Frankfurt University of Applied Sciences
Fachbereich 2: Informatik und Ingenieurwissenschaften
Studiengang Wirtschaftsinformatik

Master-Thesis
zur Erlangung des akademischen Grades
Master of Science (M.Sc.)

Detection of Smart Card Security Leakages Based on Deep Learning

vorgelegt von Johannes T. Brandau

am 27. März 2020

Referent: Prof. Dr. Josef Fink
Korreferent: Prof. Dr. Erwin Hoffmann

Erklärung

Ich versichere, dass ich die Master-Thesis selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Wiesbaden, 27.03.2020 Johannes T. Brandau

Contents

List of Figures

List of Tables

List of Source Codes

Chapter 1

Introduction

Since contactless payment has become more and more popular, smart cards in mobile devices for payment are getting common. In order to improve the security of smart cards and smartwatches, the side-channel leakage assessment is used to detect and eliminate possible security gaps at an early stage. Today, universal statistical distinction such as T-Test or X^2-test are effective methods to detect security gaps.

Amor Maoradi, Tobias Schneider and Thorben Moos describe a new technique based on deep learning in their paper "DL-LA: Deep Learning Leakage Assessment: A modern roadmap for SCA evaluations" [26]. This technique is designed to find leaks in secured and unsecured traces and is an alternative to the classical T-Test.

The following research work investigates if deep learning can be used to identify leakages in smart cards during the encryption process. The process to find leakages is necessary to figure out if a device is safe for mobile payments. Security leaks may cause that the encryption key can be decrypted and in the worst case used to make unauthorized payments. New techniques to find security leakages are welcome if they are transparent and traceable. The research in this work is split into two parts. First, simulated traces will be generated and analyzed with T-Tets and deep learning. Next real traces from a real device will be analyzed with the T-Test and deep learning technique. To figure out if these techniques can be used even with countermeasures, different kinds of countermeasures are used to protect the traces.

Chapter 2

Fundamentals

In the Fundamentals, all concepts and theoretical basics that are necessarily relevant for research are explained. The Fundamentals are split into four parts. The first part 2.1 describes the basic technique of virtual smart cards. The second part 2.2 explains artificial intelligence and the usage of machine learning. The third part 2.3 specifies the used hardware and setup of the measurement environment followed by the last part 2.4 the basics of IT-Security.

2.1 Smart Cards

Over the last few years, the use of smart cards has changed considerably. A smart card is a card with a chip on it with allows to authenticate a user. The possibilities are diverse. Be it at doors or also for payment. During the authentication process, a password can also be required. The authentication is only valid if the password matches to the card. A classic example is the credit card. The password in combination with the card authorizes the cardholder to pay. In the course of the last few years, a technical revolution has also taken place in payment. Contactless payment was introduced, where the card only has to be held at the device. With this new method of payment, the virtual smart card has become increasingly important. Mobile devices that have the technical requirements can be used for payment. The smartphone only has to be held close to the card reader, after which the user has to authenticate himself by password, fingerprint or face recognition. If the authentication is successful, the payment process is complete. For contactless payment, the mobile devices are equipped with a Near-field communication (NFC) interface. NFC enables contactless communication over short distances.

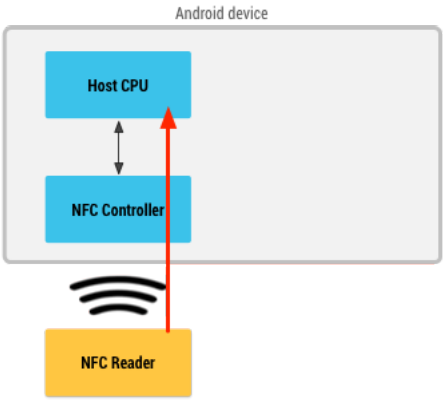


Figure 2.1 Host-based card emulation [8]

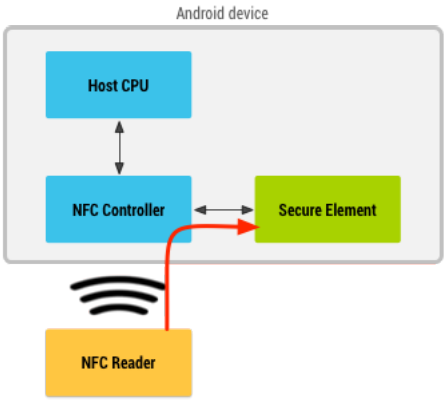


Figure 2.2 Card Emulation with secure element [8]

There are two different techniques to implement a virtual smart card in a smartphone. One is the Host-based card emulation (HCE) 2.1. The mobile device reads the card, and the data is forwarded to the central processing unit (CPU) for processing. Encryption and communication are handled entirely by the CPU.

Another method is using a secure element between the communication 2.2. The data will be first encrypted in a secure element and then forwarded to the CPU for processing [8]. In the research work, it is assumed that it is a host-based architecture and that the encryption is done inside the CPU, not in the secure element. Evaluation with a secure element is much more complicated.

HCE was in the past a common technique to emulate smart cards. Furthermore, for in android, the secure element has to be implement correctly. Mistakes during the implementation or in the usage of older hardware, the encryption will be calculated in the CPU. Older smartphones have no secure element in this case HCE is the only possible solution.

2.2 Artificial Intelligence

Over the last few years, artificial intelligence (AI) has become increasingly important. However, artificial intelligence is nothing new. In the 90s, AI was already used in a chess challenge. Technical progress, clouds and cheap hardware make it possible for AI to be used more and more. AI is a superordinate term and can be divided into two areas: Machine learning and deep learning. ML has been used as an example since the 80s and is popular in the areas of predictive analytics. Deep learning became known in 2010, mainly through the recognition of objects in images or videos [19]. This chapter is split into two parts. The first section 2.2.1 describe the classical machine learning techniques, the second section 2.2.2 illustrates deep learning more in detail.

2.2.1 Machine Learning

Machine learning (ML) is a subset of artificial intelligence and is an approach to data analysis that involves building and adapting models. It involves the construction of algorithms that adapt their models to improve their ability to make predictions. To predict the classification is based on statistical weights. The classical machine learning is not a new discipline and can split into three different classes [5]:

Supervised

The supervised learning is one of the most popular techniques. The dataset is split into the regular data (feature) and the label. In supervised learning, the dataset will split into train and test data. At first, the machine learning algorithm learns with the feature and the label from the training dataset. After the training, the algorithm gets only the feature of the test data and has to predict the label. The predicted label will be we compared with the original label of the test data to calculate the accuracy [5].

Supervised learning can be used in two different areas: Classification and Regression. The difference is the label. In classification, the data is divided into classes with the use of the label. New data should always be assigned to predict the correct classes. The label of the new data is unknown. An example could be weather data like temperature and pressure with the labels sun, rain, cloudy and snow.

Another method is the regression, where the label does not represent classes, but measurable units divided like quantity, time or size data. After training the data, it is possible to estimate the label of unknown data. An example could be weather data, and the label could be the number of litres of rain per square metre [5].

Unsupervised

Another type of machine learning is unsupervised learning. In this case, no labels are defined as in supervised learning. The goal is to use the algorithms to find patterns and structures that are not found in supervised learning. Clustering is one of the best-known methods. In unsupervised learning, two methods are often used: Clustering and Dimensionality Reduction.

In clustering, different clustering procedures are used to try to gain new and correlated information. This technique can be especially helpful for data understanding or feature generation. Dimensionality reduction is used to reduce the complexity of data sets. With the help of algorithms, unexpressed robust data and features will be removed, and the dimensions of the data reduced [5].

Reinforcement

Reinforcement Learning is another type of machine learning. It is not based on raw data like in supervised and unsupervised learning. The challenge for the algorithms is to solve the situation independently. This method is often used in games, robotics or navigation. The principle is that the algorithms always select the best option by trial and error and thus work out the ideal solution. This method is based on three components. The learner/the decision-maker, the environment, and actions [5].

2.2.2 Deep Learning

Deep learning (DL) is a subset of machine learning and will be used successfully in many fields like image classification, speech recognition, or genomics. The idea of DL is based on artificial neuronal networks (ANN), a group of interconnected nodes which are used to train and classify data. An ANN has three kinds of layer types: input, hidden and an output layer which is represented by nodes. The network starts and ends with input and output layers. The number of hidden layers between the input and output layer can vary significantly based on the dataset and the network architecture. The notes in a network will be linked to each other. Links represent the weight between the nodes [10, 25]. Deep learning initially distinguishes between two phases: Training and testing. In the training phase, the neural network is trained with the features and a label. In the second the test phase, test data will be used to validate the accuracy of the trained model. Different techniques of DL are provided. The most common techniques are the Multi-Layer Perceptron and the convolutional neuronal network.

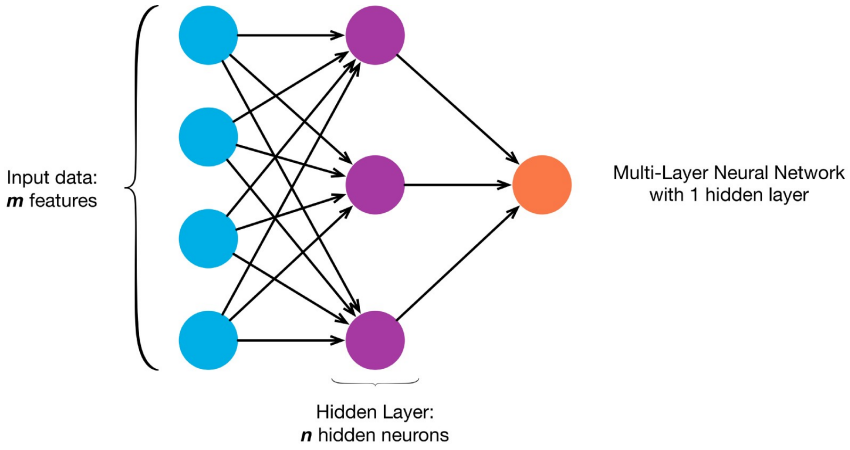


Figure 2.3 Multi-Layer Perceptron [11]

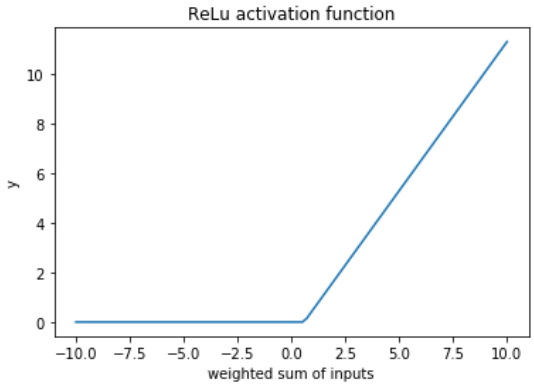
A Multi-Layer Perceptron (MLP) (figure 2.3) consists of a group of Single-Layer Perceptron (SLP) (figure 2.4). The structure of a perceptron is always the same and can be transferred to any number of perceptrons. It starts with the input $x_1, \cdots, x_m$, which is the output data on which the network is to be trained. At the end of each network, there is an output, which is calculated by first weighting the data and the input $w_1, \cdots, w_m$.

The weights are real numbers which represent the importance of the respective inputs to the output. In the next step, the weights are summed with the input:

$$\sum_{i=1}^{m}(w_i x_i) + bias$$

Subsequently, a bias b randomly selected at the beginning is added to the total. The last step to calculate the output is to use an activation function. The previously summed value is determined with the help of the activation function. The activation function can be any function, from simple to complex calculations. Classical functions include the sigmoid or a binary function [11]. The follow activation function is a ReLu-Function [14]:

$$f(x) = \begin{cases} x & if \ \sum wx + b \geq 0 \\ 0 & if \ \sum wx + b < 0 \end{cases}$$



The following figure 2.4 represent the procedure a single perceptron. Started with the Input (1), the Weights (2), Summ and bias (3), the Activation Function (4) and the Output (5).

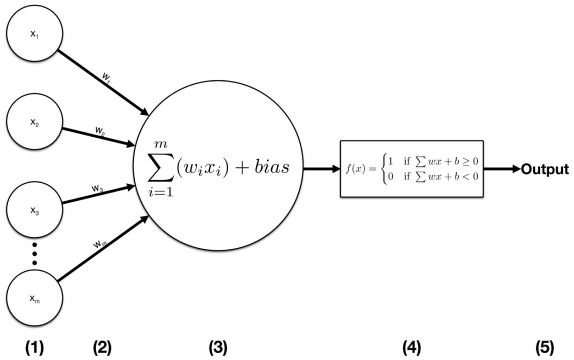


Figure 2.4 A Single Perceptron [11]

Convolutional neural networks (CNN) is a part if the neuronal networks based on two types of layers. The layer is called convolutional layers (CL) and Pooling layers and has shown excellent results in the field of image recognition. CL can be managed one-, two- or three-dimensional data as input. Images often will be transferred to pixels, in the case of side-channel attacks, the traces are one-dimensional. The kernel also called filters, convoluted the original shape into a different perspective to classify the data. Figure 2.6 describe the procedure between CL (Blue) an Kernel (Yellow). The Result is a smaller output matrix (Green) which represent the CL. It is possible to use multiple layers with different kernels sizes or CLs. (Gößе anpassen)

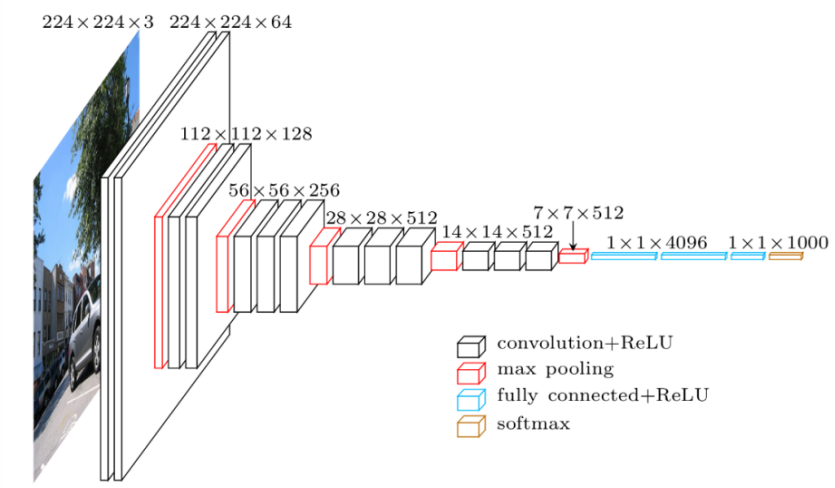


Figure 2.5: Convolutional Neural Network

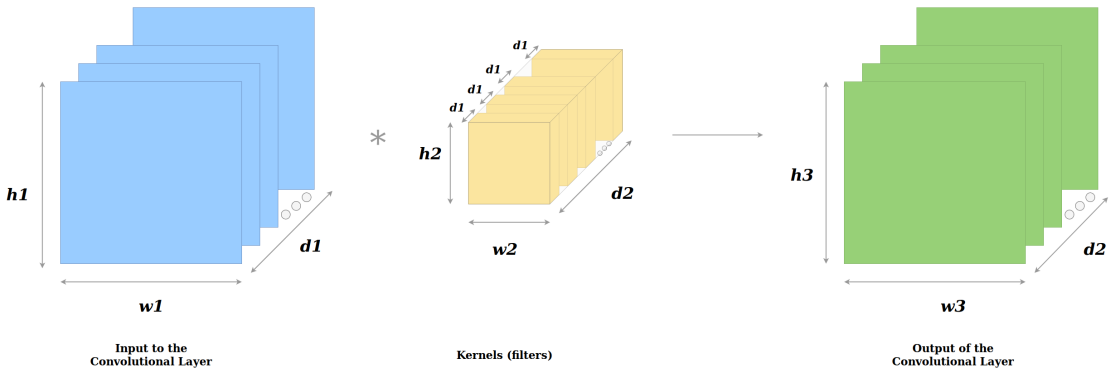
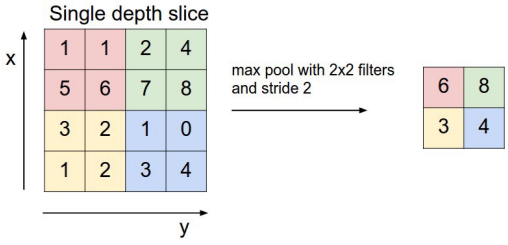


Figure 2.6 Convolutional Layer and Kernel

The Pooling Layer (PL) removes unneeded information while calculating the maximum or average of each cluster. The result is a new smaller matrix which represents the larger one. The example represents the Softmax-Function [15]:

$$P(y{=}j|\Theta^i){=}\frac{e^{\Theta^{(i)}}}{\sum_{j=0}^{k} e^{\Theta_k^{(i)}}}$$

$$where\ \Theta{=}W_0X_0{+}W_1X_1{+}\cdots{+}W_kX_k{=}\sum_{i=0,}^{k} W_iX_i{=}W^TX$$



CNN's are successfully used in learning objects in images and videos. In practice, depending on the data and area, it is necessary to decide which architecture and which network is suitable.

2.3 Hardware and Environment

This chapter deals with the used hardware and setup of the environment. In the first section 2.3.1 the used device and micro controller will be explained. The second section 2.3.2 describes the implementation of source code into the micro controller. In the last section 2.3.3 the environment an the setup for the measurement will be explained.

2.3.1 Microcontroller Board: Arduino Due

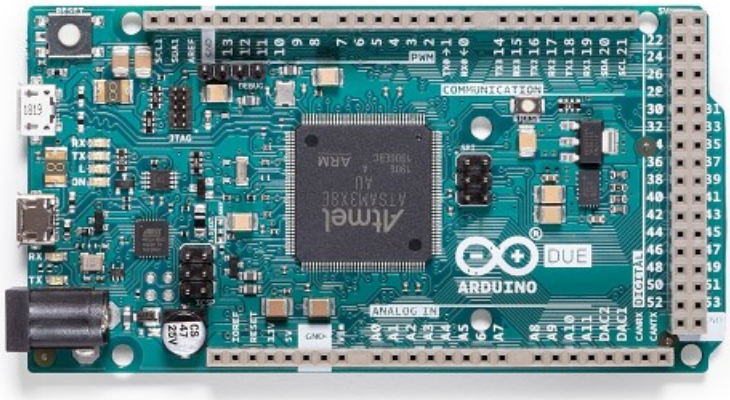


Figure 2.7 Arduino Due Board [1]

The Arduino Due board (Figure 2.7) is based on a 32-bit ARM core microcontroller. With 54 digital input/output pins and 12 analogue inputs. The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. The ARM CPU in Arduino Due is the same CPU which will be used in mobile phones to contactless payments and smart cards [13]. For research, the size of the CPU in a Arduino Board is much larger than in a mobile phone. The Board provides pins, a larger size of a CPU which make measurements and the implementation of AES more comfortable.The Arduino Device provides 32 pins. It is possible to access the pins in a C-Code implementation. The CPU of the Arduino will used in the common smart watches smart phones for mobile payments [1].

2.3.2 Microcontroller Implementation

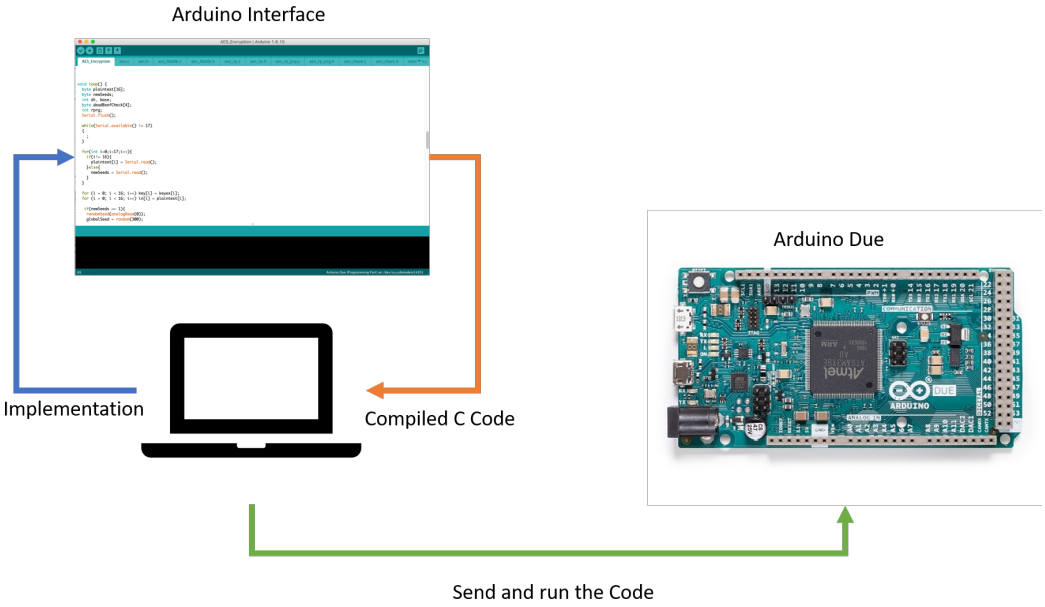


Figure 2.8 Process of Code Implementation into the Micro Controller

The Arduino Due provides a graphical user interface that allows C-Code to be executed directly from the device. The implementation of C is required to implement an encryption into the micro-controller. Further it is possible to implement encryptions with different kinds of countermeasure. The development of code directly for a microprocessor is different from normal development and has its challenges. The development proceeds as follows:

1. Integration of external libraries and pins
2. Compile the code
3. Send the compiled code to the micro controller and execute

Figure 2.8 shows the mentored process of the ingratiation of code into the micro controller. Once the code has been compiled and transferred to the device, it is impossible to intercept errors during runtime. Additionally, it is not possible to get information directly from libraries in Arduino. The Arduino interface always serves as an interface. A simple example is a print. The print command can print parameter or text in the console. Print commands which are implemented outside the Arduino interface cannot be displayed in the console and will be ignored. This can make debugging much more difficult because errors are harder to find.

The Arduino interface is provided as a script. Each Arduino Script provides two main functions, "setup()" and "loop()". The setup function will only run at the start of the device. The setup is necessary to allocate memory or to set up access to pins of the device. The loop function will run in a loop and can be used to read the input of a device. The Script is not limited in use of only these two functions. It is either possible to implement new functions or add additional libraries.

2.3.3 Environment and Measurement Setup

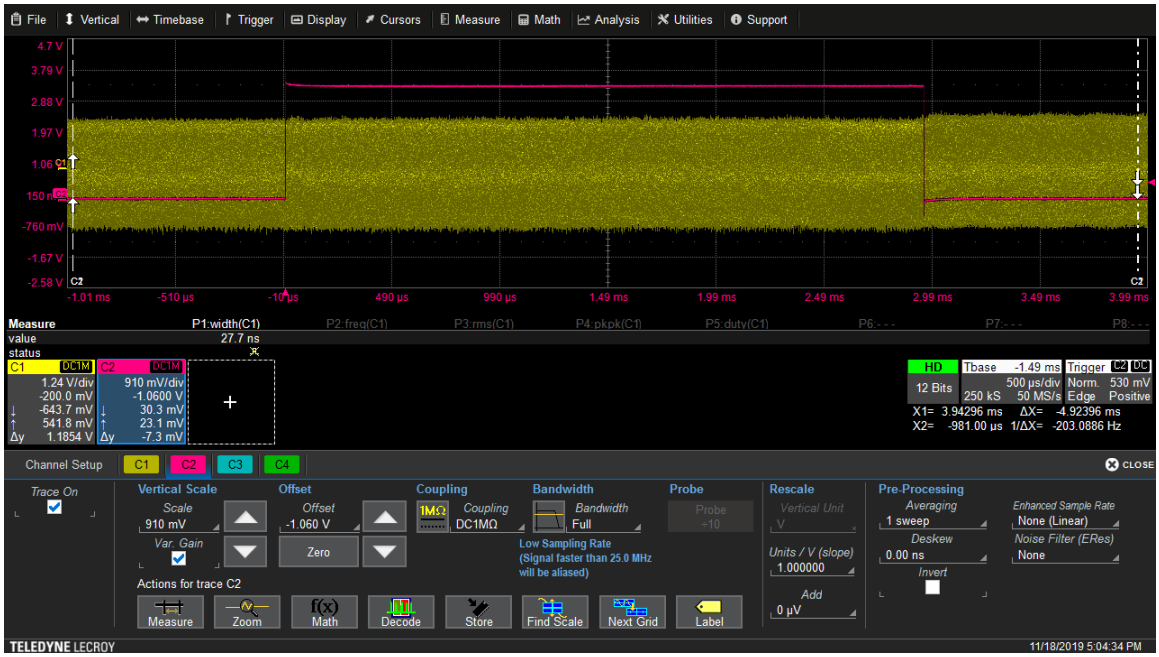


Figure 2.9 Oscilloscope GUI: Measurement of Traces (yellow) and Trigger (red)

An oscilloscope is used to measure the voltage on the CPU, with the help of a sensor. The sensor will be placed on the CPU, and some calibrations are required before it is possible to make correct measurements of traces. Figure 2.9 shows the graphical interface with the traces and the trigger. The yellow line represents a trace. The red line represents the leakage area. Only the high-frequency area of the traces are of interest and represent the calculation of the encryption. The low frequent parts of the trigger are noise and can be removed during the data preparation. The low and high-frequency will be triggered from a pin of the Arduino. The trigger is to mark the beginning and end of the encryption. Without a trigger it is hard to know when the encryption process will start and end. The signal of the trigger will be controlled in the implementation of the encryption script in the

Arduino implementation. Figure 2.10 shows the setup of the measurement. The sensor is placed on the CPU to measure the traces, the oscilloscope is connected to a pin to catch the signal of the trigger.

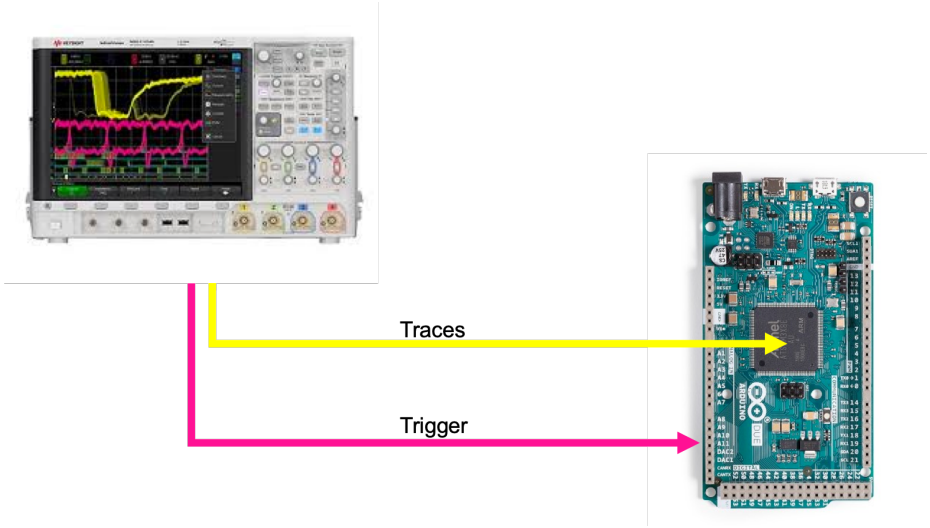


Figure 2.10 Oscilloscope Connection to the CPU and the Pins

The source code example 2.1 shows the implementation, written in C, to change the signal before and after the encryption. It is necessary to use two sensors. One is to measure the traces from the CPU and one to measure the the signal for the trigger. For the trigger a sensor is connected to one of the pins of the Arduino device, the other sensor is placed at the CPU so that the oscilloscope can measure the traces at the device.

```
1  //Innitalize the pin for the trigger
2  int LEDpin = 33;
3  pinMode(LEDpin, OUTPUT);
4
5  //Start high signal at the trigger
6  digitalWrite(LEDpin, HIGH);
7
8  //Run encryption while trigger is on hight signal
9  run_encryption(...);
10
11 //End high signal at the trigger
12 digitalWrite(LEDpin, LOW);
```

Source Code 2.1: Control of the Trigger During the Encryption

To send a signal from the Arduino device to the oscilloscope, a function called "digitalWrite()" is provided in C to send the signal to a pin as a low or hight signal. The function "digitalWrite()" has two parameters. The first parameter defines the address of the pin at the Arduino. The second parameter defines the signal type at the pin like high or low. By default the signal is low. Before the encryption starts the function "run_encryption()" will be called to set the signal to height. If the encryption is finished, the signal will be changed back to low.

The implementation of the trigger is just the first of the required steps. It is necessary to calibrate the settings at the oscilloscope. The most important configuration is the time sampling rate and the sequenced mode. The time sampling mode defines how many data points will be tracked within the target area. If the number of data points is too few, important information gets lost. If the number of data points increases the data sets is getting large. The figure 2.11 and 2.12 shows the trace with a low and a regular time sampling rate.

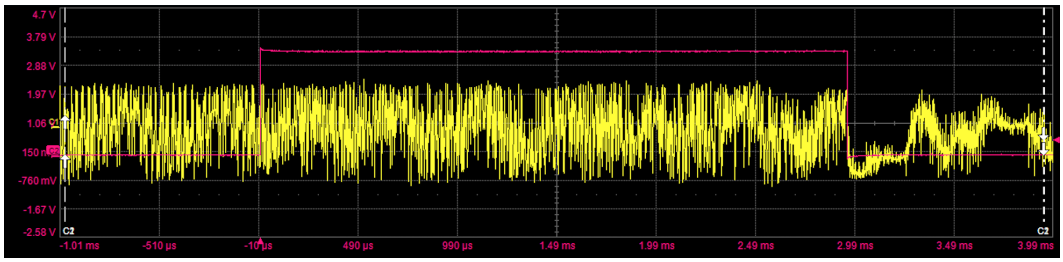


Figure 2.11 Low Time Sampling Rate

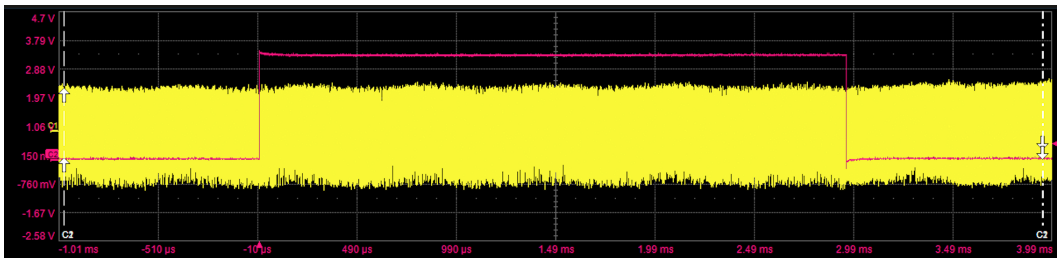


Figure 2.12 Normal Time Sampling Rate

The other necessary configuration is the sequenced mode. In default, the oscilloscope is measuring one single trace. In sequences mode, multiple measurements will be taken. The sequenced mode is not necessary for the quality of traces but increases the performance of the measurements. After the measurement, the single traces of a sequence can split in single traces. Figure 2.13 and 2.14 shows the measurement of a sampling rate of 10 an 50 encryptions.

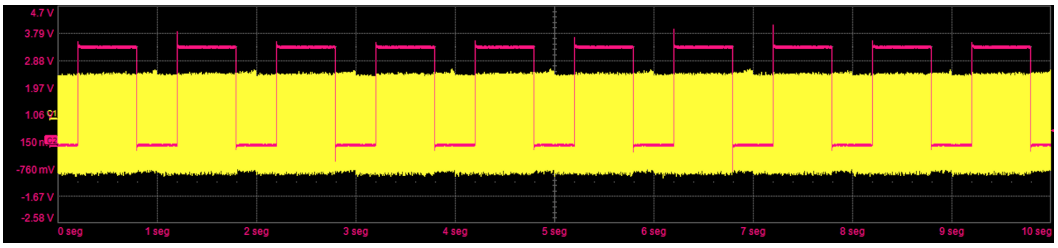


Figure 2.13 Ten Measurements in One Sequence

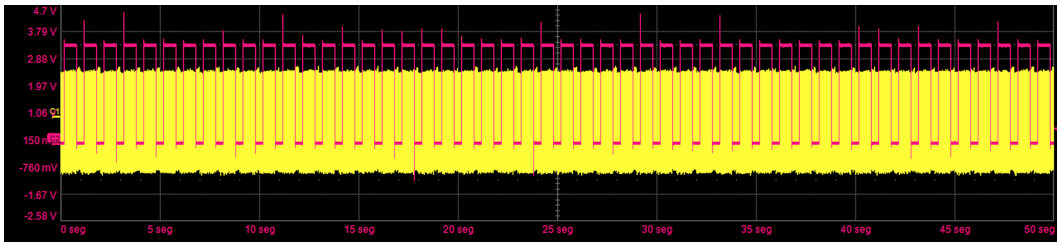


Figure 2.14 Fifty Measurements in One Sequence

The oscilloscope measures the calculation of the encryption and stores each trace. To measure the encryption at the CPU of the Arduino the oscilloscope controls the whole process of the encryption and measurement. The oscilloscope has an operating system which provides python scripts. With a script, plain text will be sent to the device (Arduino). As long as the device encrypts the plain text the oscilloscope collects the traces. After the successful encryption process the device sends the encrypted ciphertext back to the oscilloscope. This environment allows the fully automatic recording of traces during encryption process.

2.4 IT-Security

In this chapter all basics are explained on witch the research is based in the field of IT-Security. First of all the principle of substitution box and side channel attacks is explained. Then the principles of encryption methods and leakage assessment are discussed. Finally, the basics of common countermeasures and their implementations of side-channel attacks are explained.

At first in section 2.4.1 the encryption technique advanced encryption standard will be explained followed by the Rijndael substitution box in the section 2.4.2. In section 2.4.3 different kinds of protection modes of the advanced encryption standard will be explained in detail. Section 2.4.4 describes detailed how side-channel attacks works and how countermeasure in section 2.4.5 can help to prevent this kind of attacks. In section 2.4.6, a technique will be described to find unknown leakages. Section 2.4.7 explains the mathematical basics how the leakage detection works based on the current mathematical techniques, section 2.4.8 describes a new technique how leakage detection could work based on deep learning.

2.4.1 Advanced Encryption Standard

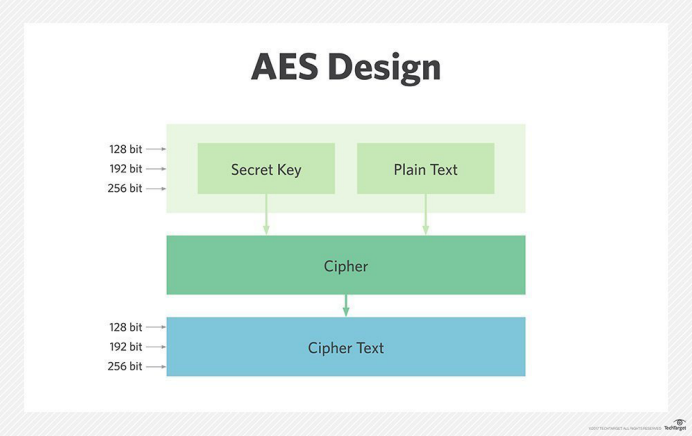


Figure 2.15 Design of the Advanced Encryption Standard [21]

Advanced encryption standard (AES) is a symmetric encryption method. Asymmetrical encryption methods use the same key (secret key) for encrypting and decrypting of the plain text. Figure 2.15 is a graphical example how the principle of AES encryption works. In case the key is known or it is possible to recover the key the intruder can encrypt the

whole conversation. In AES, a block cipher is used to encrypt messages. The size of the block cipher in AES has three sizes: 128-, 192- and 256-bits. In the following section 2.4.2 the principles of block cipher will be explained in detail. During the encryption process of the plain text, the plain text passes a certain number of rounds. Each round consists of substitution, transposition and mixing of the message (figure 2.16). The result is the encrypted message also called cipher text. The number of rounds depends on the size of the block cipher. A block cipher of 128 bits requires 10 rounds, 12 rounds for 192 bits and 14 rounds for 256 bits [22].

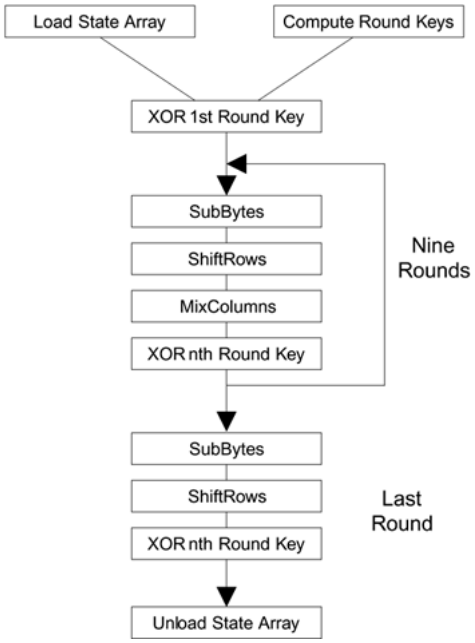


Figure 2.16 AES Rounds in Detail (128 bits) [7]

In detail, the encryption in AES with 128 bits is as follows [12, 9]. Each round in AES goes through the following 4 phases [3]. The last round in AES does not pass through all 4 phases and is therefore different from the previous phases. To decrypt the ciphertext, it has to run 10 rounds in reverse. The key to decrypt the message is the same key to encrypt the message. The following describes the four phases of the AES encryption in detail:

1. SubBytes:

The SubBytes transformation is a is a non-linear byte substitution. In this phase the plain text is encoded with the help of the S-Box. The exact procedure is explained in section [3].

2. ShiftRows:

In this phase the individual rows are shifted. The shifting is subject to exact rules which of the rows are shifted by how many positions [3].

3. MixColumns:

In the third stage, all columns are transformed by multiplying it with a specific multiplication polynomial [3].

4. AddRoundKey:

In the last stage, a round key is applied to the state by a simple bitwise EXOR. Every column is transformed by multiplying it with a specific multiplication polynomial [3].

2.4.2 Rijndael Substitution Box

$$\begin{bmatrix} s_0 \\ s_1 \\ s_2 \\ s_3 \\ s_4 \\ s_5 \\ s_6 \\ s_7 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{bmatrix} + \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

Figure 2.17 Rijndael Substitution Box (binary)

The substitution box (S-Box) is one of the basic components of symmetric key cryptography and is a key component of the Advanced Encryption Standard (AES). There are different variants of the S-Box, one of the most famous is the Rijndael AES S-Box (figure 2.17). It is considered to be resistant to algebraic attacks and is therefore preferred in AES. $[x_0, ..., x_7]$ represents the multiplicative inverse as a vector, $[s_0, ..., s_7]$ the output of the S-Box. The in- and output of the S-Box x_n, s_n has a size 8 and fits exactly 1 byte. The reason is, the encryption of the S-Box works byte wise, 8 bits represent 1 byte [4].

In AES two S-Boxes will be used, the first S-Box is used for the encryption process, the second S-Box represent the inverse of the first S-Box and is used for the decryption process. It is possible to convert the binary matrix to hexadecimal. The principles remain the

same. However, the hexadecimal notation simplifies the implementation and is common in practice. Table A.1 shows the hexadecimal S-Box for encryption. Table A.2 shows the inverse of the hexadecimal S-Box for decryption [16]. Each S-Box has one input and one output. The input represents the plaintext d which was calculated with the key k per XOR. The output s represents the encrypted message. The S-Box is calculated as follows:

$$s = sBox(k \oplus d)$$

The S-Box can also be considered as a converter. The index calculated from k and d is converted into the number placed inside the S-Box. The output returns the converted value which results from key and plaintext. The following example shows the implementation of the S-Box in python. The S-Box can also be implemented in any other language like C or C++. The source code 2.2 shows an example implementation of the S-Box in python. At the beginning in line 3, the S-Box is first implemented statically. Then in line 23 and 26 the key and plaintext are defined. In line 35, the plaintext is calculated with the key via xor and then called in the S-Box.

```
1  import numpy as np
2
3  # AES Sbox
4  AES_Sbox = np.array([
5  0x63, 0x7C, 0x77, 0x7B, 0xF2, 0x6B, 0x6F, 0xC5, 0x30, 0x01, 0x67, 0x2B, 0xFE, 0xD7, 0xAB, 0x76,
6  0xCA, 0x82, 0xC9, 0x7D, 0xFA, 0x59, 0x47, 0xF0, 0xAD, 0xD4, 0xA2, 0xAF, 0x9C, 0xA4, 0x72, 0xC0,
7  0xB7, 0xFD, 0x93, 0x26, 0x36, 0x3F, 0xF7, 0xCC, 0x34, 0xA5, 0xE5, 0xF1, 0x71, 0xD8, 0x31, 0x15,
8  0x04, 0xC7, 0x23, 0xC3, 0x18, 0x96, 0x05, 0x9A, 0x07, 0x12, 0x80, 0xE2, 0xEB, 0x27, 0xB2, 0x75,
9  0x09, 0x83, 0x2C, 0x1A, 0x1B, 0x6E, 0x5A, 0xA0, 0x52, 0x3B, 0xD6, 0xB3, 0x29, 0xE3, 0x2F, 0x84,
10 0x53, 0xD1, 0x00, 0xED, 0x20, 0xFC, 0xB1, 0x5B, 0x6A, 0xCB, 0xBE, 0x39, 0x4A, 0x4C, 0x58, 0xCF,
11 0xD0, 0xEF, 0xAA, 0xFB, 0x43, 0x4D, 0x33, 0x85, 0x45, 0xF9, 0x02, 0x7F, 0x50, 0x3C, 0x9F, 0xA8,
12 0x51, 0xA3, 0x40, 0x8F, 0x92, 0x9D, 0x38, 0xF5, 0xBC, 0xB6, 0xDA, 0x21, 0x10, 0xFF, 0xF3, 0xD2,
13 0xCD, 0x0C, 0x13, 0xEC, 0x5F, 0x97, 0x44, 0x17, 0xC4, 0xA7, 0x7E, 0x3D, 0x64, 0x5D, 0x19, 0x73,
14 0x60, 0x81, 0x4F, 0xDC, 0x22, 0x2A, 0x90, 0x88, 0x46, 0xEE, 0xB8, 0x14, 0xDE, 0x5E, 0x0B, 0xDB,
15 0xE0, 0x32, 0x3A, 0x0A, 0x49, 0x06, 0x24, 0x5C, 0xC2, 0xD3, 0xAC, 0x62, 0x91, 0x95, 0xE4, 0x79,
16 0xE7, 0xC8, 0x37, 0x6D, 0x8D, 0xD5, 0x4E, 0xA9, 0x6C, 0x56, 0xF4, 0xEA, 0x65, 0x7A, 0xAE, 0x08,
17 0xBA, 0x78, 0x25, 0x2E, 0x1C, 0xA6, 0xB4, 0xC6, 0xE8, 0xDD, 0x74, 0x1F, 0x4B, 0xBD, 0x8B, 0x8A,
18 0x70, 0x3E, 0xB5, 0x66, 0x48, 0x03, 0xF6, 0x0E, 0x61, 0x35, 0x57, 0xB9, 0x86, 0xC1, 0x1D, 0x9E,
19 0xE1, 0xF8, 0x98, 0x11, 0x69, 0xD9, 0x8E, 0x94, 0x9B, 0x1E, 0x87, 0xE9, 0xCE, 0x55, 0x28, 0xDF,
20 0x8C, 0xA1, 0x89, 0x0D, 0xBF, 0xE6, 0x42, 0x68, 0x41, 0x99, 0x2D, 0x0F, 0xB0, 0x54, 0xBB, 0x16
21 ])
22
23 # Plaintext
24 plaintext = [101, 102, 103, 104, 105]
25
26 # Key for encryption
27 key = 23
28
29 # List to to append encrypted plaintext
30 encryptedPlaintext = [ ]
31
32 # Loop through the plaintext list
33 for d in plaintext:
34
35     # XOR d with the key and call the S-Box
36     encryptedPlaintext.append( AES_Sbox[d^key] )
```

Source Code 2.2: Implemenatation of the S-Box

2.4.3 Modes of Operation

In AES, different kinds of modes exist to protect the encryption and the decryption process. The following six modes are the most common one:

1. ECB - (Electronic Codebook Mode)

The ECB mode de- and encrypt the plain text blockwise. With this technique, it is possible to parallelise the encryption or decryption because the blocks are independent from each other. The disadvantage of this technique is that the cypher text will not be blurred [2].

2. PCBC - (Propagating / Plaintext Cipherblock Chaining Mode)

In the PCBC technique, the plain text block will be XOR with an initialization vector. The initialization vector has the same size as the plain text. The disadvantage of this technique is that encryption cannot be paralleled. Also if one block will be damaged, it is unable to encrypt the whole cypher text [2].

3. CFB - (Cipher Feedback Mode)

The CFB mode is similar to the PCBC mode, in every round the data from the previous round will be encrypted and not the plain text. This technique makes it possible to use just one thread for the encryption, but the decryption of the blocks can be split in multiple threads [2].

4. OFB - (Output Feedback Mode)

In the OFB mode, a keystream of bits will be created. This keystream will be used for encrypting subsequent blocks. The encryption and decryption can be used only one thread at a time [2].

5. CTR - (Counter Mode)

In CTR a nonce represented as a unique number will be used once to encrypt the plain text bitwise. Even if one bit is corrupt the other bits can still get encrypted. The encryption and decryption can both be paralleled [2].

6. GCM - (Galois/Counter Mode)

A hash function in GCM guarantees the authenticity and confidentiality of the data. A binary Galois field defines the hash function. Additionally GCM offers the possibility of authentication assurance of the data [6].

2.4.4 Side-Channel Attacks

In a profiled side-channel attack (PA), the attacker has two identical devices and the attacker has full access to one of the two devices. The other one serves as an attacking device. The most common techniques for attacks are template attacks based on statistics or machine learning. The attack pattern is similar to supervised learning where a training set (Data) and a test set (Target) is used.

A Non-Profiled Attack (NPA) has just knowledge of the plain text and the traces, in contrast to a PA where the label is also known. In an NPA is the label unknown and has to be generated. We know the key k for every byte could be in the rage of $\{0, \cdots, 255\}$. For every trace, we generate a possible label l with each key and the known plain text d. In this case we have 256 possible labels for each byte: $l = Sbox(d \oplus k)$. To run a NAP existing different techniques are based on correlation or deep learning.

One of the most efficient attacks for NPA is the Correlation Attack (CPA). In the case of a CPA, the attack will be featurewise. For every feature F we create 256 labels L. L will be the combination on all existing keys $K = \{1, \cdots, 256\}$ with the given plaintext. So 256 correlations have to be checked per each feature and Label. If the correlation attack was successful the label with the keys correlates significantly with the label. Otherwise, the label does not include the key the dataset is protected against CPA.

2.4.5 Side-Channel Countermeasures

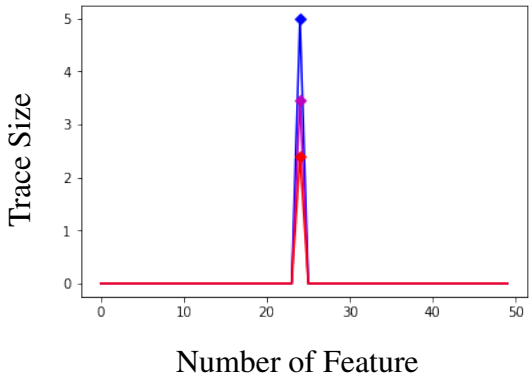


Figure 2.18 Traces with Noise

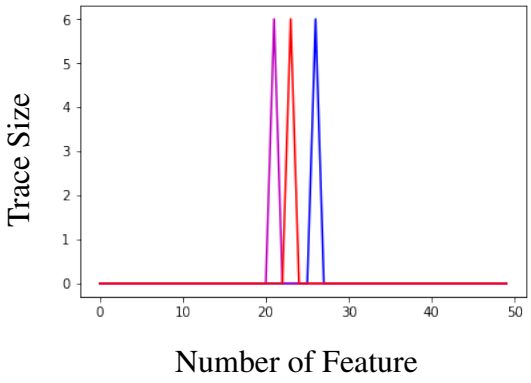


Figure 2.19 Traces with Random Delay

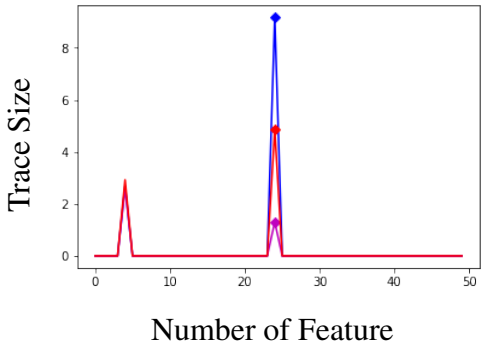


Figure 2.20 Traces with Masking

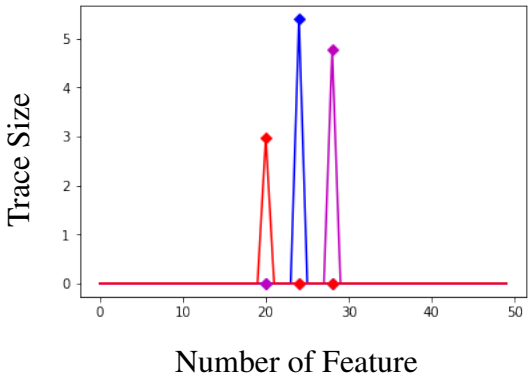


Figure 2.21 Traces with Noise and Random Delay

It is usual practice to protect traces from side-channel attacks. This chapter deals how this is done with countermeasure. In the following it is discussed which countermeasures were investigated. Some countermeasures can be taken to disguise the data and the signal. The most common countermeasures are noise, random delay, and masking.

Noise

Noise is one of the simplest countermeasures. In this case, the samples of the traces are shifted horizontally up or down by adding a random value. One of the most common methods is Gaussian noise. In figure 2.18 we can see how a sample is shifted horizontally for three different traces which using the gauss function as a countermeasure. The leakage is placed at position 25, the noise changes the point of leakage vertically.

Random Delay

The random delay is also one of the most common countermeasures. In this example, a random delay will be generated for each sample. Figure 2.19 shows how random delay can affect a sample. The leakage is placed randomly in the range from 20 to 28, the leakage will be changed horizontally.

Masking

Masking is a typical countermeasure. First, the plaintext will be encrypted with the key. Subsequently, a randomly generated value known as the mask will be XORed with the encrypted plaintext. The mask is also placed in the trace additionally protected with noise. It is possible to use several masks in a trace. Masking increases the security of tracers and is considered resistant to the traditional T-Test. If n stands for the number of masks, ao the implementation of n masks is called n-order. An attack with one mask is a first-order attack, two masks a second-order attack. Figure 2.20 shows traces protected by one mask. The mask is placed at position 5, and the leakage is placed at position 25. The leakage point changes vertically.

Combination of Noise and Random Delay

It is common practice to combine many different countermeasures. So noise and random delay may be combined. Figure 2.21 shows an example of the combination of the two countermeasures. The leakage is placed randomly in the range of 19 to 29 and changes vertically and horizontally.

2.4.6 Leakage Assessment

The idea of leakage assessment is to extract information from encrypted data. The current state of the art is to use universal statistical distinction like t-test or Pearsons x^2-test . It is necessary to use two groups of side-channel measurements which are independent of each other. An example is the encryption of two different plaintexts but with the same key. A t-test can be used to check if both groups are independent. If not, some information will be leaked, which allows identifying the two groups. The two groups must have a fixed plain text (fixed-vs-fixed). The plain text per group will not change but is different from the other group [26]. Security in smart cards has become more important since the advent of contactless payment. In the field of side-channel attacks, procedures are, therefore, being developed that make it possible to evaluate the security of devices and uncover possible vulnerabilities. Leakage Assessment offers the possibility to examine security measures for leakages and to improve them [23].

2.4.7 Student's T-Test

In leakage assessment t-test is a common technique to identify the dependency between two datasets. We denote two datasets Q_1 and Q_2 with a cardinality by n_0 and n_1. So μ_0 , μ_1 represent the means and s_0, s_1 the standard derivation. The t-statistics and the degrees of freedom v will be computed by the following formulas [26]:

$$t = \frac{\mu_0 - \mu_1}{\sqrt{\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}}} \qquad v = \frac{\left(\frac{s_0^2}{n_0} + \frac{s_1^2}{n_1}\right)^2}{\frac{\left(\frac{s_0^2}{n_0}\right)^2}{n_0 - 1} + \frac{\left(\frac{s_1^2}{n_1}\right)^2}{n_1 - 1}}$$

After the calculation of t and v the confidence p to accept the null hypothesis can be estimated with the Students's t probability density function where $\Gamma(.)$ denotes the gamma function [26, 24, 18] .

$$p = 2\int_{|t|}^{\infty} f(t, v)\, dt \qquad f(t, v) = \frac{\Gamma\left(\frac{v+1}{2}\right)}{\sqrt{\pi v}\,\Gamma\left(\frac{v}{2}\right)} \left(1 + \frac{t^2}{v}\right)^{-\frac{v+1}{2}}$$

It is usual practice to set the threshold to $|t|$> 4.5 for simplicity. If the threshold is $|t| > 4.5$ and $v >$ 1000 the confidence p to accept the null hypothesis is smaller than 0.00001 which is equivalent to 99.999% confidence. In this case the dataset are not drawn from the same population [18] [26].

2.4.8 Deep Learning Leakage Assessment

In Deep Learning Leakage Assessment (DL-LA) deep learning will be used instead of a classical T-Test. The DL-LA consists of the following steps:

1. Create two (independent) data sets
2. Mix the two datasets and split into test and train data
3. Training of the neural network
4. Analysis of the model with the test data by sensitivity analysis
5. If the sensitivity analysis is positive leakages are found

The problem of a T-Test is that n-order (n is the number of used masks) leakages are much more difficult to detect. The idea of deep learning is that each data points of a trace will be combined in a network. The combination makes it possible to find easier leakages with deep learning. Also, in DL-LA, two groups of data are required (fixed-vs-fixed). Each group will be labelled. The first group will be labelled with a 0, the second group with a 1. Both groups will be randomly mixed and stratified split by the label in a test and train dataset.

After splitting of the dataset the neuronal network will be trained with the training dataset. Afterwards, the model will be used to classify the test dataset. After the training sensitivity analysis (SA) is required to detect leakages in the traces. With this technique, all points of the trace are examined. The sensitivity is calculated with the following formula [26]:

$$s_i = \left| \sum_j \frac{\partial y_0}{\partial x_i} \cdot X_i^j \right|$$

More specific, x_i describes the i-th input of the neuronal network, y_0 the first coordinate of the output. X_i^j describes the i-th input of a trace j. The sensitivity calculations are based on the chain-rule. All gradients of each layer are considered in the calculation [26].

The idea behind the calculation of the sensitivity is that it can be used as an indicator of whether a neural network has learned something. The higher the sensitivity per feature is, the more it can be assumed that something has been learned at this point. In the case of two independent encrypted traces, no learning should take place at any point. If this is the case, it is assumed that there is a leakage.

Chapter 3

Research

This chapter deals with the research. The research is split into two parts. The first part 3.1 concerns the simulated traces. Eight datasets were prepared to examine the impact of different countermeasures. Subsequently, all datasets were analysed with the T-Test and DL-LA. In the second part 3.2 of the research, an unprotected and protected AES encryption will be implemented in a microcontroller. After the implementation, traces from the CPU will be measured and afterwords analysed with the Students T-Test and DL-LA. The attached disk contains the source code of all experiments, as well as the implementation of the AES encryption, and the configuration to measure the traces. Most of the implementation of the functions and algorithms, had to be done manually because no standard libraries exist.

3.1 Experiment: Simulated Traces

In research with side-channel attacks, it is common practice to use simulated traces for the experiments first and then use real traces. The advantage of simulated traces is that they can be generated easily and quickly with different levels of countermeasures. A disadvantage that they are only generated artificially and can, therefore, behave differently in experiments. Section 3.1.1 deals with the data preparation and generation of the simulated traces. The experiments are classified in eight sections (section 3.1.2, section 3.1.3, section 3.1.4, section 3.1.5, section 3.1.6, section 3.1.7, section 3.1.8, section 3.1.9). These experiments analyse the impact of the different kinds of countermeasure. In section 3.1.10 the results of the simulated traces are summarized.

3.1.1 Data Preparation

Dataset	Countermesure
1.	Noise
2.	Random Delay
3.	Noise and 1 Mask
4.	Random Delay and 1 Mask
5.	Noise and 2 Masks
6.	Random Delay and 2 Masks
7.	Noise and 3 Masks
8.	Random Delay and 3 Masks

Table 3.1: Datasets of Simulated Traces and Countermeasures

During the data preparation, various data sets with different countermeasures are generated. Table 3.1 shows the generated data sets with different kinds of countermeasures. Artificial traces can be used immediately after the generation. It is not necessary to modify or prepare the datasets since they are generated. The implementation of the countermeasures was dynamic. This allowed countermeasures such as noise and random delay to be combined as desired. The source code 3.1 presents the parameter of the countermeasure before the traces are generated. Line 2 and 3 defines the number of features and traces per dataset. Line 9 to 10 are the implementation of noise, line 13 to 21 represent the implementation of the random delay.

```
1  # Define lengh for the data set
2  data_leng = 400000
3  numberOfSamples = 50
4
5  #Random Plaintext
6  FixedPlaintext = True
7
8  # Define gauss niose
9  noiseStart = 0.0
10 noiseEnd = 5.0
11
12 # Definer random delay shift range
13 activateJitter = False
14
15 if activateJitter == True:
16     # Value for jitter
17     jitterLeft = -4
18     jitterRight = 4
19 else:
20     jitterLeft = None
21     jitterRight = None
```

Source Code 3.1: Paramter to generate the Simulated Traces

The following source code 3.2 shows the implementation of the leakage point. A key and a plaintext are passed to the S-Box. Additionally noise is added to the leakage point as a countermeasure. The source code 3.3 represents the output of the generated label and the leackage point. The label is the output of the S-Box and the leakage is the label with generated noise additioned randomly. The label is needed to differentiate the data sets from each other.

```
#Generate leaked data with key=23 and labels
leakedData = []
#fixed key
key = 23
labels = []
for d in plaintext:
    leakedData.append(bin((AES_Sbox[d^key])).count("1")+gauss(noiseStart, noiseEnd))
    labels.append(bin(AES_Sbox[d^key]).count("1"))
print("leakedData:",leakedData[0:5],"\nlabels:",labels[0:5])
```

Source Code 3.2: Simulated Traces: Implementation of the Leakage Point with Noise

```
Output:
Labels: [2, 2, 2, 2, 2]
Leakage Points: [3.1814221129750884, 0.05914570915085138, 4.789588526536645, -5.63032353491941, 3.0586068205024914]
```

Source Code 3.3: Simulated Traces: Label and Leakage

3.1.2 Dataset 1: Noise

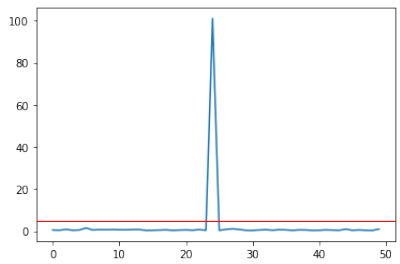


Figure 3.1 T-Test with Noise

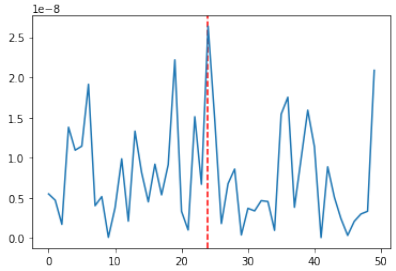


Figure 3.2 Deep Learning with Noise

This experiment shall simulate an unsecured trace with noise. The trace has a length of 50 features. Two datasets were created with the same key but different plaintext. The leakage point was created at position 24. A data set of 100.000 traces was created for the experiment.

T-Test

First, a T-test was performed, followed by a deep learning test. The T-test (figure) clearly shows a leakage at location 24 that exceeds the 5.0 make significantly. The value reaches values of up to 100, which is a strong indication of a leakage, in the other positions the value is just above 0. The mark (red) in the T-test indicates the 5 percent threshold that must be exceeded in order to demonstrate significantly that there is a leakage. Already from a quantity of 10.000 traces, leaks could be found with the help of the T-Test.

Deep Learning

The sensitivity in the deep learning test shows several peaks at position 7,18, 24 and 49. The highest peak is at position 24 represent the leakage point (red line). The other points 7, 18 and 49 are random data without any informations and leaks. For deep learning a dataset with the size of 10.000 was used.

3.1.3 Dataset 2: Random Delay

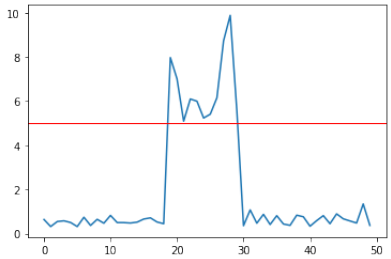


Figure 3.3 T-Test with Random Delay

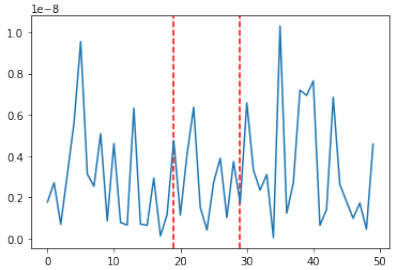


Figure 3.4 Deep Learning with Random Delay

In this experiment unprotected traces are generated with random delay. The trace contains 50 features and the leakage point is placed randomly in range [19 - 29]. The red mark in figure 3.3 shows the threshold at 5 which must be exceeded. The two red lines in figure 3.4 indicate the area where the leakage point is placed. A data set of 100.000 traces was created for the experiment.

T-Test

In the T-Test 3.3, this is clearly visible, and the value exceeds the value 5 at all points in the leakage range. At the highest point, the value 10 is reached. all other points have no noticeable increase.

Deep Learning

The plot3.4 shows the sensitivity of the deep learning model after the training with random delay as countermeasure. The red lines represent the area were the leackage point is randomly placed. The plot shows at position 5 and position 35 hight peaks. The leakage range starts from 19 and end at 29. Specially in the area of the leakages there are no visible peaks.

3.1.4 Dataset 3: Noise and 1 Mask

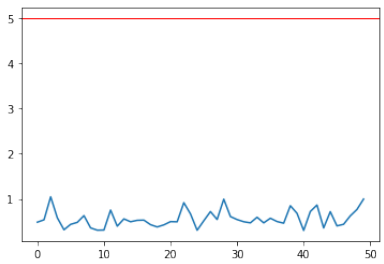


Figure 3.5 T-Test with Noise and 1 Mask

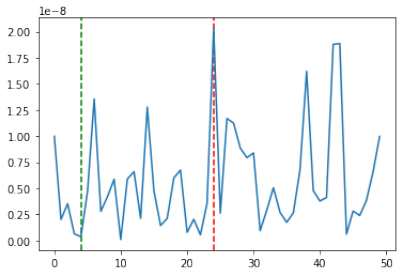


Figure 3.6 Deep Learning with Noise and 1 Mask

In this experiemnt a simulated trace is protected with a masking. This kind of masking is often used to protect traces with AES encryption. In addition to the masking, noise is also used as countermeasure. The leakage point is placed at position 24 and the masking point at position 4 in the trace. The red mark in figure 3.5 shows the threshold at 5 which must be exceeded. The two red lines in figure 3.6 indicate the leakage point, the green line represents the masking point. A data set of 100.000 traces was created for the experiment.

T-Test

In the T-Test no leakage point is found. This result was expected because it is not possible to find leakages with a simple T-Test. A possible solution is to multiply every data point in the trace with each other and run the T-Test again. This technique would be called second order attack. The second order attack needs a loot of resources and time and was not part of this research.

Deep Learning

The plot 3.6 shows the sensitivity of traces with one protected mask and noise. The plot shows two high peaks at position 24, 41, 42, and 43. The highest peak at the position 24 represents the leakage. The mask at position 5 is one of the lowest points and gives no hints about any leaks. Every other point in the trace is random.

3.1.5 Dataset 4: Random Delay and 1 Mask

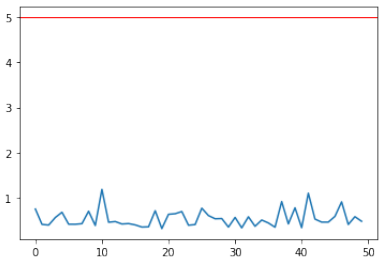


Figure 3.7 T-Test with Random Delay and 1 Mask

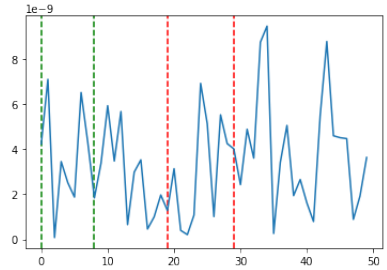


Figure 3.8 Deep Learning with Random Delay and 1 Mask

This experiment simulates traces which are protected by masking and random delay. As a countermeasure for both points random delay was used. The red mark in figure 3.7 shows the threshold at 5 which must be exceeded. The two red lines in figure 3.8 indicate the range of the leakage points [19 - 29], the green lines represent the range of the masking points [0 - 9]. A data set of 120,000 traces was created for the experiment.

T-Test

In the T-Test no leakage point is found. This result was expected because it is not possible to find leakages with a simple T-Test. This test was made to proof that the trace has no first order leakage. If this test would be positive with masking it is a sign that something is wrong with date dataset.

Deep Learning

The 3.8 presents the sensitivity of a dataset with leakage, one mask and random delay. The area between the green marks represents the mask, the area between the red marks the leakage. The sensitivity has two points with higher peaks, point 34 and 43. Both data points are not in the masking or leakage area. The leakage and masking area give no information about any leakages.

3.1.6 Dataset 5: Noise and 2 Masks

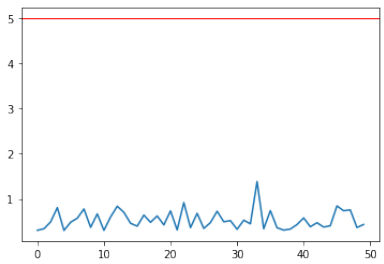


Figure 3.9 T-Test with Noise and 2 Masks

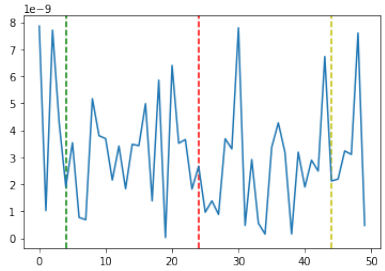


Figure 3.10 Deep Learning with Noise and 2 Masks

This experiment simulates traces which are protected by two masks. This form of protection is already more complex and more difficult to be attacked. To implement these protection 3 data points were modified in the simulated traces. The red mark in figure 3.9 shows the threshold at 5 which must be exceeded. The red line in figure 3.10 indicates the leakage point at position 24, the green lines represent thefirst mask at point4 and the yellow point at position 44 represent the second mask. A data set of 200,000 traces was created for the experiment.

T-Test

In the T-Test no leakage point is found. This result was expected because it is not possible to find leakages with a simple T-Test. This test was made to proof that the trace has no first order leakage. If this test wold be positive with masking it is a sign that something is wrong with date dataset.

Deep Learning

In this experiment a dataset with 2 masks and 1 leakage point was used for the deep learning. The the figure 3.10 represent the sensitivity. The green and yellow marks are the masks and the red mark is the leakage. At position 3, 4, 29 and 48 peaks are higher but none of the peaks match the masks or the leakage. Of the whole trace no leakage or masks can be detected with sensitivity.

3.1.7 Dataset 6: Random Delay and 2 Masks

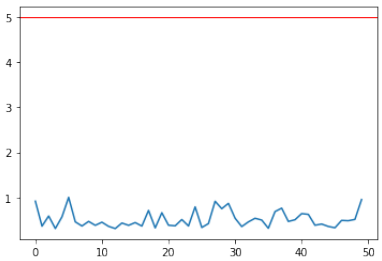


Figure 3.11 T-Test with Random Delay and 2 Masks

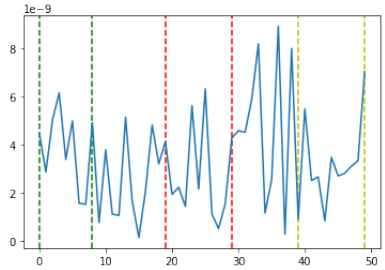


Figure 3.12 Deep Learning with Random Delay and 2 Masks

This experiment simulates traces which are protected by two masks and random delay. The red mark in figure 3.11 shows the threshold at 5 which must be exceeded. The red lines in figure 3.12 indicates the leakage points range [19 - 29], the green lines represent the range of the first mask [0 - 9] and the yellow lines the range of the second mask at [39-49]. A data set of 200,000 traces was created for the experiment.

T-Test

In the T-Test no leakage point is found. This result was expected because it is not possible to find leakages with a simple T-Test. This test was made to proof that the trace has no first order leakage. If this test wold be positive with masking it is a sign that something is wrong with date dataset.

Deep Learning

The dataset for this experiment was protected with two masks, one leakage point and random relay. The masks were placed between the green and yellow area and in the read area the leakage point. The sensitivity shows the peaks at the points 34, 36 and 39. All three points are not in the range of the leakage or masks. The traces give no information of the real leakages in the traces.

3.1.8 Dataset 7: Noise and 3 Masks

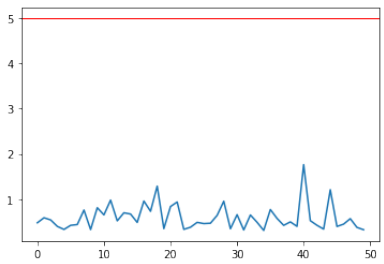


Figure 3.13 T-Test with Noise and 3 Masks

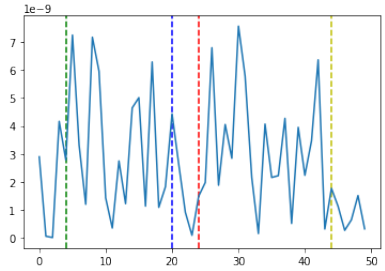


Figure 3.14 Deep Learning with Noise and 3 Masks

This experiment simulates traces which are protected by three masks and noise. The red mark in figure 3.13 shows the threshold at 5 which must be exceeded. The red lines in figure 3.14 indicates the leakage points at position 24, the green line represents at position 4 the first mask and the yellow line the second mask 44 and the third mask as blue line is placed at position 20. A data set of 1.800.000 traces was created for the experiment.

T-Test

In the T-Test no leakage point is found. This result was expected because it is not possible to find leakages with a simple T-Test. This test was made to proof that the trace has no first order leakage. If this test would be positive with masking it is a sign that something is wrong with date dataset.

Deep Learning

The Figure 3.14 shows the sensitivity of the dataset with three masks (green, blue, yellow) and a leakage point (red). The dataset has multiple high peaks but none of this peaks represent a mask or leakage. If we take a look at the whole plot it is not possible to detect any leaks masks based on the sensitivity.

3.1.9 Dataset 8: Random Delay and 3 Masks

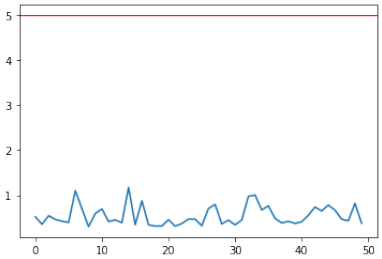


Figure 3.15 T-Test with Random Delay and 3 Masks

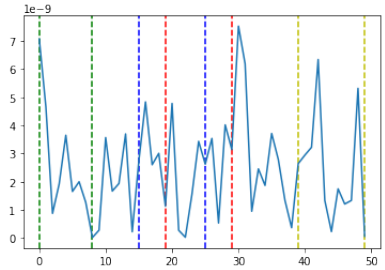


Figure 3.16 Deep Learning with Random Delay and 3 Masks

This experiment simulates traces which are protected by three masks and noise. The red mark in figure 3.15 shows the threshold at 5 which must be exceeded. The red lines in figure 3.16 indicates the leakage points range [19 - 29], the green lines represent the range of the first mask [0 - 9] and the yellow lines the range of the second mask at [39-49]. The blue lines represent the area with the third mask [16 - 24] the leakage point overlaps with one of the masks. A data set of 1.800.000 traces was created for the experiment.

T-Test

In the T-Test no leakage point is found. This result was expected because it is not possible to find leakages with a simple T-Test. This test was made to proof that the trace has no first order leakage. If this test wold be positive with masking it is a sign that something is wrong with date dataset.

Deep Learning

The sensitivity 3.16 is based on a dataset with tree masks, a leakage point and random delay. The areas green, blue and yellow represent the masks, the read area the leakage. In the yellow area are two higher peaks, the highest peak at position 31 and not a part of the masks or leakage points.

3.1.10 Summary: Simulated Traces

The series of tests with simulated traces indicate that they are only useful to a limited extent for DL-LA. Already at the beginning the sensitivity shows leaks inaccurately even in the unsecured traces. However, since the data were created artificially the leaks are manifest only at one data point. Furthermore the random delay archived significantly worse results than traces protected only with noise. The T-Test with random delay scored significantly better than that with DA-DL. It also showed that a leakage point with deep learning was detected where the T-test did not work anymore. In this research, other points were also shown as leakages although they are random values.

In all other experiments, no significant leakage could be found with the help of DL-LA this may depend on various factors. The parameters for the neural network may not be adapted optimally. Significantly more data may have to be used for deep learning, even though this was simulated data. The experiments have also shown that DL-LA reacts differently to specific countermeasures. The random delay seems to be a more significant challenge for the neural network than noise.

3.2 Experiment: Real Traces

This experiment investigates how the T-Test and DL-LA behave on a CPU used for virtual smart cards. At first in section 3.2.1, an protecterd and unprotected AES encryption hast to be implemented in a microcontroller. Afterwards in section 3.2.2 the right position at the CPU for the measurements must be located. In the next step in section 3.2.3, the traces will be analysed as to the quality and the information gain. The next two sections: 3.2.4 and 3.2.5 deal with the experiments. T-Test and DL-LA will be used to find leakages in the traces. The last section 3.2.6 is the summary of the results in this chapter.

3.2.1 Microcontroller Implemenatation

For the research the AES encryption has to be implemented in a microcontroller with C because it is very efficient in performance and close to the hardware. Especially for the measurement of traces performance is important, since milliseconds affect the size and the measurement duration. Lack of performance can have different side effects: measurements lasting several days, the collected data reaching a size of several terabytes. The implementation was performed in an Arduino Due. No operating system was installed on the device and the implementations are done via the developer interface provided by arduino. A correct, secure and high-performance implementation of AES is very complex. Therefore, a secure C library was used which provides the AES encryption. However, the library is not designed directly for microprocessors and therefore needs to be modified in some places to be used for the Arduino Due.

The main challenge with microprocessor implementations is the limited debugging capacity. After the code is executed in the microprocessor exceptions or errors will ne be possible to debug or to trace. For the experiments two AES encryptions are implemented in the microcontroller: an unprotected AES encryption and a protected AES encryption with random delay and one mask.

The first implementation is an unsecured AES encryption. During the encryption process no countermeasure is used. This represents a rudimentary encryption process with an S-Box. The second implementation represents a secured AES encryption with random delay and masking. Every encryption process has random parameters, even if the same plain text is used multiple times. During the data preparation it was necessary to execute exactly the same encryption process multiple times. The solution is to control the random parameter and use the same randomness for the same plain text. Within the encryption process the random parameters in the framework are accessible from the arduino inteface and can be

controlled during the encryption process. For the implementation the AES library "Higher Order Countermeasures for AES and DES" at github (https://github.com/coron/htable) was used. During the research the library was modified for the usage of the microcontroller in the Arduino Due environment.

For the randomness, the marsaglia xorshift generator [17] was used and implemented in C. This generator is known for the performance and the quality of randomness. The Source Code 3.4 shows the implementation of the random generator. By default, the random generator has three static numbers x, y and z (line 2). The random function "xorshf96(void)" is called multiple times during the encryption process. Depends on the static parameters the randomness can be controlled, and it is possible to reproduce multiple time the same randomness. During the shifting procedure the randomness will be created depending on the initial parameters x, y, z (line 7 to 13).

```
1  //random seeds
2  static unsigned long x=123456789, y=362436069, z=521288629;
3  static unsigned int randcount=0;
4
5  //random function
6  unsigned long xorshf96(void) {
7    unsigned long t;
8
9    randcount++;
10
11   x ^= x << 16;
12   x ^= x >> 5;
13   x ^= x << 1;
14
15   t = x;
16   x = y;
17   y = z;
18   z = t ^ x ^ y;
19
20   return z;
21 }
```

Source Code 3.4: Implemenatation of the Marsaglia Xorshift Random Generator

3.2.2 Grid Search

Figure 3.17 Heatmap with 200 measured correlation points at the CPU

Each CPU type has a different architecture. Calculations can be executed at different positions of the CPU. How the architecture of a CPU is functioning is usually only known by the manufacturer. In order to figure out where the calculations are performed on the CPU, the AES encryption is executed unsecured in 200 different positions. A correlation attack follows each of the 200 results. The better the position on the CPU, the higher the results of the correlation attack. Subsequently, all 200 results are plotted as a heat map. The a heat map shows visually which positions are best suited for the measurements.

First results showed that the correlation at each point is much too low. It can be depending on the CPU and its architecture. Furthermore, the measuring instruments can influence the quality of the measurements. To improve the quality of the traces averaging is a common technique. The same calculation is carried out several times, and then the average is calculated. This procedure minimises noise and external influences and improves the quality of the traces. The results show that the correlations after averaging are much stronger. Figure 3.18 shows a heatmap with the individual correlations. Figure 3.17 shows two areas suitable for measurements.

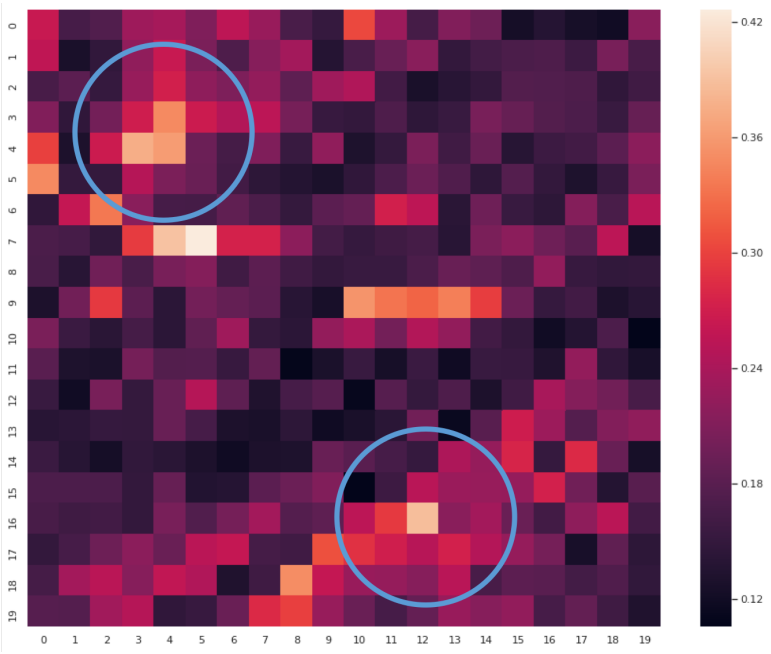


Figure 3.18 Heatmap with two possible positions for the future measurement

To perform a grid search on the CPU technical requirements were necessary because the manual placement of the sponsor on the CPU can become very inaccurate at 200 points. Also it must be possible to place the sensor in the same place for further measurements. Therefore the whole process of scanning and controlling the sensor was automated. For this purpose, a script to measure traces and control the position of the sensor has to be implemented. The scrip has to make the measurements and the movement of the sensor on the CPU fully automated. Every position of the sensor will be stored in the form of coordinates, together with the measured traces. This procedure makes it possible to reproduce every position of the sensor to run future experiments at these specific positions.

The Source Code 3.5 shows the initialization of the sensor and the grid on the CPU. In Line 4 and 5 define how many steps the sensor should move in the x and y direction. The sensor is initialized in line 9 and the connection is established. In line 12 and 15 start and end coordinates are defined. The sensor performs and saves the measurements automatically at each of the defined positions.

```
import EM_station

#Steps in x and y direction
no_steps_x = 20
no_steps_y = 20
no_steps_z = 0

#initalisierung of the sensor and the directions
myEM = EM_station('COM31', analysis_steps_x=no_steps_x, analysis_steps_y=no_steps_y, analysis_steps_z=no_steps_z)

#Start position on the CPU
myEM.set_pos_A([-123000, 144000, -97000])

#End position on the CPU
myEM.set_pos_B([-75000, 197000, -97000])
```

Source Code 3.5: Position controll of the Sensor at the CPU

Figure 3.19 shows the sensor placed on the CPU of the Arduino Due board. The green cable (right) is connected to the ground and the black one (left) is connected to the pin with the trigger. In figure 3.19 the sensor is placed at the marked position in figure 3.18 to measure the traces.

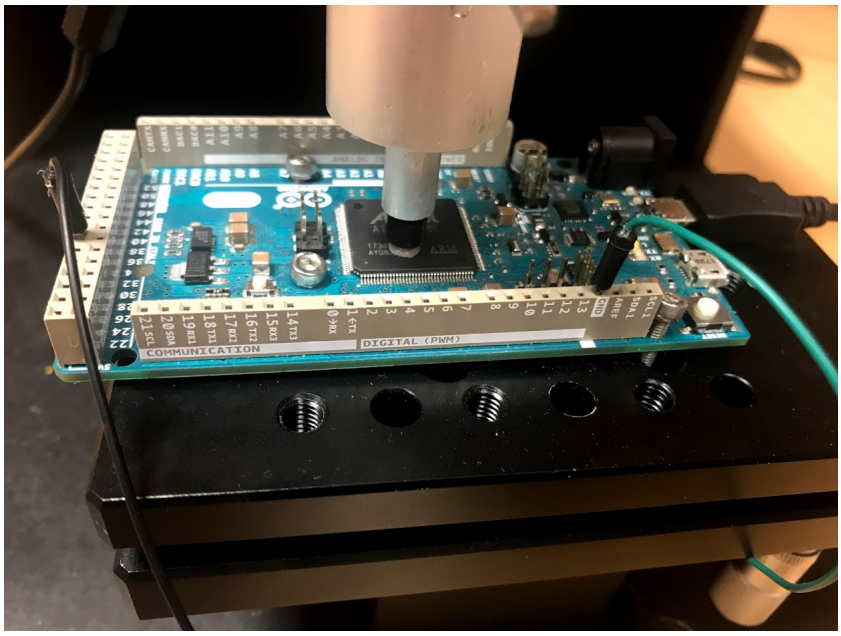


Figure 3.19 Arduino Due with a Sensor placed on the CPU for the Measurements

3.2.3 Signal to Noise Ratio

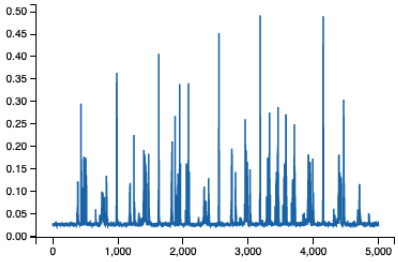


Figure 3.20 SNR Unprotected Traces

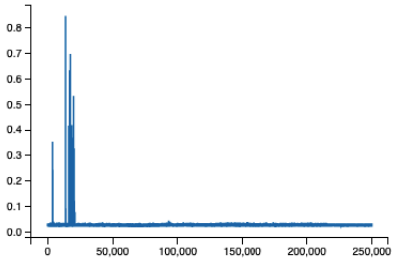


Figure 3.21 SNR Protected Traces - 10 Rounds AES

An SNR (signal to noise ratio) can be used to investigate the information gain of data. A dataset to calculate the SNR contains the measured traces during the encryption of the plaintext with AES. The plaintext for the encryption is chosen randomly between 0 and 254 (255 classes). The accuracy of SNR depends on the size of the dataset. If the dataset is to small the 255 classes contains not enough individual traces and the SNR gets inaccurate. Consequently as many traces as possible have to be measured to get a significant SNR. The SNR is calculated by the flowed formula [20, 16]:

$$SNR = \frac{variance_y\left(mean_i\left(L_y^i\right)\right)}{mean_y\left(variance_i\left(L_y^i\right)\right)}$$

The traces i in the dataset L are be grouped by the label y. The $variance$ of the $mean$ will be divide by the $mean$ of the $variance$ of every group L_i. Figure 3.20 reveals the SNR of the traces with an unprotected AES encryption. The information gain is clearly viable at the high peaks. At figure 3.21 the SNR was calculated of an dataset with a protected AES encryption with random delay and one mask. The information gain is only visible in the first of the ten rounds of the AES encryption.

During the calculation of the SNR, it was shown that the standard functions like the mean, the sum and the variance could not be applied to the whole data set. The problem is that the amount of data is too large to compute the functions. The first dataset consists of 20.000 traces and 5000 features per trace. The second data set consists of more than 40.000 traces with 250.000 features each, the size of the larger data set was about 60 gigabyte. When the data reaches a certain level of complexity and size, conventional programming functions can no longer be used for the calculations then take too long or are aborted. To solve this problem, functions were implemented manually, which read and calculate the data

line by line. The line-by-line reading is technically slower than loading a whole data set into memory but has the advantage that even vast amounts of data can be processed and calculated.

The example figure 3.22 shows in pseudo code the implementation of the SNR function. At line 6 and 7 the dataset will be read row wise and stored temporary in the variable buffer. From line 10 ro 14 the sum and the product will be estimated row wise. It is not possible to use the default function "math.SUM()" and "math.PROD()" for the calculations. At line 18 to 24 the mean, and variance will be calculated based on the sum and product from line 12 and 14. The rowwise calculation of data is limited on the mathematical possibilities. Deep learning needs the whole dataset for calculations and can not calculate the model rowwise.

```
1:  procedure COMPUTE_SNR(dataset)                          ▷ SNR Function
2:
3:      SUM = [...]
4:      PROD = [...]
5:
6:      for (i=0 to dataset_length - 1) do
7:
8:          buffer = dataset[i]
9:
10:         for (j=0 to trace_length - 1) do
11:
12:             SUM[j] += buffer[j]                         ▷ Sum calculation
13:
14:             PROD[j] += buffer[j] * buffer[j]            ▷ Product calculation
15:
16:     SNR = [...]
17:
18:     for (i=0 to dataset_length - 1) do
19:
20:         N = dataset_length
21:
22:         MEAN = SUM[i] / N                               ▷ Mean calculation
23:
24:         VAR = 1/N ( -(SUM[i])^2 / N + PROD[i] )         ▷ Variance calculation
25:
26:         SNR += VAR(MEAN) / MEAN(VAR)                    ▷ Signal to noise ratio calculation
27:
28:     return(SNR)
```

Figure 3.22 Pseudo Code: SNR Calculation by Row

3.2.4 Dataset 1: Unprotected

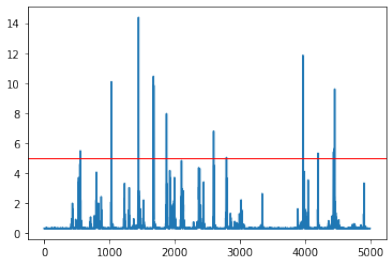


Figure 3.23 T-Test: Real Trace Unprotected

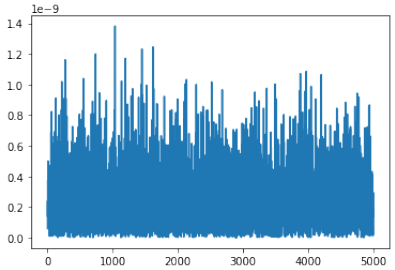


Figure 3.24 Deep Learning: Real Trace Unprotected

For this experiment, 80.000 traces were measured with an unsecured AES encryption. The dataset contains of two different plaintexts. To improve the measurements, averaging was performed. The measured traces have a length of 5.000 features and contain the first round of AES encryption.

T-Test

Figure 3.23 shows the T-test which was performed with the real traces. It can be seen that there are several leakages in the traces. The 5 percent quantile was exceeded several times.

Deep Learning

In sensitivity analysis with deep learning it is difficult to recognize whether there are leaks in the data. In comparison to the T-test, the results are only approximately correct and do not show any clear leaks.

3.2.5 Dataset 2: Protected with Random Delay and 1 Mask

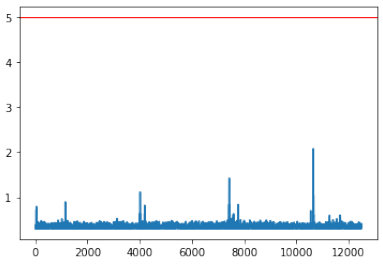


Figure 3.25 T-Test: Real Trace Protected

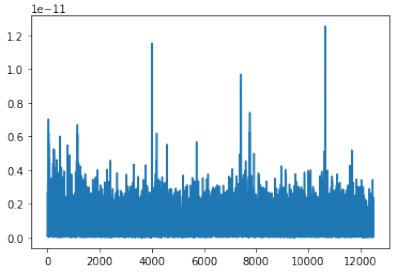


Figure 3.26 Deep Learning: Real Trace Protected

In this experiment 10.000.000 protected traces have been created. Next averaging was performed to reduce the number of traces in the dataset to 1.000.000 entries. Masking and random delay protect the traces. The traces contain 12.000 features and represent the first round of AES encryption.

T-Test

In this experiment the T-Test was carried out with secured real traces. As expected, the T-Test shows no leakages. Masked data are normally protected against a simple T-Test. In some places slight changes can be seen, but none of the values exceeds the 5 percent quantile.

Deep Learning

This experiment investigates the real traces with deep learning. The result is similar to the results of the T-Test. The sensitivity shows three increases between the points 4.000 and 11.000. These increases correspond with the small increases in the T-Test. The increases in deep learning are significantly below 5 percent quantile and is irrelevant but could contain further information.

3.2.6 Summary: Real Traces

Analyzing the experiments with the real traces, it is evident right from the beginning that data preparation and the implementation of encryption was a substantial part of the installation of the experiments. Only with adaptations like averaging usable traces can be measured. To ensure the quality of the datasets SNR and Correlations Attacks are required.

The DL-LA did not work with the unsecured traces, but the secured traces showed first signs of leakages. This was an interesting result. A large number of traces was necessary to achieve the corresponding results which do not show any clear leakages in the data. Optimizing the neural network or bigger datasets could improve the results.

Chapter 4

Discussion

The outcome of the two experiments show several interesting results not apparent at first:

- The T-Test performs as expected.
- Deep learning never clearly indicates any leakages.
- Deep learning behaves differently depending on the countermeasures.
- In deep learning countermeasures with noise can detect better then random delay.
- In both experiments deep learning performs poorer in the experiments without countermeasures.
- In deep learning random data are incorrectly shown as leakages.
- Deep learning seem to be not reliable in regard to the current setup of the neuronal network.

Therefore it cannot be excluded that deep learning includes other factors besides leakages and is unknown so far. This also concerns the optimization of parameters and the amount and quality of data.

In both experiments the traces were tested for their quality and the information gain. Many factors influencing the results have to be considered. Even a device of identical construction could have minimal deviations in the traces so the optimizations have to be adjusted again. Since in the production of CPUs identical results are not possible minimal differences occur but they do not affect the quality or performance. It may influence the position of the leak in the CPU and the quality of the traces. In IT-Security results have to be: valid, unambiguous, transparent, comprehensible.

Than it is still unclear whether the parameter optimization can improve results. Correct parameters must be found before DL-LS can be performed. However, this is not the case

with DL-DA as it is with the T-test. The T-test always delivers a clear result. The sensitivity in deep learning can show leaks in many areas even though there are none. Furthermore, the results are not always reproducible and depend strongly on setup and the training of the neural network. The training the neuronal network during the experiment for several times with the same dataset the result of the sensitivity varies and do not exactly match the previous once. This could quickly lead to misinterpretations and cannot be validated.

An interesting finding was made towards the end of the experiments: Real traces without protection perform worse in DL-DA than the data set with masking and random delay. It could be assumed that the neural network needs more data or needs to be optimized.

During the implementation the sensitivity calculations were not described in detail. It was possible only through personal contact with the scientists to find out about their way of implementation to then implement the calculation itself.

Chapter 5

Future Work

The research work in regard to the data preparation and implementation of the algorithm proved to be complex and was met by a substantial time effort. Then necessary data had to be examined with techniques such as correlations attacks and SNR. Another challenge was a large amount of data generated during the measurements. At times, more than 600 gigabytes per data set were generated. Therefore to perform calculations like SNR analysis, conventional functions can no longer be used to store all data in memory at once. Functions were adapted to calculate huge datasets line by line. This was not evident at the beginning of the project.

A result of the analysis of the simulated traces is the lack of suitability for DL-DA with the previous parameters of the neural network. Even without countermeasures results could be obtained. The analysis with real traces was only slightly better. However, the sensitivity does not show an evident characteristic of leaks, even with these traces.

In the process of the research, various aspects have been identified to be pursued in the future:

- Using networks like CNN instead of MLP.
- Analyzing the influence of CPU modes like GCM.
- Analyzing different kinds of CPUs with DL-LA.
- Parameter optimization of the neuronal network.
- Adding noise to the traces and analyse if noise can improve sensitivity [12].

In summary the T-Test is still considered to be very reliable while deep learning cannot detect leakages with certainty and can even lead to misinterpretation. In IT-security, a technique should always lead to the same result and be traceable, but DL-DA leads to different results of the sensitivity after repeated execution. Also, much depends on the quantity and quality of the data. If correct parameters are not to be obtained the prediction of existing leakages is not possible. But further research can open new outlooks on the future development of deep learning preventing leakages.

Chapter 6

Bibliography

[1] Arduino.cc. *Arduino Due | Arduino Official Store*. visited on 29.12.2019. URL: https://store.arduino.cc/arduino-due.

[2] Naseema Bhanu and N V Chaitanya. "Aes Modes of Operation". In: *International Journal of Innovative Technology and Exploring Engineering (IJITEE)* 9 (2019), pp. 2278–3075. DOI: 10.35940/ijitee.I8127.078919.

[3] Joan Daemen and Vincent Rijmen. "AES Proposal: Rijndael". In: (Jan. 2003).

[4] Joan Daemen and Vincent Rijmen. "The Block Cipher Rijndael". In: vol. 1820. Jan. 1998, pp. 277–284. DOI: 10.1007/10721064_26.

[5] Divyansh Dwivedi. *Machine Learning For Beginners - Towards Data Science*. visited on 26.02.2020. 2018. URL: https://towardsdatascience.com/machine-learning-for-beginners-d247a9420dab.

[6] Morris Dworkin. *Recommendation for Block Cipher Modes of Operation: Galois/-Counter Mode (GCM) and GMAC*. Tech. rep. 2007. DOI: 10.6028/NIST.SP.800-38d.

[7] ETutorials.org. *Steps in the AES Encryption Process :: Appendix A. Overview of the AES Block Cipher :: Appendixes :: 802.11 security. wi-fi protected access and 802.11i :: Networking :: eTutorials.org*. visited on 29.12.2019. URL: http://etutorials.org/Networking/802.11+security.+wi-fi+protected+access+and+802.11i/Appendixes/Appendix+A.+Overview+of+the+AES+Block+Cipher/Steps+in+the+AES+Encryption+Process/.

[8] Google. *Host-based card emulation overview | Android-Entwickler*. visited on 30.01.2020. URL: https://developer.android.com/guide/topics/connectivity/nfc/hce.html.

[9] Razi Hosseinkhani. "Using Cipher Key to Generate Dynamic S-Box in AES Cipher System". In: *International Journal of Computer Science and Security (IJCSS)* 6 (2012), pp. 19–28.

[10] M Jordan, J Kleinberg, and B Scho. *Pattern Recognition and Machine Learning*. 2006. ISBN: 9780387310732.

[11] Nahua Kang. *Multi-Layer Neural Networks with Sigmoid Function - Deep Learning for Rookies (2)*. visited on 25.12.2019. URL: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f.

[12] JooHyung Kim. *What is AES? — Step by Step - zeroFruit - Medium*. visited on 29.12.2019. URL: https://medium.com/@14wnrkim/what-is-aes-step-by-step-fcb2ba41bb20.

[13] Arm Limited. *Microprocessor Cores and Technology – Arm*. visited on 11.12.2019. URL: https://www.arm.com/products/silicon-ip-cpu.

[14] Vikashraj Luhaniwal. *Analyzing different types of activation functions in neural networks-which one to prefer?* visited on 25.12.2019. URL: https://towardsdatascience.com/analyzing-different-types-of-activation-functions-in-neural-networks-which-one-to-prefer-e11649256209.

[15] Hamza Mahmood. *Hamza Mahmood - Towards Data Science*. visited on 25.12.2019. URL: https://towardsdatascience.com/@hamzamahmood.

[16] Stefan Mangard. "Hardware Countermeasures against DPA – A Statistical Analysis of Their Effectiveness". In: vol. 2964. Feb. 2004, pp. 222–235. DOI: 10.1007/978-3-540-24660-2_18.

[17] George Marsaglia. "Xorshift RNGs". In: *Journal of Statistical Software* 8 (2003), pp. 1–6. ISSN: 15487660. DOI: 10.18637/jss.v008.i14.

[18] Amir Moradi et al. *Leakage Detection with the x2-Test*. 2018. URL: https://tches.iacr.org/index.php/TCHES/article/view/838/790.

[19] NVIDIA-Corporation. *Deep Learning | NVIDIA Developer*. visited on 30.01.2020. URL: https://developer.nvidia.com/deep-learning.

[20] Santos Pozo et al. “Side-Channel Attacks from Static Power: When Should We Care?” In: 2015 (Apr. 2015), pp. 145–150. DOI: 10.7873/DATE.2015.0712.

[21] Ludovic Rembert. *What is AES Encryption? A Beginner Friendly Guide - Privacy Canada*. visited on 10.03.2020. 2020. URL: https://privacycanada.net/aes-encryption-guide/.

[22] Margaret Rouse. *What is Advanced Encryption Standard (AES)? - Definition from WhatIs.com*. visited on 11.12.2019. URL: https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard.

[23] Tobias Schneider and Amir Moradi. “Leakage Assessment Methodology-a clear roadmap for side-channel evaluations”. In: (2015).

[24] Tobias Schneider and Amir Moradi. *Leakage Assessment Methodology-a clear roadmap for side-channel evaluations*. Tech. rep. 2015.

[25] Benjamin Timon. “Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis”. In: 2019, Issu.2 (2019), pp. 107–131. DOI: 10.13154/tches.v2019.i2.107-131.

[26] Felix Wegener, Thorben Moos, and Amir Moradi. *DL-LA: Deep Learning Leakage Assessment A modern roadmap for SCA evaluations*. 2019.

Appendix A

General Additions

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

Table A.1: Rijndaels S-Box for encryption

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

Table A.2: Inverse Rijndaels S-Box for decryption