 1$ to $N_r - 1$ **do**
    $SubBytes(state)$
    $ShiftRows(state)$
    $MixColumns(state)$
    $AddRoundKey(state, w[round \cdot N_b, (round + 1) \cdot N_b - 1])$
end for

$SubBytes(state)$
$ShiftRows(state)$
$AddRoundKey(state, w[N_r \cdot N_b, (N_r + 1)N_b - 1])$

$ciphertext = state$

2.1.2 Modes of Operation

A mode of operation defines how to use the operation of a block cipher repeatedly to securely encrypt data larger than a block.
A mode of operation defines how a block cipher can be employed to transform data that are larger than a single block. There are different modes of operation for the symmetric key block cipher algorithm AES. The National Institute of Standards and Technology (NIST) recommends a total of five methods of operation [10]. Specifically, the following five:

- Electronic Codebook (ECB)
- Cipher Block Chaining
- Cipher Feedback
- Output Feedback
- Counter

All these modes can, according to NIST, provide data confidentiality if its used in conjunction with the AES. The single most important mode of operation for the understanding of this work is the ECB mode. Therefore, this mode of operation and its functioning will be discussed in further detail:

ECB is the least complex mode of operation between the five. This mode divides the massage into blocks of 128-bits and encrypts the bytes independently. Both the forward and inverse cipher function is applied directly to the plaintext and ciphertext respectively, as shown in Figure 7. The mode is defined as in equations (4) and (5),

$$Encryption : C_j = CIPH_K(P_j) \ \ for\, j = 1...n, \tag{4}$$

$$Decryption : P_j = CIPH_K(C_j) \ \ for\, j = 1...n, \tag{5}$$

respectively, where j is defined as the j-th 128 bit block of data and K the secret key.

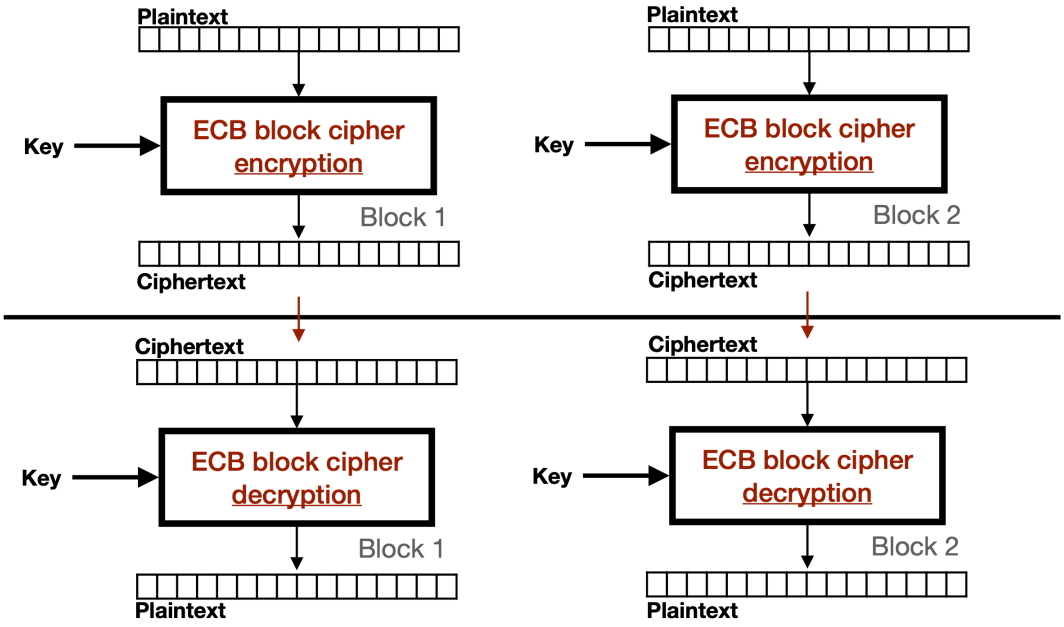


Figure 7: Encryption and decryption of two 128-bit blocks using the Electronic Code Book (ECB) mode of operation.

In this mode of operation, under a fixed key, any given block of plaintext always gets encrypted into the same cipher block and thus the mode has a lack of diffusion of the data being processed.

2.2 Side-Channel Attacks

There are several methods for attacking cryptographical systems. Common attacks include repeatedly attempting different keys, attacking the cryptographic algorithm itself or attacking the software executing the algorithm.
One example is the Brute Force Attack. It is an ineffective method for cracking AES-128 encryption. This strategy is based on guessing the secret key by attempting all feasible combinations. This approach is inefficient due to the fact that AES-128 has 2^{128} possible combinations of the key. It would take too long and need too much processing power for modern computers to test all these possible key combinations.
Crypto-analysis is when the adversary are evaluating the system and its implementation to discover concealed vulnerabilities or take control of the crypto system. For example, mathematical flaws in cryptography can be exploited to compromise encryption systems. This is why encryption algorithms should always be publicized, so that researchers discover the vulnerabilities before other adversaries that may be exploiting these for a bad cause [3].
SCAs are a subcategory of crypto-analysis. Rather than focusing on the cryptographic algorithm itself, these attacks rely on information leaking from a system during the en-

cryption process. The objective of SCA is to recover the secret key or portions of the key of a cryptographic algorithm. Usually, the adversary employs a divide-and-conquer technique, which entails attempting to recover subkeys and combining the result in order to recover the full key. With a known set of plaintexts P, the adversary collects a set T of physical measurements, e.g. power consumption, EM emission or timing measurements. Each trace corresponds to a a physical measurement of the system processing a known plaintext with the key K to create the ciphertext.

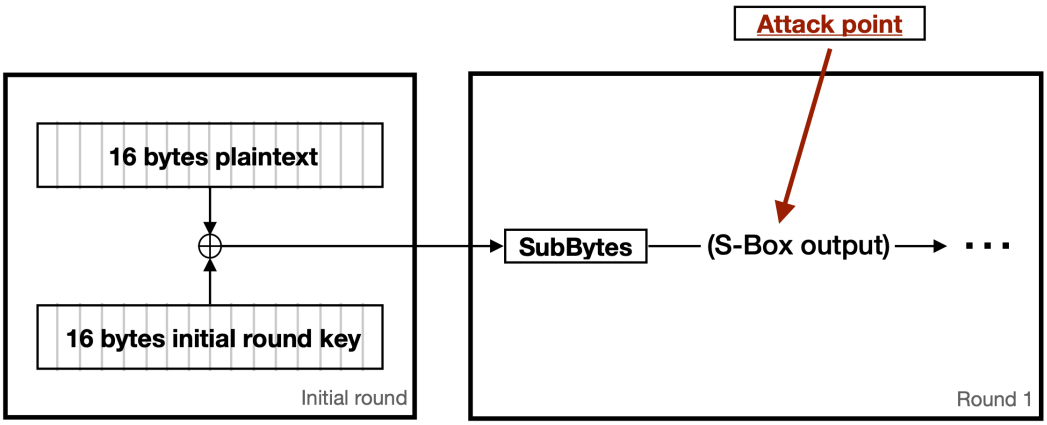


Figure 8: The attack point of the SCA is the byte oriented transformation *SubBytes* in the first round of AES.

The attack point of the SCA performed in this project is visualized in Figure 8. For AES-128 the 16 bytes of the key K are divided into 16 subkeys k, each subkey consists of 1 byte. Every subkey K_k for $k \in \{1, 2, ..., 16\}$ is then independently recovered [6]. As explained in Section 2.1, the byte-oriented transformation *SubBytes* operates on one byte at a time. To speed up the operation, a lookup table called the S-Box is used. In the first round of AES-128, the plaintext and the secret key, both having a size of 16 bytes, are XORed byte-wise and the result of this operation is used to substitute the state array byte-wise using the lookup table. In a scenario when the plaintext and fixed secret key are known to the adversary, such as during the profiling phase, the S-Box output can be calculated. With both the plaintext and encryption key known, it is possible to label every trace with its corresponding S-Box output given the plaintext and key used in the encryption process. The lookup table consists of 256 entries corresponding to all possible combinations of a single byte. Each operation in the lookup table has a characteristic pattern in the physical trace.
Since the real key is used in the first round of AES-128, the plaintext is known and a correlation between the S-Box output and the trace can be made, it is possible to derive the sub-key that was used in a specific S-Box operation.

2.3 Electromagnetic Side-Channel Emissions

When talking about SC leakage of EM emissions, they can be divided into two categories, namely direct and indirect EM emissions. The former category of EM emission is investigated in this project. The indirect EM emissions arises from the coupling effect between the different components of the MCU chip. The encryption data stream is generated on the digital side of the chip where the bit flips are controlled by the system clock of the CPU. This data stream is then transferred into the bus and in to DAC. The analog signal is transferred to the RF block where the signal is modulated to a specified frequency of the transmission protocol and transmitted through the antenna. Figure 9 illustrates how the analog component adds the digital circuit's noise during AES-128 execution to the radio signal.

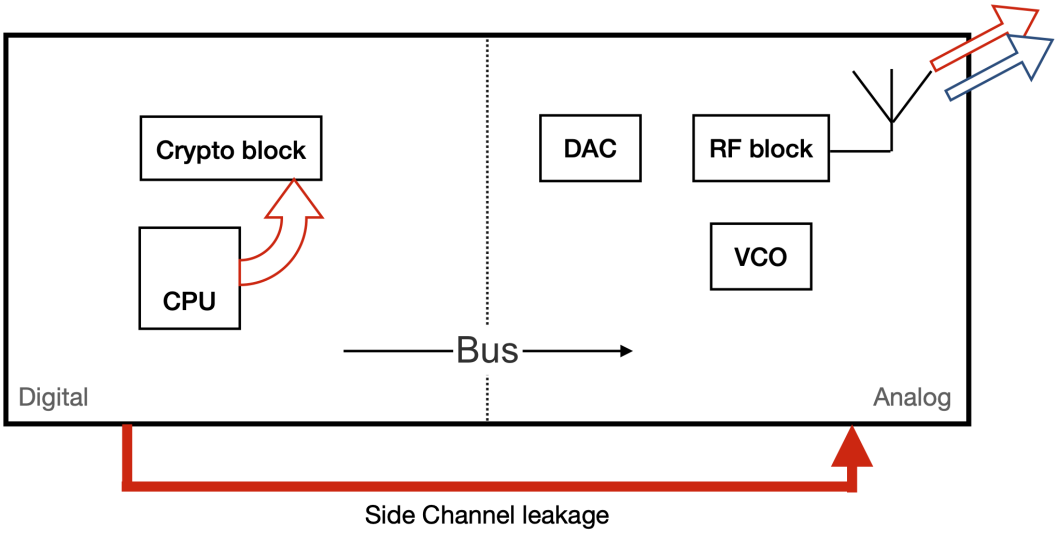


Figure 9: Digital noise arising and transferred from the digital part to the analog part and transmitted by the antenna. The red arrow from the antenna shows the leakage while the blue arrow shows the intended signal.

The CPU clock signal generates digital noise due to the bit flip operations where 0 represents the low-level current and 1 represents the high-level current. The operations of the encryption are amplitude modulated (AM) by the square wave noise coming from the switching activities of the CPU on the digital part and then interact with the bus which transfers this modulated noise into the DAC on the analog part. The RF block and VCO on the analog part modulate and amplifies this digital noise and send it out through the antenna. This makes it possible for an adversary to detect this digital noise that contains the information about the encryption process and enables the adversary to detect the signal from a large distance of the device [11].

2.4 Correlation Analysis

The collection of non-profiling SCAs, which consists of differential power analysis and correlation power analysis (CPA), corresponds to an adversary with only physical leakage information from the target device. A CPA attack comprises four phases: In the first phase, the attacker must create a model of the victim device's physical output. Here, the model will concentrate on a particular part of the encryption algorithm, such as the S-Box output in the first round of AES-128. The second phase involves forcing the victim device to encrypt several known plaintexts and capturing the energy traces emitted by the victim device during each encryption. In the third phase, a divide-and-conquer strategy is employed, and subparts of the entire key are attacked. For every possible combination of each subkey, the model from the first phase is used to calculate the output, given the known plaintexts and every possible subkey guess. The model output for each possible subkey guess can be compared to the actual physical measurement using Pearson's correlation coefficient. The subkey with the highest correlation is selected. In the fourth phase, the best subkey guesses are combined to generate the full key [12].

The Hamming weight or Hamming distance

$$HammingWeight(x \oplus y) = HammingDistance(x, y) \tag{6}$$

are common models used in CPA for SCAs. The Hamming weight is the number of ones in a binary number and the Hamming distance is the number of distinct bits between two binary strings. For AES-128, both the key and the plaintext contain 16 bytes. Every subbyte in the key has 256 possible combinations. For every plaintext byte encrypted by the encryption algorithm, there are 256 possible subkeys that could have been used. The number of plaintexts are given by P_n where $n = \{1, 2, ..., N\}$ and the number of possible single key-byte combinations are given by K_j where $j = \{1, 2, ..., J\}$. Independently whether the Hamming weight or Hamming distance model are used, the first phase of CPA generates a matrix of size $N \times J$

$$F = \begin{bmatrix} f_{1,1} & f_{1,2} & \cdots & f_{1,J} \\ f_{2,1} & f_{2,2} & \cdots & f_{2,J} \\ \vdots & \vdots & \ddots & \vdots \\ f_{N,1} & f_{N,2} & \cdots & f_{N,J} \end{bmatrix}, \tag{7}$$

where every element $x_{n,j}$ are the n-th estimated model output using a specific subkey guess j. Given a fixed plaintext for every row and every possible combination of one key-byte in the columns, the element of matrix F are the output of the model from the fist phase of CPA. From the second phase of CPA, the physical traces are stored in a matrix

$$T = \begin{bmatrix} t_{1,1} & t_{1,2} & \cdots & t_{1,M} \\ t_{2,1} & t_{2,2} & \cdots & t_{2,M} \\ \vdots & \vdots & \ddots & \vdots \\ t_{N,1} & t_{N,2} & \cdots & t_{N,M} \end{bmatrix} \tag{8}$$

with the size $N \times M$ where M are the number of datapoints in a single physical trace. Each row in T represents a physical trace of the encryption process of a unique plaintext, while each column represents the data points in the trace.
Finally the Pearson correlation coefficient ρ is used to find the most likely subkey, which is given by the columns with the highest correlation and where the operation is located in the physical trace [5].

$$\rho_{J,M} = \frac{\sum_{i=1}^{N}[(F_j[i] - \overline{F_j})(T_m[i] - \overline{T_m})]}{\sqrt{(\sum_{i=1}^{N}[(F_j[i] - \overline{F_j})^2][(\sum_{i=1}^{N}(T_m[i] - \overline{T_m})^2)]}}, \; j \in J, \; m \in M. \tag{9}$$

2.5 Deep Learning in Side-Channel Analysis

The type of SCA performed in this project is a profiled SCA and DL-SCA belong to this category. In this sort of attacks the adversary is assumed to have full control over the device in which we refer to as the profiling device. From this profiling device the attacker can create a leakage profile. The leakage profile is then used to recover the key from the device under the attack stage. In order to conduct a profiling attack, there are some assumption that have to be made of the adversary, which are the following:

- The adversary has access to a profiling device that can be utilized to produce the leaking profile.
- The adversary possesses total system control of the profiling device.
- The adversary temporarily possesses physical access to the target device.

SCA can be seen as a classification problem where the objective is to correctly classify the key-byte of interest in the key. One byte consists of eight bits and there are 256 possible combinations of bits in this single byte. This results in a classification problem of 256 classes in total.
In the profiling stage, a DL model is trained to classify the 256 possible outputs from the S-Box operation in the first round of AES-128. Here, the neural network processes traces from the set T_p of traces captured in the profiling stage in which the attacker has full control over the device. Each trace is given a label, consisting of the S-Box output of a specific plaintext-byte and key-byte in the first round of AES-128. The objective is to train the weights in the network in order to be able to to classify the trace to its corresponding label. In the attack stage, the model is fed with traces from the set of traces T_a captured in

the attack stage, where the labels are unknown. There are different ways of combining the fundamental component in DL neural networks, the neurons, into different structures that are referred to as DL architecture. In the two following sections, two of the most common DL architectures, used in SCA to solve this classification problem, will be presented.

2.5.1 Neural Networks

Inspiration for the concept of a neural network, which analyzes data in order to discover hidden patterns, came from the human brain. The single neuron, which is the fundamental component of a neural network, attempts to imitate the function of its biological counterpart in the human brain [13]. An illustration of a single neuron is shown in Figure 10.

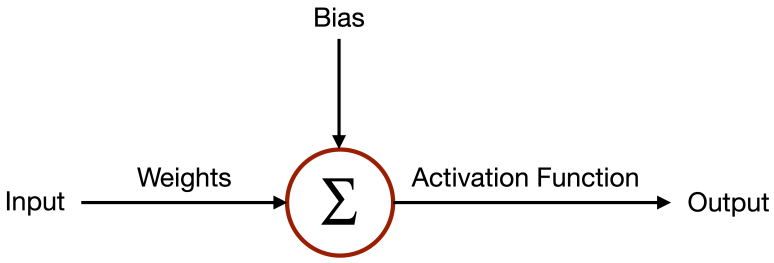


Figure 10: The structure of a single neuron.

The neuron's input consists of two components. The first component consists of inputs with corresponding weights, whereas the second represents the bias. The neuron sums the corresponding parts of the input and passes the result to the activation function, which generates the neurons output. In addition to an input and output layer, a neural network has one or more hidden layers. Every neuron in the hidden layer has a nonlinear activation function that modifies its output. The weight and bias are parameters that can be modified to alter the neural networks output. The network iteratively modifies the control parameters until the predictions are sufficiently accurate or another stopping threshold is achieved [14].
This work employs two different kinds of activation functions. Rectified Linear Unit (ReLU) is the activation function of every neuron whose activation is not in the output layer. ReLU is a nonlinear activation function that returns its input value if the input is equal to or greater than zero and zero otherwise [15]. The neurons in the output layer have Softmax as its activation function and provide the probabilities for each class label present in the classification problem [16]. Back propagation is the essence of neural network training. It is the process of adjusting the network's weights based on the error rate from the previous iteration. This is performed until the accuracy is sufficient or another stopping criterion is met. The gradient-based optimization technique used to train the neural networks in this study is RMSprop. Learning rate is a model hyper-parameter that specifies the amount

by which the optimizer used to train the model should modify the control parameters each time they are updated. RMSprop employs an adaptive learning rate rather than treating it as a hyper-parameter, meaning that the rate of learning changes throughout the training phase [17].

2.5.2 Multi-Layer Perceptron

The characteristics of an multi-layer perceptron (MLP) architecture is that: it has an input layer, one or more hidden layers and an output layer. Every layer consists of one or more neurons. These neurons will from now on be referred to as nodes. The size of the input layer completely depends on the dimensionality of the input data. Every node that is not an input node is associated with a non-linear activation function. The size of the output layer is determined by the number of categories in the classification problem. Every layer, except for the input layer, are fully connected meaning that every node in these layers is connected to all the nodes in the adjacent layer [14]. An example of MLP architecture is illustrated in Figure 11

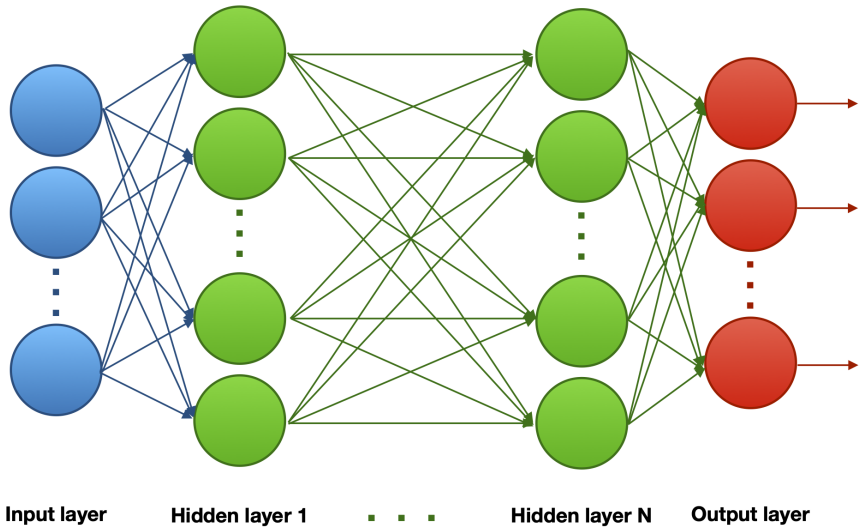


Figure 11: MLP architecture with an input layer, N hidden layers, and an output layer.

Every node in the input layer represents a data point from a trace. Thus, the length of traces determines the size of the input layer. In the SCA setting of this study, the last layer is the output layer and has Softmax as its activation function. This SoftMax layer calculates the probabilities of every possible S-box operation. With a known plaintext as input to the AES algorithm, the corresponding key byte can be derived. Every node in a hidden layer have weights and a bias associated with it. The weights and the biases are the trainable parameters of the network in which the model can modify in order to alter the output. The model adjusts these weights iteratively until a certain stop criterion. Mathematically, the MLP architecture model function $\hat{g}$, composed of multiple linear functions and non-linear activation functions and can be described by

$$\hat{g}(\vec{t}) = s \circ H_n \circ \alpha_{n-1} \circ H_{n-1} \circ \cdots \circ H_1(\vec{t}), \tag{10}$$

where

- H_i are the hidden layers expressed as affine functions, a linear function in addition with constants, $A\vec{t} + \vec{B}$. The matrix A are holding the weights, $\vec{t}$ are the input trace data and $\vec{B}$ are a vector holding the biases,
- α_i is the activation function of layer H_n for $n = \{1, 2, ..., N\}$, where N are the number of hidden layers, and
- s is the SoftMax function which will be the activation function present in the output layer to calculate the probabilities of the possible S-box operations [1].

2.5.3 Convolutional Neural Network

Along with the fully connected layer, the convolutional neural network (CNN) architecture introduces two more layer types: convolutional layer and pooling layer [18].
A convolutional layer acts as a feature extractor in the sense that it preserves the relationship between adjacent data samples of the input. To explain the functioning of a convolutional layer, an illustration of convolutional filtering is presented in Figure 12 with input trace defined as f, convolutional filter g and filtered traces as h.

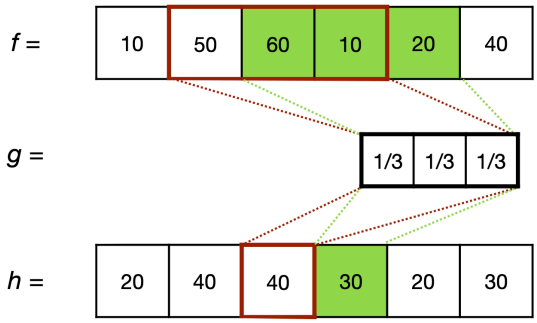


Figure 12: Convolutional filtering with $W = 3$, $n_{filter} = 1$, $stride = 1$ and $padding = same$.

To apply the convolution to the input data, the convolutional filter, with a given kernel W and column length n_{filter} slides over the input data by a unit called stride. The stride determines how the convolutional filter is sliding across the input. The input data can be padded with zeros for the resulting output vector after convolutional filtering to have the same number of points as the input data. When the input and output are equal after convolutional filtering, the padding is referred to as "same" padding [19].

The purpose of the pooling layer is to reduce the complexity of the deep learning model. The partial input to the pooling function is the segment which is defined by a stride

and a size. The size determines how many of the input elements are provided to the pooling function while the stride determines how the pooling filter is sliding across the input. Similar to the convolutional layer, the pooling layer slides a filter across the input. In contrary to the convolutional layer the pooling layer does not have trainable weights, instead it slides across the input layer and applies a pooling function to the input. Two common pooling functions are the the max pooling and average pooling. The first outputs the maximum value within the segment, while the later outputs the average of the the coordinates of the segment. An example of average pooling is presented in Figure 12 where h represents the data before average pooling and p after average pooling [20].

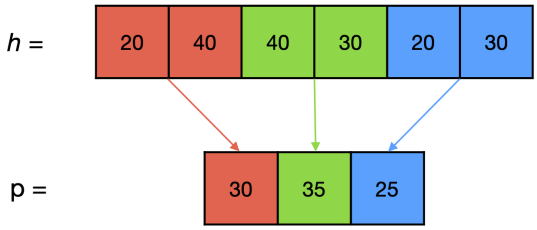


Figure 13: Average pooling layer with $size = 2$ and $stride = 2$.

Prior research suggests that the 16-layer VGG-16 network is a good starting point when deciding on a CNN architecture due to its good performance in SCA [1]. Figure 14 illustrates the VGG-16 architecture, which was first introduced in 2014 for use in image recognition [21].

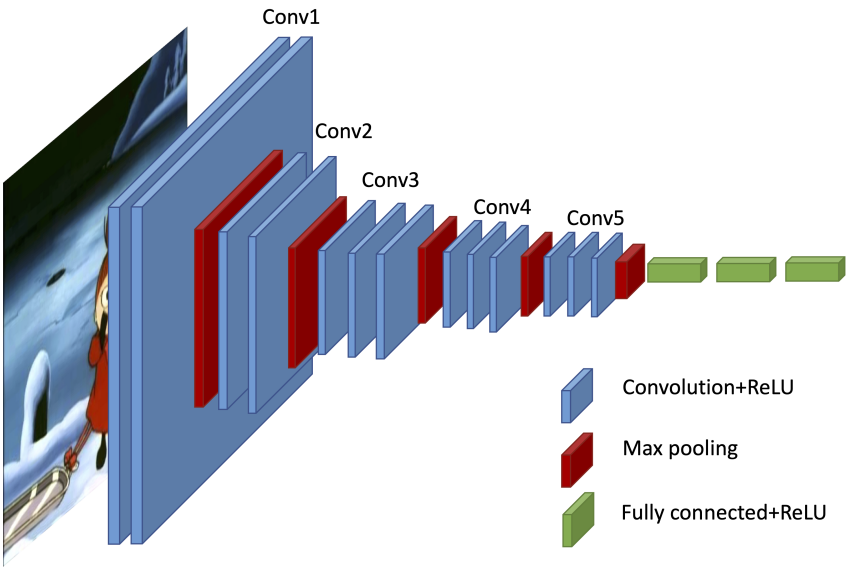


Figure 14: The VGG-16 architecture working as baseline for CNN architecture in a SCA setting.

2.5.4 Training of the Deep Learning Model

In the training phase of the DL model we tune the trainable weights of the model architecture. It is an iterative approach with the aim of minimizing the loss function that quantifies the classification error across a training set that in a SCA setting will be the dataset built from traces captured in the profiling stage.
This study investigates one hyper-parameter, epochs, when training the weights of the neural network model. Epochs determines the number of times the entire training data set is processed by the model. After one epoch in the training session has passed, every trace in the training data set has affected the choice of the updated trainable parameters in the model. For example, assume the entire training data set consists of 100 000 traces and the batch size is set to 1000. Before 1 epoch is complete, the model will process 1000 input traces at a time, update its internal weights, and repeat this process 100 times.

2.5.5 Efficiency Evaluation Method for Deep Learning Side-Channel Attacks

In this classification problem, there are 256 possible key-bytes for each single byte in the 16 byte long key. In order to determine the efficiency of the model trained in the profiling stage, one wants to know how many traces are needed in the attack stage in order to recover the correct key-byte. For every trace processed by the model, the model provides a probability of every possible key-byte. Given that K is the set of all possible combinations of key-bytes k, T is the set of traces captured in the attack stage during the encryption of plaintexts in the set P, the rank function

$$\text{rank}(\hat{g}, D_{attack}, n) = |\{K^* \in K : P[K^*|T, P] < P[K|T, P]\}|, \tag{11}$$

then provides the number of keys-bytes in K with a probability greater then the real key byte K^*.
where $\hat{g}$ corresponds to the model trained in the profiling stage, D_{attack} is the dataset collected in the attack stage and contains traces and their corresponding keys and plaintexts, and n is the number of traces to calculate the rank for.
To determine the average number of traces required to recover the right key-byte, the attack datasets can be permuted and the point where the rank reaches 0 in the majority of test sets can be calculated.

$$\text{mean}(\text{rank}(\hat{g}, D_{attack}, n)) = \frac{\sum_{i=1}^{n_p} rank(\hat{g}, D_{attack,i}, n)}{n_p} \leq 0.5, \tag{12}$$

where $D_{attack,i}$ represents a randomly permuted dataset generated from the original D_{attack} dataset and n_p is the number of times the rank test will be conducted with a permuted attack dataset [1].

3 Methodology and Implementation

This project follows the method presented in [6] and [11] but is implemented on the system presented in Figure 1, where the Nordic MCU are in a system of connected devices. A different setup of equipment, seen in Table 1, are used compared to the two earlier studies.

Table 1: Setup of hardware and components used in the implementation of this work.

Category	Equipment
Transmitting	Nordic nRF52 DK mounted with Bluetooth Chip nRF52832. Alfa 2.4GHz/5GHz Antenna ARS-25-57A.
Receiving	HackRF One SDR. Alfa 2.4GHz/5GHz Antenna ARS-25-57A.
Connected devices	ESP32-DEVKITC-32UE. Raspberry Pi 3 Model A+.
Other	Coaxial cable. RF Attenuator 20dB.

The MCU under attack periodically encrypt 128 bits of data using a AES-128 implementation. The data transmission sent by the radio frequency block on the connected device node consists in this case of the two major parts.

1. The cipher-text with the central frequency $f_{transmission}$ protocol defined by the wireless transmission protocol.
2. The modulated SC leakage with central frequency given by $2f_{clock} + f_{transmission}$ protocol, where f_{clock} is the frequency of the CPU clock on the MCU.

The SC leakage that occurs when the cryptographic block encrypts data into ciphertext contains the information required to recover a portion of the secret key during Bluetooth client-to-client communications. In addition, all experiments will be conducted in an anechoic chamber, which will create an environment devoid of any interfering EM noise.

3.1 Profiling Stage

The aim of this stage is to collect traces and characterize the physical leakage of the SC leaking from the target device, the device in which the attacker has full control over in the profiling stage.
The experimental setting in setup 1 is visualized in Figure 15.

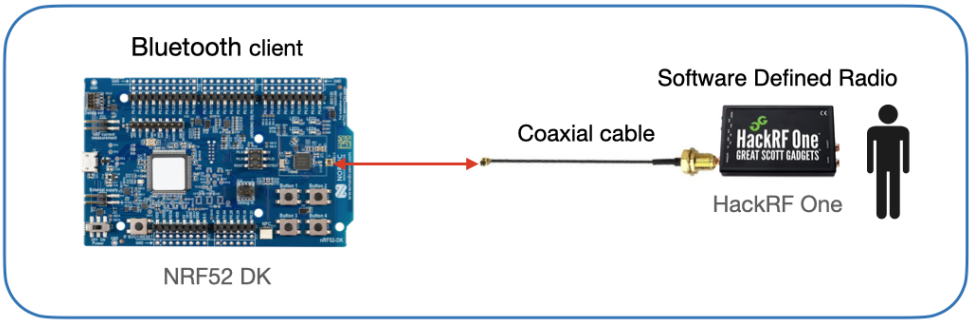


Figure 15: Experimental setup to profile the EM SC leakage when running the AES algorithm on the device.

The antenna is leaking information when the module is executing the cryptographic algorithm. The antenna transmits the SC from the antenna that is turned on, which it will be in all other experiments as well. The device execute encryption of AES-128, periodically with a fixed 128-bits key and randomly generated plaintext as input. A Software Defined Radio (SDR) mounted with a coaxial cable is used to receive and sample the SC information. The coaxial cable is directly connected to the external antenna connector on the nRF52832 Bluetooth client device to minimize the noise in the sampled EM signal. To not exceed the maximum input power on the receiver side, an attenuator of 20 dB is mounted between the transmitter and receiver.

3.2 Attack Stage

In a real life SCA scenario, the adversary does not have physical access to the target device. Therefore, the direct connection between the external antenna connector on the nRF52832 device can no longer be established. But instead, both the nRF52832 client and the SDR are in this stage equipped with antennas. In the attack stage, the idea is to create a more realistic scenario. The same process will be implemented as in the previous profiling stage of experiment 1. But instead of using a coaxial cable to sample the EM traces, antennas are mounted on both the SDR as well as on the nRF52832 client device external antenna port. The results from this stage will be presented using the average rank function to evaluate the amount of attack traces needed on average to retrieve a sub-key.
Experimental setup 2 represents the first attack stage, the nRF52 Development Kit will be isolated from other connected devices. The setup for experiment 2 is visualized in Figure 16.

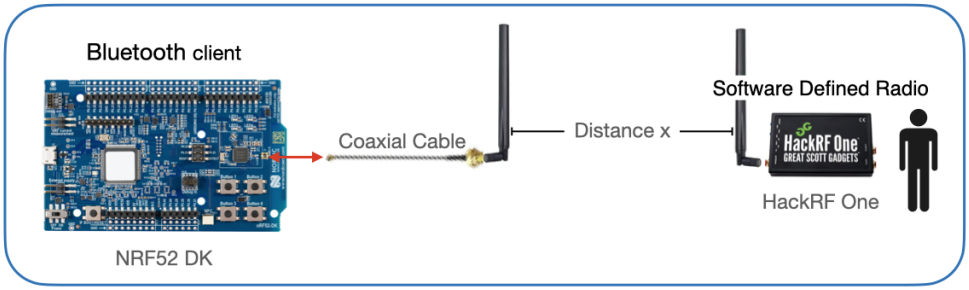


Figure 16: Experimental setup that reflects an attack scenario where the nRF52 development kit is isolated from other connected devices.

Experimental setup 3 represents the second attack stage, the nRF52 Development Kit will be placed in a system of connected devices. The experimental setup 2 is visualized in Figure 17.

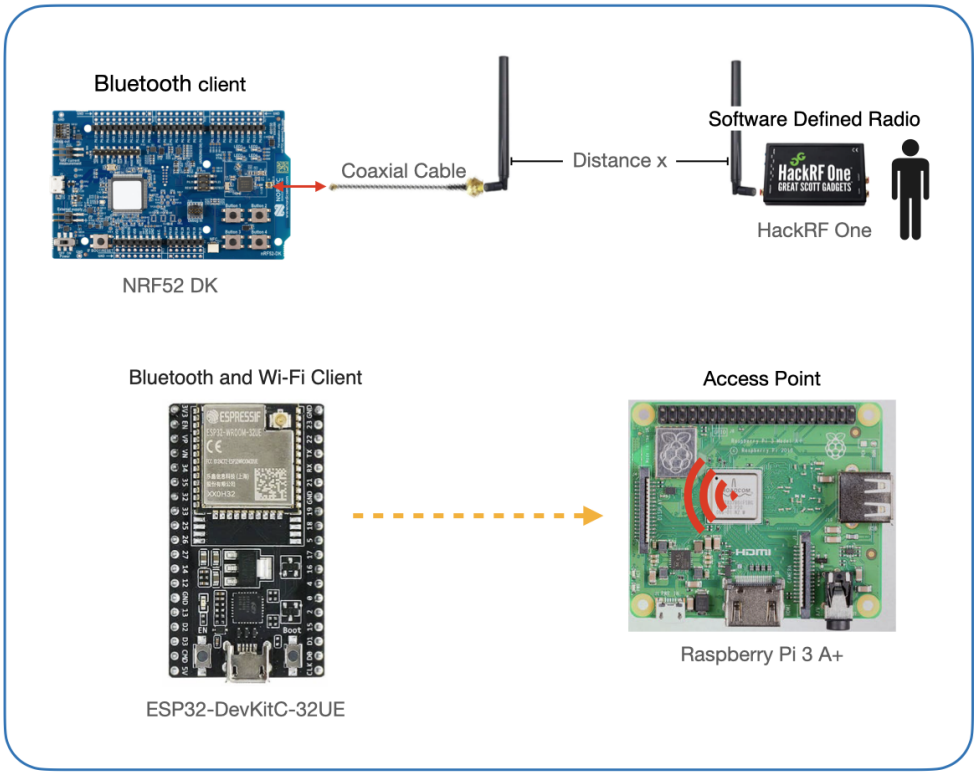


Figure 17: Experimental setup that reflects a real wold attack scenario where the nRF52 is placed in a system of connected devices.

During the data collection phase of the system of interconnected devices, depicted in Figure 17, the ESP32 MCU will be connected to the Raspberry Pi 3 A+ (RPI3A+) MCUs acting as the access points. Moreover, the ESP32 will continuously send HTTP requests

to the RPI3A+ during the entire period of the data collection phase. The wireless communication protocol is completely determined by the RPI3A+. The access point uses the IEEE 802.11b/g/n wireless LAN protocol on the 2.4 GHz frequency band.

3.3 Electromagnetic Trace Alignment and Preprocessing

In total, seven datasets of traces were acquired during this project's phase of trace acquisition. On the SDR receiver side, the central frequency was set at 2.528 GHz, a value known from earlier research targeting the same type of MCU [5, 6, 8, 11]. Trace acquisition, alignment, and preprocessing are based on the methodology provided in [11]. Through serial communication between the PC and the MCU, the PC transmits the plaintext utilized in the encryption phase together with the fixed secret key, and the MCU performs periodic encryptions. Since the key is fixed, it is only necessary to send it to the MCU once, whereas each plaintext is transferred to the MCU before each encryption phase begins. A trigger signal that can be derived from the signal trace captured by the SDR is used in order to determine the approximate beginning of each periodically executed encryption.
Figure 18 demonstrates the process of slicing periodic encryption segments from the received signal data stream, aligning the segments, and averaging the aligned traces if the same encryption is repeated.

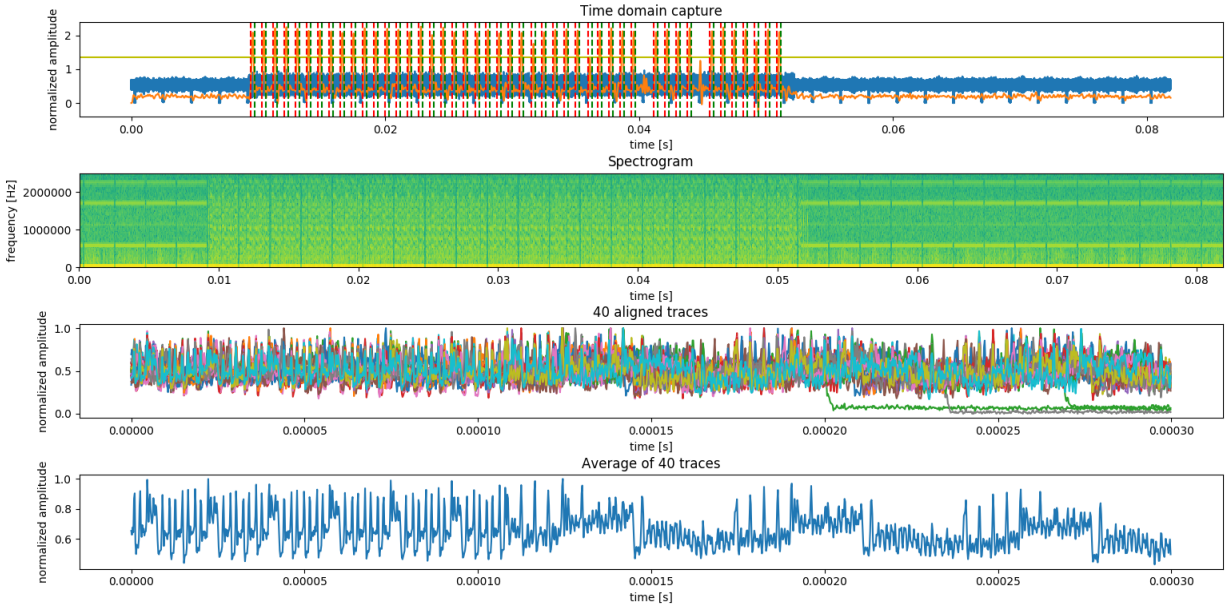


Figure 18: The preprocessing and alignment of collected traces. The upper graph demonstrates the trigger signal locating the approximate beginning of each encryption. The spectrogram of the received signal is displayed in the second upper plot. The second lower graph demonstrates the aligned traces. The lower graph represents the mean of the forty aligned traces in the second lower plot.

There are three main phases involved in extracting the trigger signal. Initially, the AM radio signals that have leaked from the MCU are demodulated by taking the absolute value of the received signal. The second phase involves passing the demodulated radio signal through a 5th-order Butterworth band pass filter with corresponding lower and upper band frequencies of 1.85 MHz and 1.95 MHz. In the third phase, the amplitude demodulated and filtered signal is passed through a 5th-order Butterworth low pass filter with a 5 kHz cutoff frequency. The approximate beginning of each encryption in the received and sampled signal stream can be determined by locating the intersections between the rising edges of the filtered signal, shown in orange on the upper graph, and the horizontal line, shown in in yellow on the same graph and indicating this signal's mean.
The trigger signal is then utilized to include parts of the received signal that contain data from a single encryption process. By adding an offset to the trigger signal, the segmented received signal now contains the data samples representing the initial key scheduling routine and the 10 rounds of AES-128 encryption, see the top graph of Figure 23.
The correlation between the received signal and a template is utilized to refine the alignment and locate the interesting data samples among the captured segments. The template is comprised of segmented and aligned traces of 1000 repetitions of the same encryption. Therefore, each trace in the raw data sets includes 4275 data samples.
Depending on the number of repetitions of the same encryption each trace in the data sets contains, each encryption is repeated with a fixed key and plaintext, and then the finely aligned traces are averaged.
Traces of all 10 rounds of encryption, including the key expansion routine, are included in the raw data sets' traces. All the data sets are processed with CPA so that each trace in the final data sets only contains the data samples of the initial 16 S-Box operations of the first round in the AES-128 algorithm. The S-Box operation of the profiling data set are located in the data sample interval of [1055, 1154] in every trace. The same operations are found to be located in the data sample interval of [1023, 1122] in every trace for both the isolated and system data sets.
Only the relevant part of the physical traces are included in the final datasets. The traces, plaintexts, keys, and labels of the profiling dataset are stored in .npy files. Figure 19 illustrates the structure of the datasets and the dimensions of the .npy files.

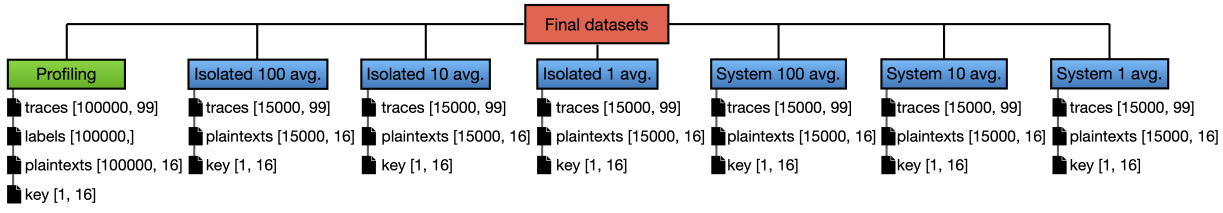


Figure 19: The structure of the datasets and the dimensions of each .npy file.

3.4 Choice of Deep Learning Architecture

The final MLP and CNN model architecture are displayed in Table 2 and 3. Both models are defined to take 99 data samples as input.

Table 2: The architecture of MLP_{99}

Layer type	Input	Output	Parameters #
Input Layer Dense	[(None, 99)]	[(None, 99)]	0
Dense	(None, 99)	(None, 100)	10000
Dense	(None, 100)	(None, 100)	10100
Dense	(None, 100)	(None, 100)	10100
Dense	(None, 100)	(None, 100)	10100
Dense	(None, 100)	(None, 100)	10100
Dense	(None, 100)	(None, 100)	10100
Output Layer Dense	(None, 100)	(None, 256)	25856
Total Parameters 86,356			

Table 3: The architecture of CNN_{99}

Layer type	Input	Output	Parameters #
Input Layer Conv1D	[(None, 99, 1)]	[(None, 99, 1)]	0
Conv1D	(None, 99, 1)	(None, 99, 1)	16
AveragePooling1D	(None, 99, 1)	(None, 99, 1)	0
Conv1D	(None, 99, 1)	(None, 99, 1)	104
AveragePooling1D	(None, 99, 1)	(None, 99, 1)	0
Conv1D	(None, 99, 1)	(None, 99, 1)	400
AveragePooling1D	(None, 99, 1)	(None, 99, 1)	0
Conv1D	(None, 99, 1)	(None, 99, 1)	1568
AveragePooling1D	(None, 99, 1)	(None, 99, 1)	0
Flatten	(None, 99, 1)	(None, 256)	0
Dense	(None, 100)	(None, 100)	608200
Dense	(None, 100)	(None, 100)	40200
Dense	(None, 100)	(None, 100)	51456
Total Parameters 701,944			

The selection of model architecture was based on previous research [6]. The sole hyper-parameter tweaked was the number of training epochs employed.

3.5 Model Training and Key Extraction

Having built the final datasets, the DL models are trained using the profiling data set, which contains 100,000 traces with matching labels, plaintexts, and key. The labels are

based on the first round AES-128 S-Box output, retrieved from the lookup-table. All labels are created by placing the result of the exclusive or operation between the plaintext-byte and the key-byte into the S-Box lookup according to

$$SBox\,output = SBox(plaintext[i] \oplus key[i]). \tag{13}$$

All trained models have a constant batch size of 128 and a learning rate of 0.0001. Adam and RMSprop are used as optimizers for the MLP and CNN architectures, respectively. For the MLP architecture, training was terminated after 100 epochs and the model was stored after each epoch, resulting in a total of 100 saved models. The same was done for the CNN architecture, in which 50 models were saved after 50 epochs of training. The CNN model was limited to 50 epochs since it overfitted earlier than the MLP model when smaller tests were conducted.

Using the rank test, the optimal model for the MLP and CNN architectures was selected. Figure 20 illustrates how the rank is computed using traces, their related plaintexts, and the key.

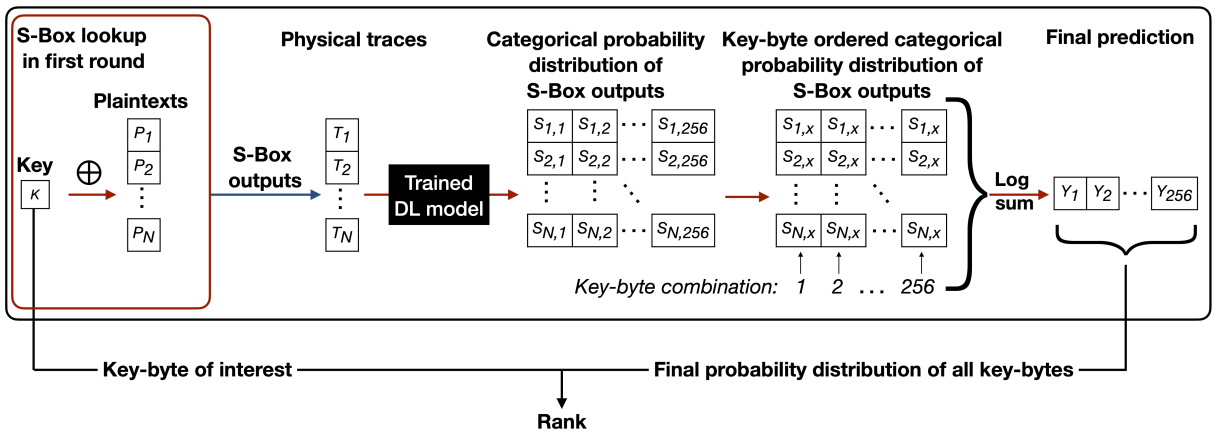


Figure 20: The procedure of key extraction utilizing the DL model and rank test.

To evaluate the model, we must use any of the attack datasets shown in blue in Figure 19. The rank test begins with the calculation of the S-Box output given a plaintext and a key, as depicted in the red box in Figure 20. The corresponding physical trace is then inserted into the DL model.

The DL model generates a categorical distribution for each input trace. Each categorical distribution represents the likelihood of all the 256 possible S-Box outputs. Since the plaintext corresponding to each categorical distribution is known, each element in the categorical distribution corresponds to a certain key-byte. The output probability of the S-box is then sorted, with each column of the ordered matrix corresponding to a particular key-byte. Depending on the number of traces N for which we desire to calculate the rank, the final prediction will be the logarithmic sum of the N ordered categorical distributions, yielding a final prediction given N traces, with each element of the final prediction vector of length 256 representing a unique key-byte.

The key-byte is unknown to a real attacker, who would simply assume that the key-byte in the final prediction vector with the highest probability is the actual key-byte. In this study, however, the key-byte is known, so we can locate the actual key-byte within the final prediction vector and determine its rank from there.

4 Analysis and Results

This section begins by introducing the partial findings of the alignment and preprocessing of EM traces for this study. The performance of the CPA in ranking the 16 subkeys is also provided. The final architecture of the MLP and CNN models, as well as the selection of hyper parameters, will be shown. Finally, the performance of the SCA utilizing the chosen DL architectures is illustrated. Experiments 2 and 3 yielded six distinct attack datasets, upon which the efficiency of the SCAs was evaluated.

4.1 Trace alignment and preprocessing

Following the process of the three experiments described in Section 3, the traces were preprocessed, aligned, and sorted into the seven distinct datasets listed in Table 4. One referred to the Profiling dataset, where each trace corresponds to the average of 100 encryptions of the identical plaintext performed in experiment 1. In experiment 2, three datasets referred to as Isolated 1, 10, and 100 correspond to traces collected in an isolated environment, where each trace corresponds to the average of 1, 10, and 100 encryptions, respectively. Moreover, three datasets referred to as System 1, -10, and -100 correspond to traces collected in the system environment during experiment 3, with each trace corresponding to the average of 1, 10, and 100 encryptions, respectively.

Table 4: Final datasets after preprocessing and trace alignment. A single trace consists of 99 data samples representing the S-Box output of the AES-128 encryption.

Dataset	Distance to device	Average of measurement of the same encryption	Number of traces
Profiling	coaxial cable	100	100 000
Isolated 100	2m	100	15 000
Isolated 10	2m	10	15 000
Isolated 1	2m	1	15 000
System 100	2m	100	15 000
System 10	2m	10	15 000
System 1	2m	1	15 000

Using templates, we align all traces in each of the three experiments with more accurate precision. After calculating the correlation between the template and the traces, each trace

is comprised of 4275 data samples. For every experiment conducted, representing a unique setting, a new template was designed. The template utilized to fine-align the traces during the profiling phase is depicted in Figure 21, which represents the mean of one 1000 traces.

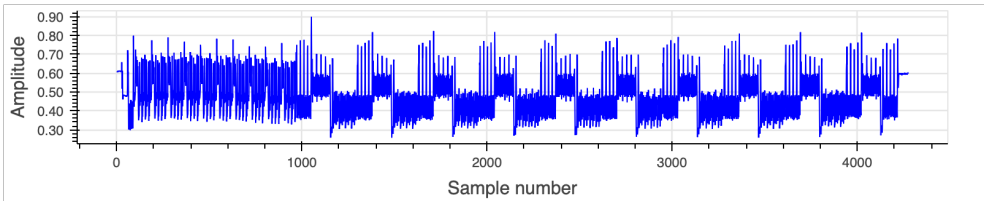


Figure 21: The template used to fine align the traces captured in the trace acquisition phase. This template contain 4275 data samples and due to this every trace in all the data sets also contain exactly 4275 data samples.

All traces currently contain 4275 data samples representing the whole AES-128 encryption procedure. We are only interested in a tiny subset of the samples in each trace, which reflect the S-Box outputs from the initial round. Using CPA analysis, the interesting segments of the traces were identified. The CPA model will be utilized to identify the highest correlation between the S-Box output and the trace data samples. The upper graph of Figure 22 displays the correlation value for each trace processed by the model, and it can be seen that the correlation is strong in the interval $[1000, 1200]$ of the samples. The lower graph depicts how, during an iterative procedure, the interval of data samples is decreased until only the 16 S-Box operations are displayed.

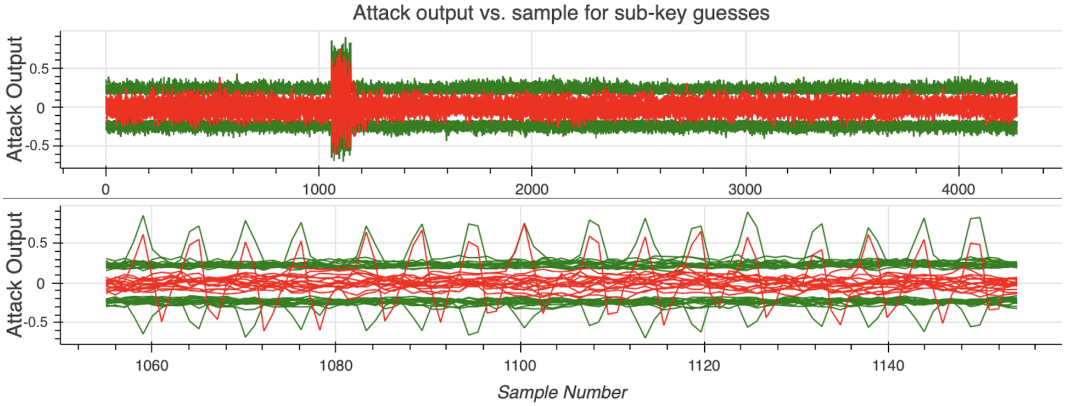


Figure 22: CPA analysis to find the interesting part of the trace. Beginning with a traces holding around 4275 and ending with a trace of 99 data samples containing only the information of the S-Box output.

By iteratively decreasing the interval of data samples in the traces provided to the CPA model, the location of the interesting data samples is determined. Beginning with traces containing 4275 data samples, the traces of the final datasets will only comprise 99 data samples representing the S-Box outputs of AES-128 in the first encryption round.

Figure 23 shows a trace from the Profiling dataset and the top graph represents the whole trace of 4275 data samples. Since the location of the S-Box outputs are known from CPA analysis, it is possible to zoom in on the 16 S-Box operations in the trace. The middle graph represents the whole first round of encryption and the lower graph shows 16 distinct peaks corresponding to the 16 S-Box operations. The traces that are stored in the final datasets contain 99 data samples representing the 16 S-Box outputs in every trace.

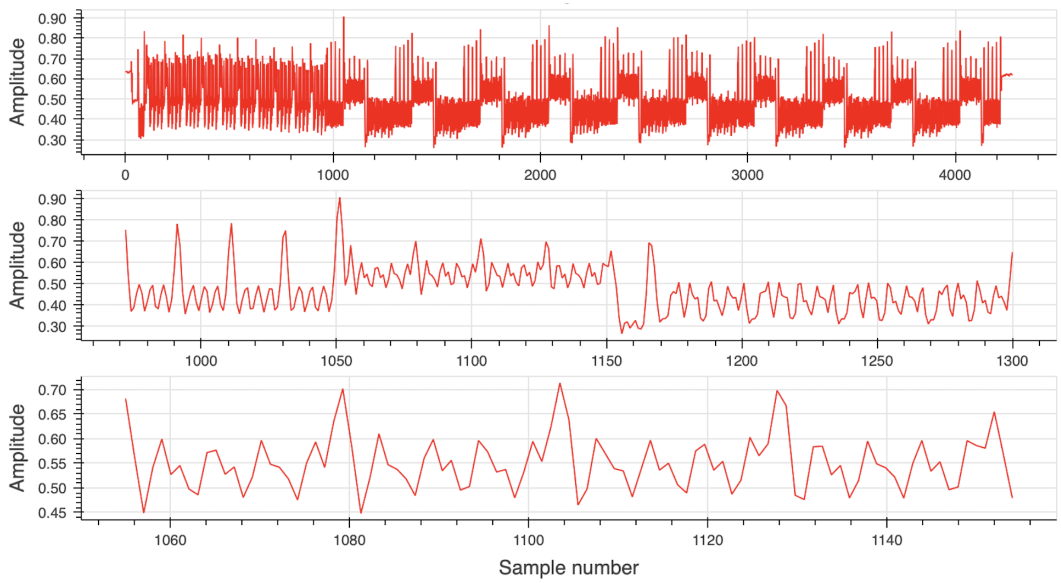


Figure 23: Using the CPA model, the original trace's interesting data points can be located in the interval $[1055, 1154]$ of the profiling dataset traces.

Figures 24 and 25 illustrate, for the Isolated and System environments, respectively, the impact of averaging the traces. Given that the number of averaged traces increases from the top to the bottom of the graphs, it is clear that the quality of the traces also improves. Examining the graph of 100 averaged traces for both figures, the key expansion procedure and the subsequent ten rounds of encryption can be distinguished. In the corresponding centered graphs showing 10 averaged traces of the same encryption for the isolated and system environments, the typical pattern of the encryption process can still be seen, but not as clearly as for the 100 averaged traces. However, for the two traces in the top graphs of the two figures, each of which represents a single aligned trace collected in the isolated and system environments, it is now harder to identify the typical patterns of the encryption process.

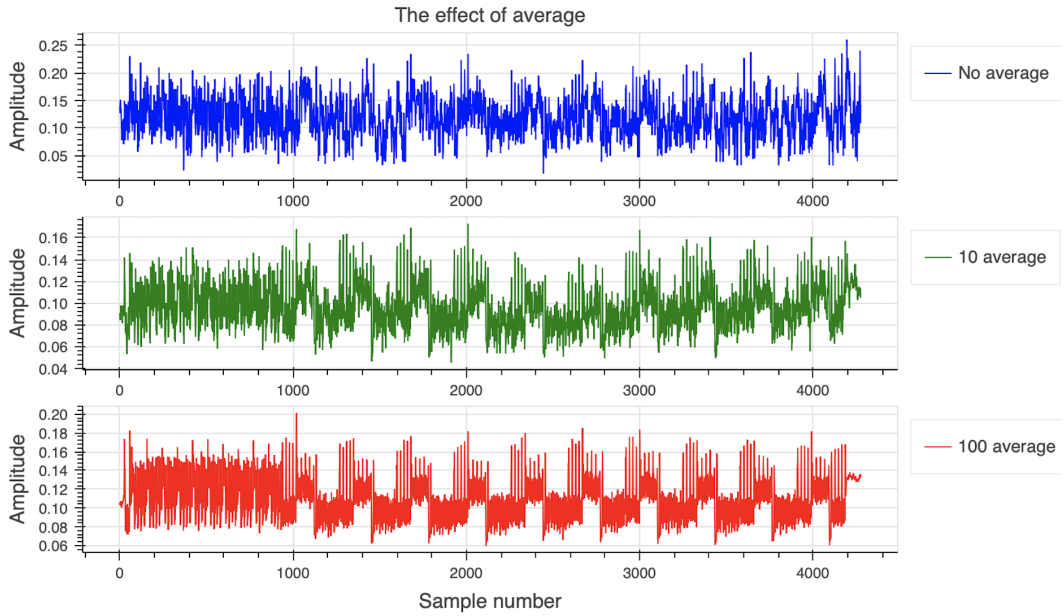


Figure 24: The effect of averaged traces captured in the isolated environment.

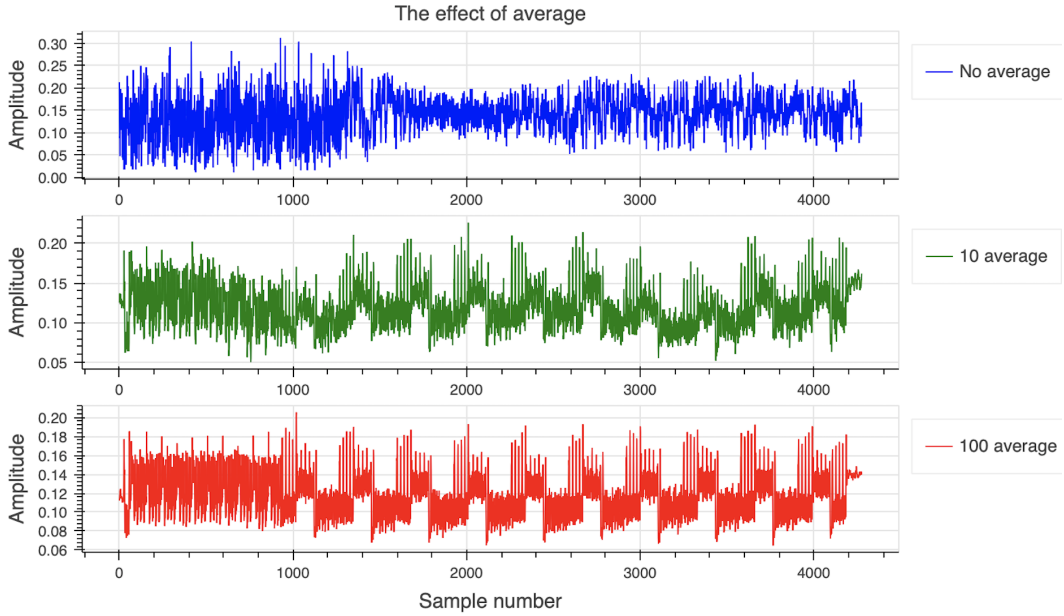


Figure 25: The effect of averaged traces captured on the system environment.

The data samples from the various types of datasets that are provided as input to the DL models are depicted in Figure 26. This figure visualizes 100 averaged traces captured and segmented from each of the three distinct stages, namely the profiling, isolated, and system stages. Since the traces were collected in diverse environments, the signal strength and corresponding amplitude differed substantially between traces. Before providing it as

input to the model, the final segmented traces was therefore normalized on a scale of 0 to 1 before being fed as an input to the DL model.

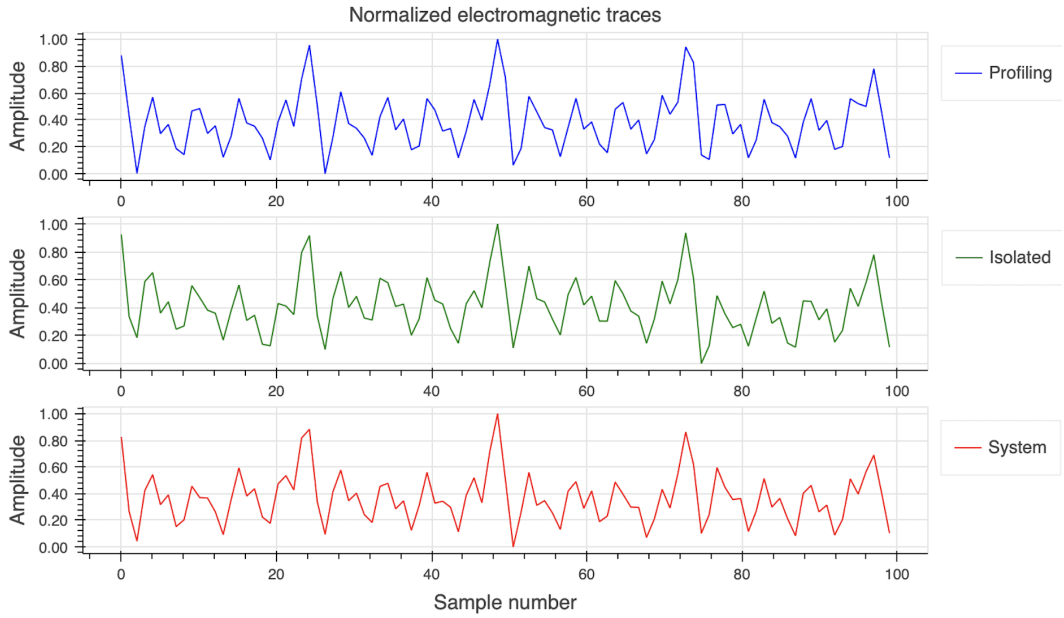


Figure 26: Example of some of the normalized traces fed the DL model.

Figure 27 shows the Partial Guessing Entropy (PGE) of all 16 sub-bytes of the key, calculated by the CPA model. The model was implemented on each of the seven datasets. Only for the Profiling and Isolated 100 datasets did the Partial guessing entropy of every key-byte converge towards zero. However no key-byte reached a PGE of 0 within 15000 traces. With the exception of the Profiling and Isolated 100 datasets, no other datasets converged, as evidenced by the random appearance of the PGE.

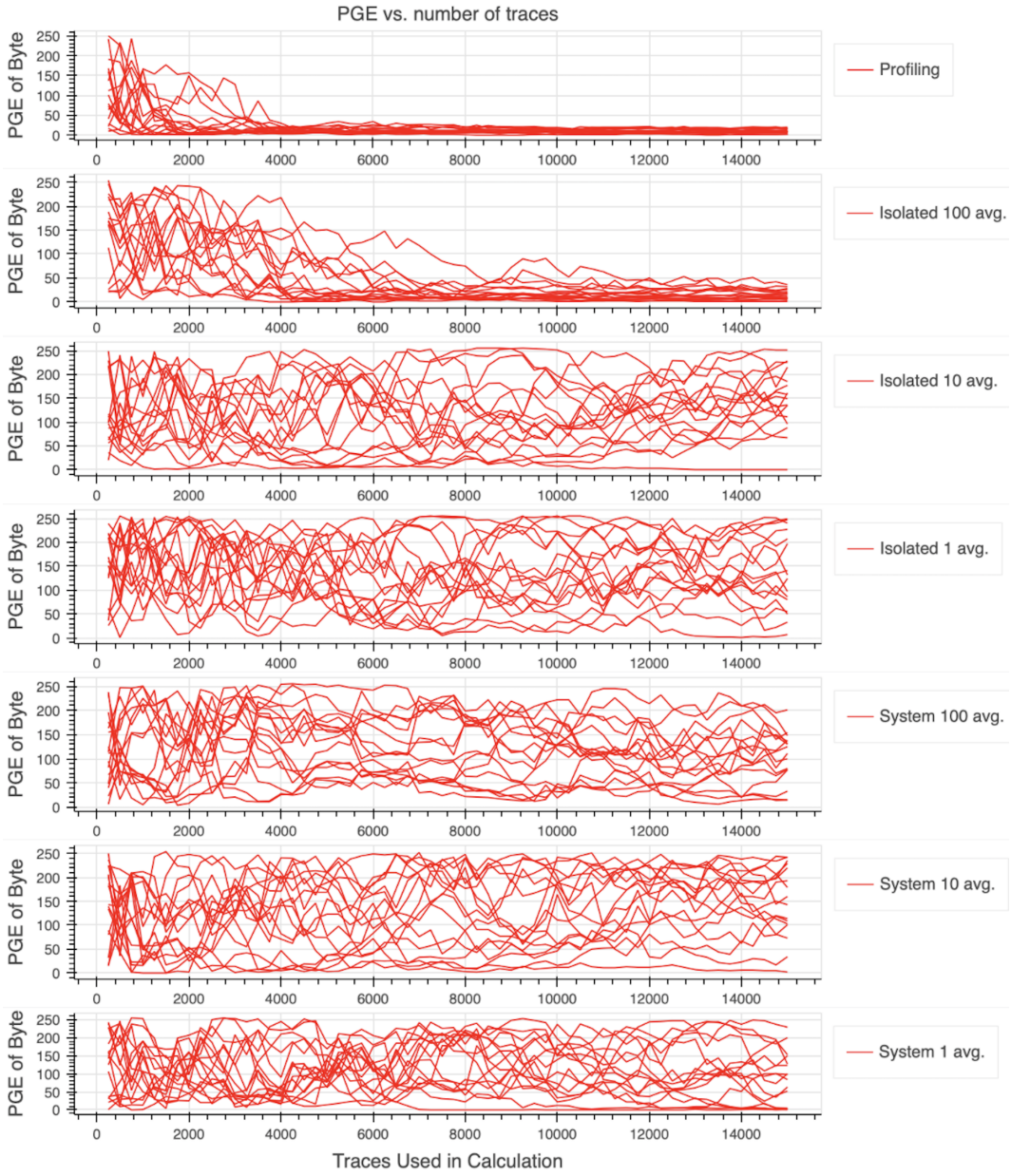


Figure 27: Partial Guessing Entropy of all 16 sub-keys in the secret key using Correlation Power Analysis. The model was applied to all of the final datasets.

4.2 Tuning of Deep Learning Models

In order to yield a good SCA efficiency, proper fitting of the models trainable parameters had to be considered. Figure 28 demonstrates the consequences of improper fitting. The red line indicating the CNN model with 22 training epochs falls below or equals 0.5 after 12007 traces. The blue line indicating the CNN model with 10 training epochs and does not drop below mean rank 80 within 15000 traces. The green line, representing the CNN

model with 40 epochs, quickly converges to rank 1 after 100 traces, but does not reach a value below or equal to 0.5 within the 15000 traces.

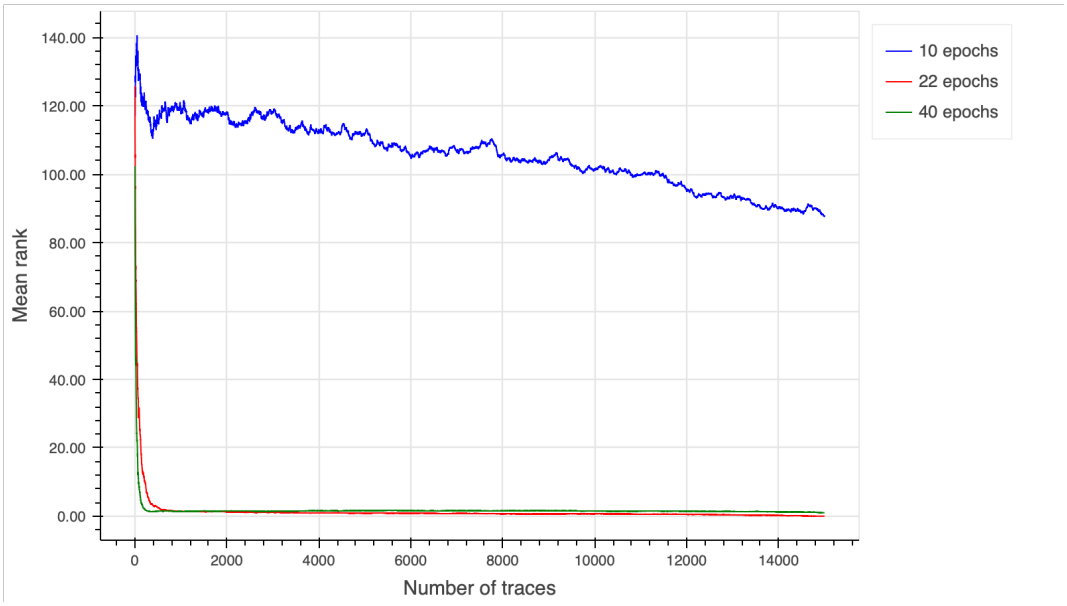
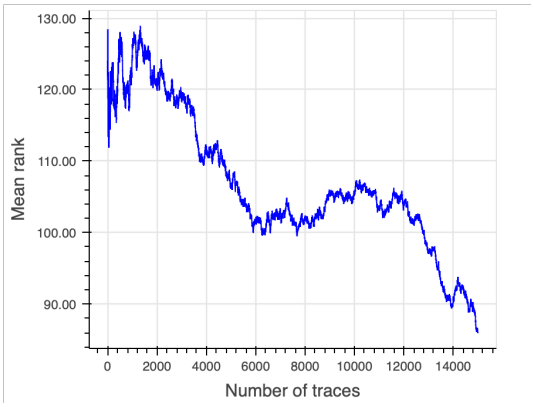


Figure 28: The consequences of improper fitting. Running the same CNN model with different amount of epochs.

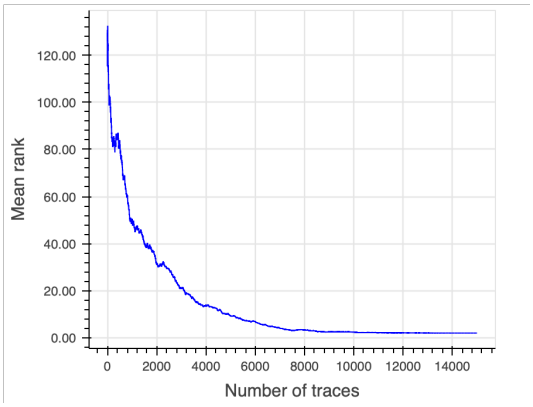
The average rank test was conducted on all stored models, which included 100 MLP architecture models and 50 CNN architecture models, respectively. Best performance was attained with 85 epochs for the MLP architecture and with 22 epochs for the CNN architecture.

4.3 Results of Mean Rank Tests

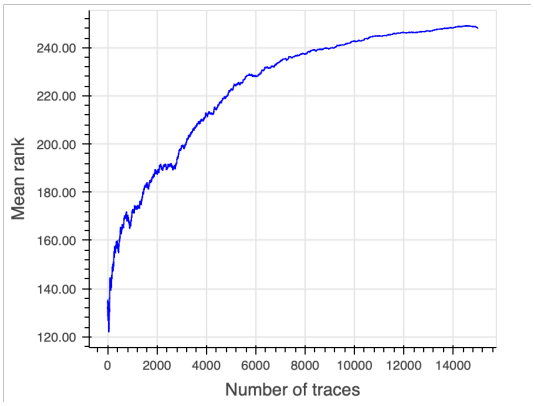
Figure 29 and 30 below show the efficiency evaluation of the six attack datasets using the MLP and CNN models, respectively. The mean rank is used as the evaluation methodology, and the findings are based on an average of 100 rank function tests. Each trace is the average of either 1, 10, or 1000 measurements of the identical encryption.



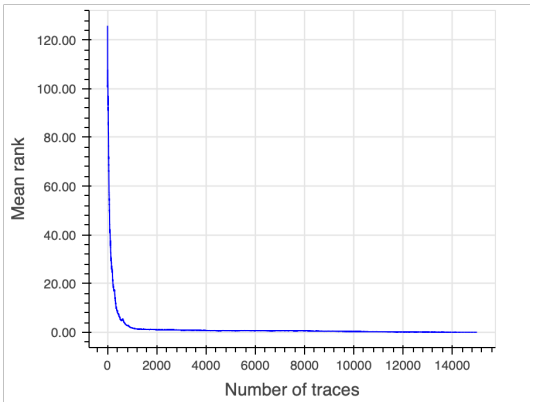
(a) System environment with no averaged traces.



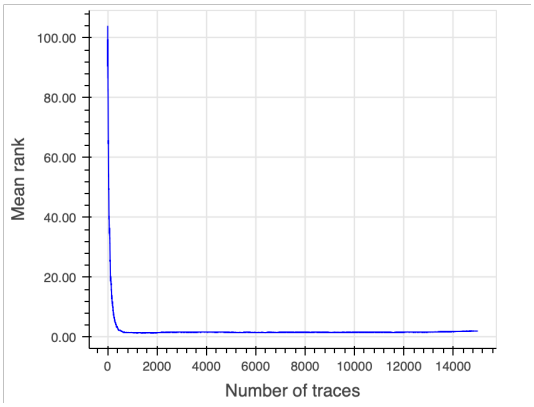
(b) Isolated environment with no averaged traces.



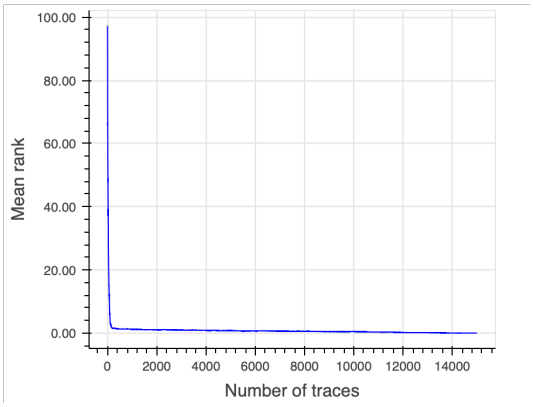
(c) System environment with averaged traces of 10 measurements.



(d) Isolated environment with averaged traces of 10 measurements.

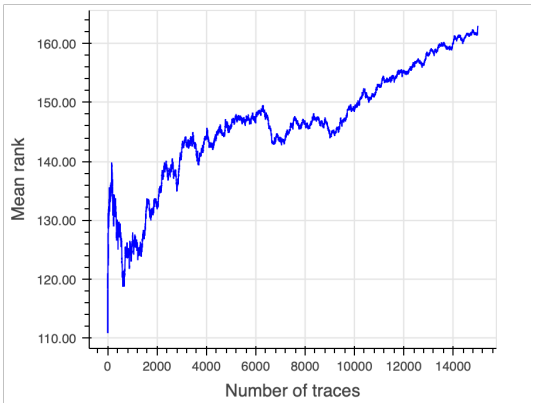


(e) System environment with averaged traces of 100 measurements.

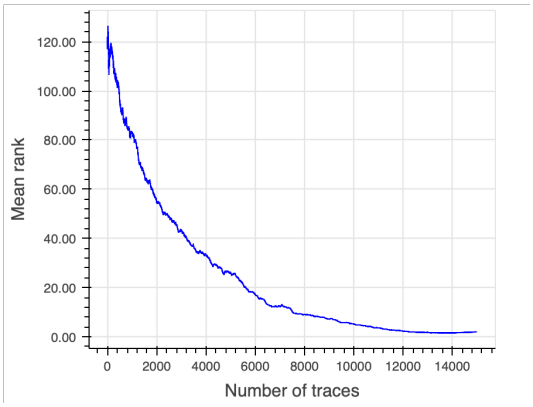


(f) Isolated environment with averaged traces of 100 measurements.

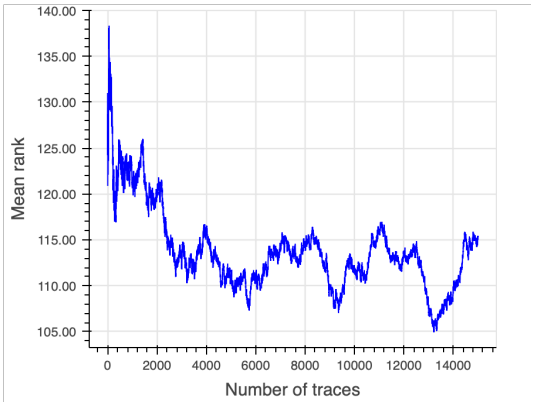
Figure 29: Average rank of 100 test for traces captured in an isolated and system environment using a MLP model presented in Table 2 with 85 epochs and batch size 128.



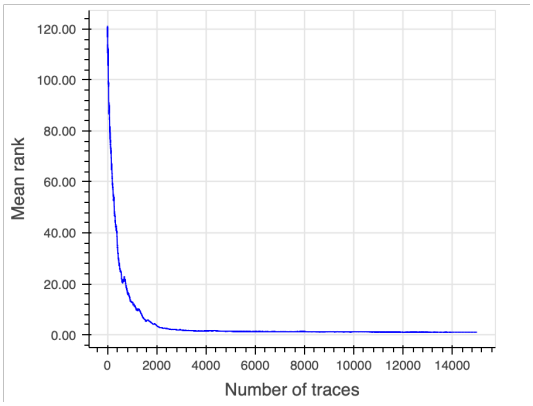
(a) System environment with no averaged traces.



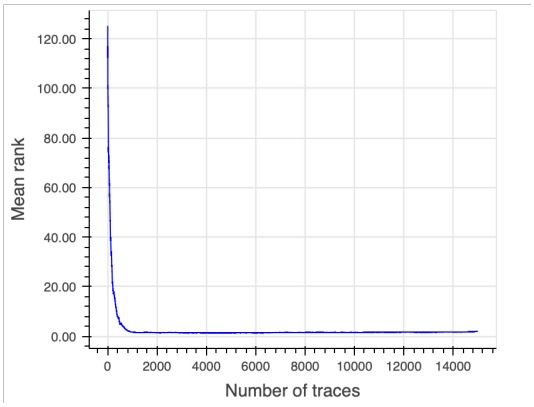
(b) Isolated environment with no averaged traces.



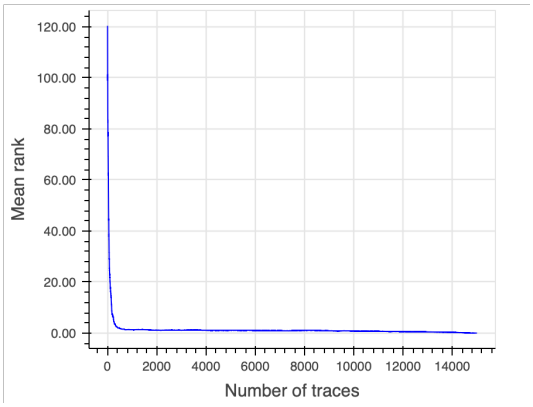
(c) System environment with averaged traces of 10 measurements.



(d) Isolated environment with averaged traces of 10 measurements.



(e) System environment with averaged traces of 100 measurements.



(f) Isolated environment with averaged traces of 100 measurements.

Figure 30: Average rank of 100 test for traces captured in an isolated and system environment using a CNN model presented in Table 3 with 22 epochs and batch size 128.

From Table 5, one can see that only three different system settings reach an average below 0.5 within the 15000 traces used in the rank test for different environment settings. The rank test of the dataset corresponding to 100 averaged traces collected on the isolated system environment converges to a value below or equal to 0.5. The number of traces needed from this environmental setting to find the right sub-key is 8406 and 12182 traces for MLP and CNN models, respectively.
In the average rank test of the dataset corresponding to 10 averaged traces collected on the isolated system environment, only the MLP model converges to or below 0.5 using 8198 traces. For the rest of the datasets collected in other environmental settings, none of the models converges to or below 0.5 in the average rank test.Although most average rank tests do not converge to a result equal to or lower than 0.5 within 15000 traces, some converge to a relatively low value above 0.5.
Table 6 indicates the value at which each rank test converges and the number of traces required to attain this value when the mean rank is steady. The table indicates convergence when the mean rank is less than or equal to 5 and remains below this threshold for the subsequent 5 calculations of mean rank.

Table 5: Number of traces required on average to retrieve the sub-key. The findings are based on an average of 100 rank function tests. Each trace is the average of N measurements of the identical encryption.

	N=1		N=10		N=100	
Setup	MLP_{99}	CNN_{99}	MLP_{99}	CNN_{99}	MLP_{99}	CNN_{99}
Isolated	>15000	>15000	8198	>15000	8406	12182
System	>15000	>15000	>15000	>15000	>15000	>15000

Table 6: Number of traces needed to reach a Converged solution $\leq$5. The findings are based on an average of 100 rank function tests. Each trace is the average of N measurements of the identical encryption.

	N=1		N=10		N=100	
Setup	MLP_{99}	CNN_{99}	MLP_{99}	CNN_{99}	MLP_{99}	CNN_{99}
Isolated	6501	10040	530	1752	98	249
System	>15000	>15000	>15000	>15000	600	493

5 Discussion and Conclusion

5.1 Discussion

This study aimed to determine how efficient DL SCAs are in retrieving part of the secret key in a simple IoT system equipped with Bluetooth 5.3 and IEEE 802.11b/g/n wireless LAN protocol. To address this research question, we created two attack scenarios representing the two different environmental settings in experiments 2 and 3. From the results of the two separate attacks, we can analyze the attack's efficiency in each environment and compare them to determine how the ambient noise of the wireless protocol operating in the same space affects the efficiency of these types of attacks. In addition, we will discuss what may be valuable to investigate in future research, such as the extent to which distinct environmental settings influence the effectiveness of SCAs and their potential causes.

Figures 29 and 30 in the Results section show the mean rank of the individual datasets collected in Experiments 2 and 3. Table 5 displays, for the two tests, the average minimum size of the attack set traces required to ensure that the correct key is ranked highest among the 256 possible key-byte combinations.
Examining Table 5 reveals that 50 percent of the mean rank tests converge to a mean rank value of 0.5 or less within the 15000 traces for attack datasets collected in experiment 2. Two of these mean rank tests correspond to MLP and CNN architectures processing traces, each composed of 100 averaged traces of the same encryption. The last test corresponds to the MLP architecture processing traces composed of ten averaged traces of the same encryption. The remaining rank test does not converge to a mean rank equal to or below 0.5 within 15000 traces. As a result, three out of twelve mean rank tests converged to a value corresponding to the rank reaching zero in most test sets. An attacker using these models for the three corresponding settings would expect the sub-key with the highest probability in the final prediction vector to be the correct key.
Although most mean rank tests do not reach the desired mean rank of 0.5, eight out of twelve mean rank plots of Figures 29 and 30 reveal distinct patterns indicating that the majority of mean rank tests for the various datasets and models converge to relatively low mean rank values within the 15000 traces. The number of traces processed before the mean rank reaches a stable value of below or equal to five, as demonstrated in Table 6, indicating when the mean rank reaches stable values below or equal to five. In practice, this means that an attacker in these eight corresponding settings, using these two models, would expect to find the correct key-byte within the six first key-bytes of the final prediction vector.
The datasets corresponding to ten average traces of the same encryption and no averaged traces, both of which were collected in experiment 3, did not achieve a sufficiently low rank to be classified as even partially successful in retrieving the correct sub-key. Examining the four graphs 29a, 29c, 30a and 30c in Figures 29 and 30, one can determine that only 29a and 30c display patterns that tend to converge toward a lower mean rank value.
Although the desired rank can not be reached for most datasets, there is a clear pattern in ten out of twelve graphs in Figure 29 and 30; most mean rank tests reach a value of five or

below within 15000 traces. In comparison with the Partial Guessing Entropy presented in Figure 27, it is evident that the DL models are better for finding the correct key byte in a hard case like this. Only for the dataset where each trace is composed of 100 averaged traces of the identical encryption is CPA able to show some tendency of converging as the number of traces used in the calculation increases.

Optimization and tuning of DL models have a substantial impact on the effectiveness of SCAs. As shown in Figure 28, the number of epochs employed during the training phase of the DL model has a significant effect on its performance in an SCA. Since model architecture and hyper-parameters are based on previous research, adjustment of model parameters was limited to selecting training epochs. The network design and remaining hyper-parameters were optimized using a different data set, other hardware for trace collection, and extra hardware such as a low noise amplifier. Thus, the network architecture and hyper-parameters are optimized for the conditions in this previous research. The optimization of DL models was not the main focus of this investigation. Nonetheless, supposing that additional tuning of the models' architecture and hyper-parameters had been undertaken in this work a better mean rank could have been achieved, and the consequent efficiency may have been improved.

The efficiency of SCA differs significantly between experiments 2 and 3. In the second experiment's isolated setting, part of the resulting mean ranks reached the desired value, indicating that the model correctly identified the correct subkey within 15000 traces. Examining Table 6, datasets corresponding to the isolated environment in experiment 2 converge to a steady value, less than or equal to five, faster than datasets related to the system environment in experiment 3. Within the 15000 traces of datasets collected in experiment 3, only the dataset corresponding to traces comprised of 100 averaged traces of the same encryption achieved a constant mean rank value equal to or below five. As previously indicated, Figures 29a and 30c for the remaining datasets of experiment 3 are displaying converging patterns. However, their equivalent in experiment 2 exhibits a more prominent and faster convergence tendency.

5.2 Conclusion

To summarize the results, 12 SCAs are performed on six datasets captured in two distinct environments using two different DL models for each dataset. The correct key-byte can be retrieved in three of these SCAs. All three successful attacks are made in an isolated environment without any interfering noise. The best performance is achieved with the MLP DL architecture, processing traces each composed of 10 averaged traces of the identical encryption, and the correct key-byte is recovered after 8198 traces. The two other successful attacks correspond to the MLP and CNN models processing 100 averaged traces of the identical encryption, and the correct key-byte is recovered after 8406 and 12182 traces, respectively. In the system environment to which we added a Wi-Fi communication, no

SCA could recover the correct key-byte from any of the three datasets within 15000 traces. It becomes clear that interfering signals in the system environment negatively impact the performance of SCAs. The DL models are trained on data acquired in a noise-free environment, which may explain why the SCAs performed poorly in the system environment. If the DL models are trained on a more diverse dataset containing additional noisy traces, performance could be improved for the SCAs conducted in the system environment. In addition, optimizing the hyper-parameters of the DL models using the profiling dataset collected in this study is likely to improve the performance of the SCAs in both environments.

Future research may investigate the effect of other distinct ambient noise sources on the performance of DL-SCAs. In this investigation, an IEEE 802.11b/g/n wireless LAN protocol was employed to transmit and receive HTTP requests and responses as a source of ambient noise. Zigbee and Bluetooth Low Energy are other alternative sources of background noise. Investigating if disruptive noise could be used as a countermeasure for SCAs is an additional relevant future topic. However, there is a possibility that an optimized and generalized DL model trained on a wide variety of data could circumvent these countermeasures. An improvement to this study would be to add a controlled quantity of additive white Gaussian noise to the traces in the dataset collected in experiment 1 that is used for the profiling stage.

References

1. Emmanuel, P., Remi, S., Ryad, B., Eleonora, C. & Cecile, D. Study of deep learning techniques for side-channel analysis and introduction to ascad database. *CoRR,* 1–45 (2018).
2. Dworkin, M. *et al. Specification for the Advanced Encryption Standard (AES)* tech. rep. (National Inst of Standards and Technology Gaithersburg MD Computer security Div, 2001).
3. Björklund, F. & Landin, N. *Board and Chip Diversity in Deep Learning Side-Channel Attacks: On ATtiny85 Implementations Featuring Encryption and Communication* MA thesis (Royal Institute of Technology, 2021).
4. Zhao, Z. *Far Field Electromagnetic Side Channel Analysis of AES* MA thesis (Royal Institute of Technology, 2020).
5. Wang, R. *Deep Learning Based Side-Channel Analysis of AES Based on Far Field Electromagnetic Radiation* MA thesis (Royal Institute of Technology, 2020).
6. Wang, R., Wang, H. & Dubrova, E. *Far field EM side-channel attack on AES using deep learning* in *Proceedings of the 4th ACM Workshop on Attacks and Solutions in Hardware Security* (2020), 35–44.
7. Liu, K. *Far Field EM Side-Channel Attack Based on Deep Learning with Automated Hyperparameter Tuning* MA thesis (Royal Institute of Technology, 2021).
8. Wang, R., Wang, H., Dubrova, E. & Brisfors, M. *Advanced Far Field EM Side-Channel Attack on AES* in *Proceedings of the 7th ACM on Cyber-Physical System Security Workshop* (2021), 29–39.
9. Daemen, J. & Rijmen, V. *The design of Rijndael* (Springer, 2002).
10. Dworkin, M. *Recommendation for block cipher modes of operation. Methods and techniques* tech. rep. (National Inst of Standards and Technology Gaithersburg MD Computer security Div, 2001).
11. Camurati, G., Poeplau, S., Muench, M., Hayes, T. & Francillon, A. *Screaming channels: When electromagnetic side channels meet radio transceivers* in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (2018), 163–177.
12. *Correlation Power Analysis* `https://wiki.newae.com/Correlation_Power_Analysis`. Accessed: 2022-06-06. 2018.
13. McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **5,** 115–133 (1943).
14. Goodfellow, I., Bengio, Y. & Courville, A. in, 264–267 (MIT Press, 2016).
15. Goodfellow, I., Bengio, Y. & Courville, A. in, 267–272 (MIT Press, 2016).
16. Goodfellow, I., Bengio, Y. & Courville, A. in, 280–284 (MIT Press, 2016).

17. Goodfellow, I., Bengio, Y. & Courville, A. in, 303–305 (MIT Press, 2016).
18. Goodfellow, I., Bengio, Y. & Courville, A. in, 326–327 (MIT Press, 2016).
19. Goodfellow, I., Bengio, Y. & Courville, A. in, 327–329 (MIT Press, 2016).
20. Goodfellow, I., Bengio, Y. & Courville, A. in, 335–339 (MIT Press, 2016).
21. Simonyan, K. & Zisserman, A. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556* (2014).