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

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

Assessment of Smart Card Leakage Detection Approaches Based on Deep Learning

vorgelegt von Johannes T. Brandau

am 15. Juli 2020

Referent: Prof. Dr. Josef Fink
Korreferent: Dr. Shivam Bhasin

Erklärung

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

Wiesbaden, 15.07.2020 Johannes T. Brandau

Contents

List of Figures

List of Tables

List of Source Codes

Chapter 1

Introduction

Since contactless payment with smartcards using devices such as smartphones or smartwatches via near field communication (NFC) is becoming more and more popular, the corresponding means of payment are also increasing [38, 53, 54]. The security of the payment process is an essential precondition to ensure user acceptance. The detection of security gaps (leakage detection) is of considerable importance, e.g. in areas of the economy, data and consumer protection. A high level of security also ensures better acceptance by users when making payments.

Identified security gaps (leakages) can be used, e.g. to uncover the key of a secured payment process and effect unauthorized payments. One way of detecting security gaps is focused on so-called side-channel attacks. There are different kinds of side-channel attacks in identifying potential security gaps, e.g. based on the voltage of a CPU. Moradi, Schneider and Moos [58] describe how deep learning can be applied to find security gaps in a dedicated hardware environment, even if countermeasures on the payment device should prevent this. Moradi, Schneider and Moos claim that the leakage detection with deep learning presented by them is capable of detecting leaks without any preparation or prior knowledge of the underlying implementation [58]:

> „We do claim that the chosen network is able to learn first-order, higher-order, univariate, multivariate and horizontal leakages without requiring any pre-processing or prior knowledge of the underlying implementation.“

However, Moradi, Schneider and Moos only examine data from one CPU (Spartan-6 on a SAKURA-G Board) and calibrate the neural network to this CPU. It is therefore still unclear whether leakage detection with deep learning can also be applied to other CPUs, e.g. CPUs which are used for virtual smart cards [58].

An established procedure for the detection of potential security gaps (leakages) is the T-Test, which is widely used and reliable in practice [15, 51, 52]. The T-Test is often used as a benchmark for newer methods, e.g. those based on deep learning. For a method to be suitable for leakage detection, it must be examined whether it is fast, reliable and robust [51]. In practice these are the main requirements for leakage detection. The detection of leakages should be accomplished in an adequate period of time. A leakage detection has to lead into the same results, not unless the given technique is reliable. Since each technique has its strengths and weaknesses, further research improves the robustness of weak points [52].

1.1 Theoretical Fundamentals

The T-Test is considered a reliable method for detecting leaks in voltage time series, also known as traces [11]. During the last years, machine learning and especially deep learning based methods became more and more popular in side-channel attack prevention [6, 22, 28, 30, 50]. E.g. with the help of deep learning, there is considerable evidence to reproduce the key based on traces [20, 27, 33, 43, 56, 57, 59]. Supervised techniques like support vector machines, random forest or approaches with neural networks were successful in recovering the key from protected or unprotected cryptographic algorithm implementations [57].

Deep Learning Leakage Detection differs from the T-Test in such a manner that all data points are connected in a neural network. The neural network is trained to assign encryption keys to sequences of traces. This allows previously unknown information and anomalies to be disclosed. The advantageous approach from Moradi, Schneider and Moos combines as many points of the neural network for the classification of the groups as necessary, even in complex scenarios with multivariate or horizontal leakages [58].

After a neural network is trained with the help of the deep learning based Sensitivity Analysis it is possible to make leakages visible in traces. The Sensitivity Analysis should locate all points of interest by quantifying how much they have contributed to the leakage function learned from the neural network [58]. The neural network has so far only been trained on the voltage traces of a specific CPU of the SAKURA-G board. Moradi, Schneider and Moos leave the application and evaluation of their approach to other CPUs for future work on this topic [58].They have not investigated how their Deep Learning Leakage Detection method can be applied to other CPUs. Also, it is unclear to what extent countermeasures such as random delay, noise or masking influence the Sensitivity Analysis, and if the Sensitivity Analysis can be used to identify the different types of countermeasures. With masking, the information is hidden at two data points in a trace. Masking is known as a countermeasure to protect cipher implementations like the Advanced Encryption Standard (AES) against side-channel attacks. During the masking every sensitive intermediate variable will be randomly split into shares [45].

The identification of countermeasures enables attackers to use more effective attacking techniques. It is known from other areas of side-channel attack prevention in combination with deep learning that the identification of countermeasures is possible [3, 16, 19, 24, 29, 43, 56, 57].

In real-world settings, measuring and analyzing traces can be very time-consuming, depending on the CPU architecture and countermeasures. In practice, it is therefore common to first generate synthetic traces. This can differ from real traces in the detection accuracy of leakages. Simulated traces can be generated quickly and cost-effectively. They can also be used to test the correctness of procedures or implementations by simulating specific leaks and countermeasures. The generated leaks must then be found by a statistical hypothesis test known as T-Test. This guarantees the correctness of the implementation and of the T-Test. The architecture of a CPU plays an essential role in finding leaks. Moreover, too much noise in a signal can lead to few information being contained in the data being too low and therefore leaks can not be found. To increase the quality and prevent false measurement results, there are methods such as the signal-to-noise ratio (SNR) or noise reduction. SNR is a technique that analyzes the information content of traces based on the signal and noise [24, 52, 60]. It has been shown over the last few years that the number of measured traces is decisive for measurement accuracy [4]. An increasing number of traces increases the possibility of finding leaks and reduces misinterpretations [52, 56, 58].

1.2 Research Questions and Current State

In their work [58], Moradi, Schneider and Moos are concerned with leakage detection based on deep learning. The investigation of leakage detection is of further use: It can be investigated if the method presented by Moradi, Schneider and Moos is applicable to another CPU, it could also be investigated whether the Sensitivity Analysis provides indications on certain countermeasures [2, 35, 58, 61]. The added value of finding anomalies in the voltage trace and knowing what type of vulnerability is involved would make it easier for CPU architects to improve security [46]. Possible vulnerabilities or errors are masked afterwards [58]. The current state of research leads to the following questions:

- Can anomalies (false positives) be detected in synthetic voltage traces even if no encryption was used (baseline measurement of traces with just noise or random generated data)?
- Is it possible to detect anomalies in simulated traces using the T-Test and deep learning (Sensitivity Analysis)? And if so, with what accuracy (true positives)?
- What is the difference in prediction performance between the two methods (T-Test and Sensitivity Analysis) for detecting anomalies in the voltage in a synthetic environment? Are there differences in detection performance?
- To what extent do countermeasures such as noise, random delay, and masking influence the detection performance of the above techniques in a synthetic environment?
- Is it possible to apply the neural network training parameters, which have been especially adapted for one CPU of the SAKURA-G board, to another CPU, which is used for virtual smart cards?
- Do countermeasures influence Sensitivity Analysis? If so, can countermeasures be detected?
- Can the results (accuracy of the T-Test and the Sensitivity Analysis) be compared with those from the paper by Moradi, Schneider and Moos?

1.3 Methodology

The research is divided into three sections. In order to investigate the method mentioned by Moradi, Schneider and Moos, a T-Test is applied in each experiment, which is considered a baseline. Then the Sensitivity Analysis is used and compared with the T-Test:

a) In the first experiment different traces without any security holes have been created which should provide a baseline test. In the beginning, an experiment with two necessary tests is carried out. These tests should help to reveal the extent to which the two tests reveal anomalies. Especially in these cases without any security holes at all, detections would be identified as false positives. First of all, it will investigate how Sensitivity Analysis behaves with traces that contain no leaks. It then examines how Sensitivity Analysis behaves with traces if Simulating peaks in voltage that stem from computationally intensive tasks other than encryption at certain points in the trace. This test is intended to show how Sensitivity Analysis behaves in CPU-intensive phases, comparable to that of encryption. The difference to encryption is that it does not deal with leaks, but with computationally intensive tasks.

b) In the second experiment, different countermeasures have been applied to the generated traces in order to prevent the disclosure of potentially sensitive information to attackers. Eight data sets with traces (virtual traces) are generated. The traces are protected by countermeasures of different strength, such as noise, random delay and masking. This experiment is intended to investigate the extent to which differences in sensitivity are apparent in the various countermeasures. It will also check whether leaks can be found in the artificially created traces.

c) In the third experiment, real data (real traces) of a CPU are examined, which is used in smartwatches with virtual smart cards. In the experiments with real traces, a sensor for measuring the current voltage is positioned directly on the CPU. Then the voltage (traces) is measured over time, before, during and after data will be encrypted. This results in further experiments. One experiment contains traces, with unsecured countermeasures. The other set of data contains traces that have been protected with masking. Subsequently, as with the virtual traces, the data sets are first examined with a T-Test as a baseline and then compared with the results of a Sensitivity Analysis.

For the test with the real traces, a sensor for measuring the current-voltage is positioned on the CPU, followed by the immediate measurement of the voltage during the encryption of data. This results in two data sets to investigate how Sensitivity Analysis performs under real-world conditions, with and without countermeasures.

The aforementioned experiments can be motivated by the cells of a confusion matrix. The performance of the procedures for the detection of potential security gaps is split down into false positive, true negative, false negative and true positive. The idea is to holistically assess the performance of T-Test and Sensitivity Analysis along the cells of a confusion matrix.

Chapter 2

Background

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

2.1 Virtual Smart Cards

Over the last few years, the use of smart cards has increased considerably. A smart card includes a chip which authenticates a user. The possibilities are diverse, be it at doors or 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. When contactless payment was introduced, a card has only to be held at the device. With this new method of payment, the virtual smart card has become increasingly important.

Mobile devices with the technical requirements can be used for contactless payments. The smartphone or smartwatch 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.

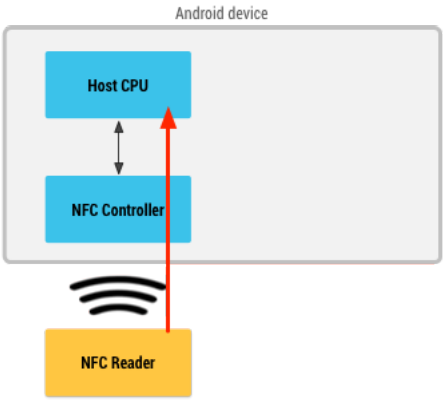

Figure 2.1 Host-based card emulation [17]

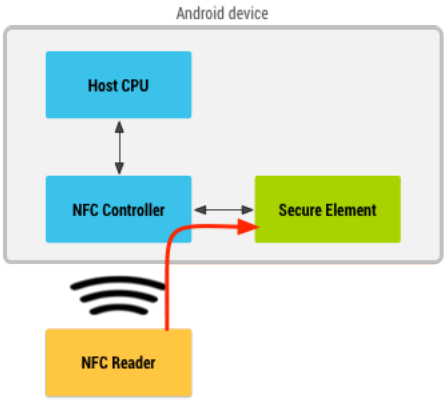

Figure 2.2 Card Emulation with secure element [17]

There are two different techniques to implement a virtual smart card in a smartphone. One is the Host-based card emulation (HCE) figure 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 figure 2.2. First the data will be encrypted in a secure element and then forwarded to the CPU for processing [17].

In the past HCE was a common technique to emulate smart cards. Furthermore, in android, the secure element has to be implement correctly. If mistakes occur in the implementation or in the usage of older hardware the encryption is calculated in the CPU. Older smartphones have no secure element. In this case HCE is the only possible solution. The research work at chapter 3 is based on a host-based architecture. The encryption is done inside the CPU, not in the secure element.

2.2 Artificial Intelligence

Over the last few years, Artificial Intelligence (AI) has become increasingly important. However, AI is not quite new as it was already used in the 90s in a chess game competition. But it was technical progress, cheap hardware and clouds, which made possible the growing use of AI. AI along with other techniques include: Machine learning and deep learning, whereas machine learning has been used 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 [41]. The first section 2.2.1 describe the classical machine learning techniques, the second section 2.2.2 illustrates deep learning 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 requires the construction of algorithms that adapt their models to improve their ability to make predictions, based on statistical weights. The classical machine learning is not a new discipline and can be split into three different classes [12]:

Supervised

Supervised Learning is one of the most popular techniques. 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 compared with the original label of the test data to calculate the accuracy [12].

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, a label does not represent classes but divides measurable units like quantity, time or size of 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 [12].

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. 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. Algorithms are needed to remove unexpressed robust data and features and to reduce the dimensions of the data [12].

Reinforcement

Reinforcement Learning is another type of machine learning. It is not based on raw data like in supervised and unsupervised learning. The algorithm has to figure out the situation on its own. 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 [12].

2.2.2 Deep Learning

Deep Learning (DL) is a subset of machine learning used successfully in many fields like image classification, speech recognition, or genomics. The idea of DL is based on Artificial Neural Networks (ANN), a group of interconnected nodes 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 are linked to each other. Links represent the weight between the nodes [21, 56]. 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 are 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 neural network.

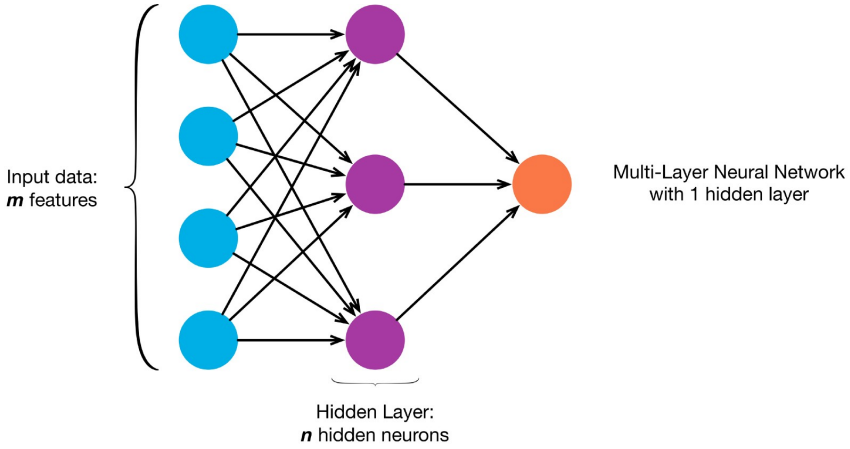

Figure 2.3 Multi-Layer Perceptron [23]

A Multi-Layer Perceptron (MLP) (figure 2.3) consists of a group of Single-Layer Perceptron (SLP) (figure 2.5). 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, 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. Then the weights are summed up 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. For the research in this work a ReLU-Function (Rectified Linear Unit) was used as activation function. ReLU is calculated by the following formula [32]:

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

Figure 2.4 shows the most common activation functions for neural networks:

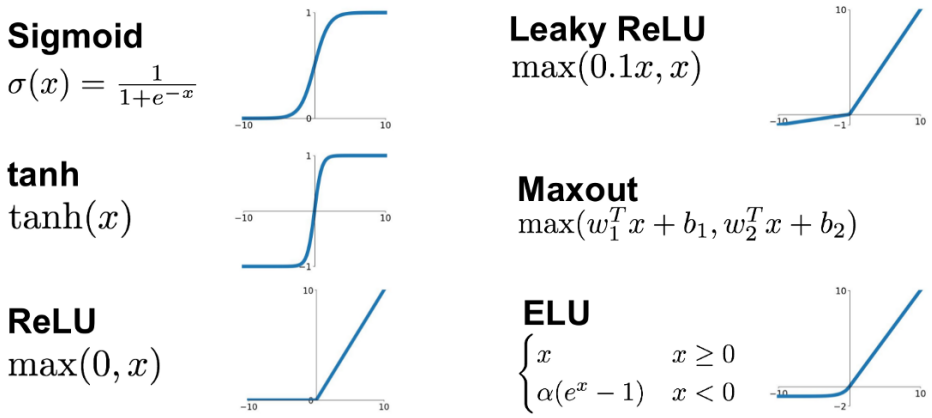

Figure 2.4 Activation functions for neural networks [42]

The following figure 2.5 represents the procedure of a single perceptron. Starting with the Input (1), the Weights (2), Summ and bias (3), the Activation Function (4) and the Output (5).

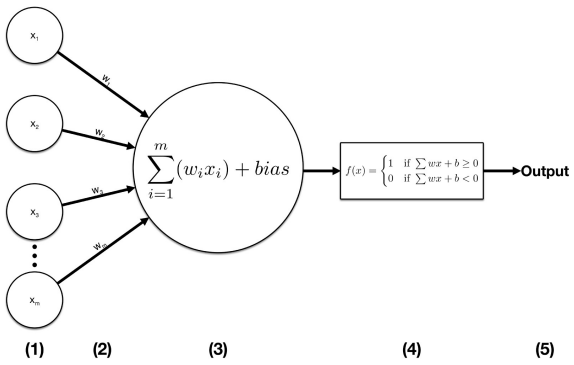

Figure 2.5 A Single Perceptron [23]

Convolutional Neural Network (CNN) is a part if the neural 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. The kernel, also called filters, convoluted the original shape into a different perspective to classify the data. Figure 2.7 describes the procedure between CL (blue) an Kernel (yellow). The Result is a smaller output matrix (green) which represents the CL. It is possible to use multiple layers with different kernels, sizes or CLs.

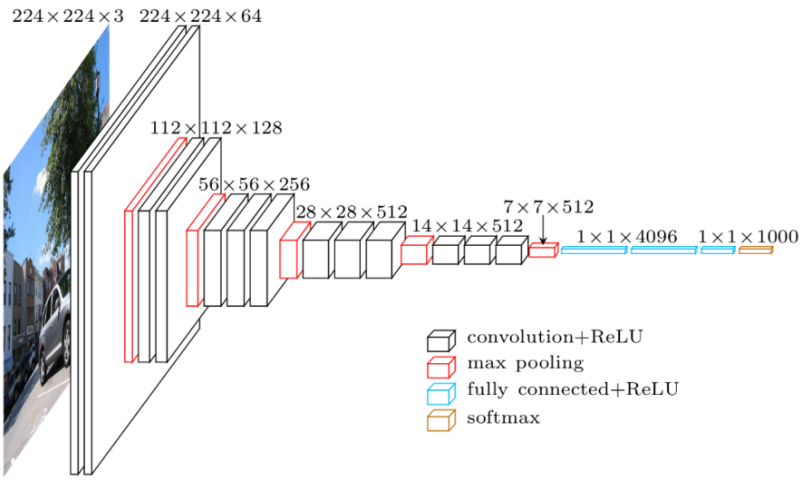

Figure 2.6: Convolutional Neural Network

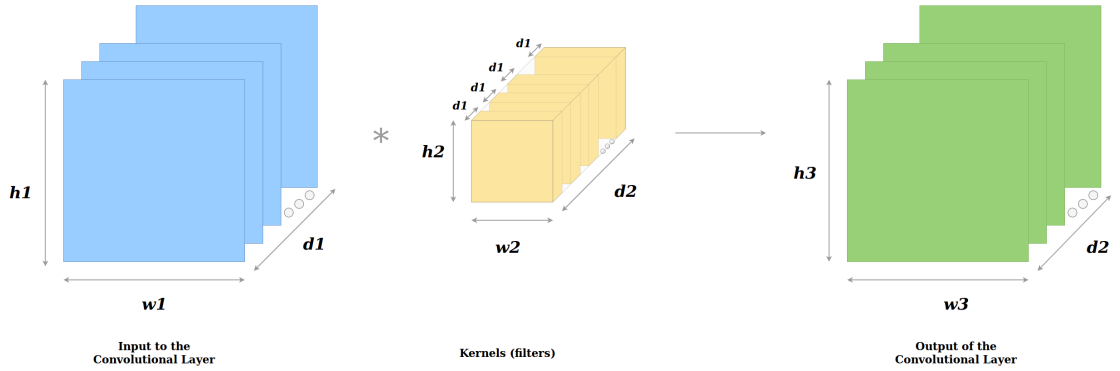

Figure 2.7 Convolutional Layer and Kernel

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 [34]:

$$P(y{=}j|\Theta^i){=}\frac{e^{\Theta^{(i)}}}{\sum_{j=0}^{k} e^{\Theta_k^{(i)}}}$$
$$where\ \Theta{=}W_0X_0{+}W_1X_1{+}\cdots{+}W_kX_k{=}\sum_{i=0,}^{k} W_iX_i{=}W^TX$$

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. Figure 2.8 represents the process of max pooling, a matrix with the size of 4x4 will be transfert to a 2x2 matrix based on max pooling [10].

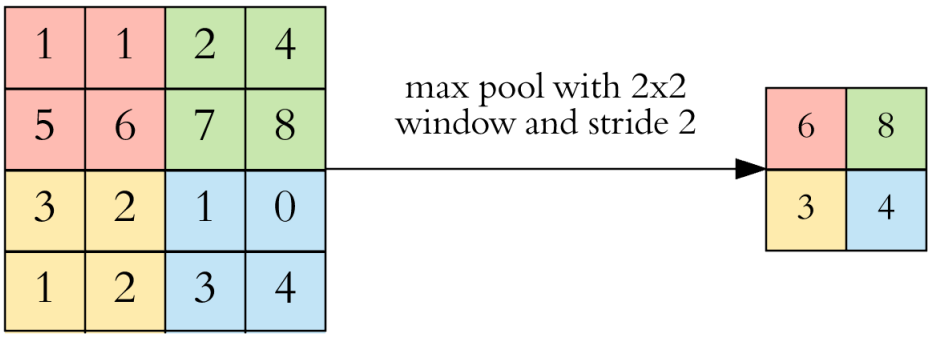

Figure 2.8 Max pooling [10]

2.2.3 Confusion Matrix

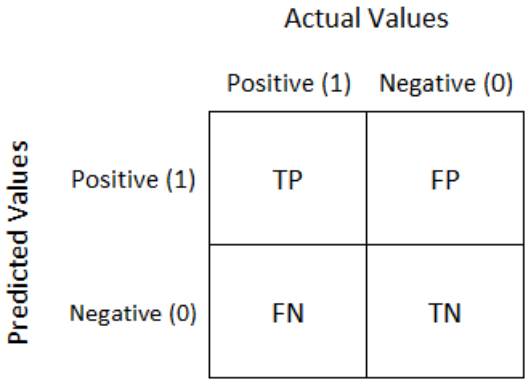

Figure 2.9 Example of a Confusion Matrix [49]

A Confusion Matrix (CM) is used in machine learning especially in classification to measure and visualize the performance of an algorithm. The output of the classification can be two or more classes. Figure 2.9 represent a confusion matrix with two values. The confusion matrix has four terms: True Positive (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) [49]. A confusion matrix has two dimensions. The first dimension represents the predicted values, the second the active values. In the case of the True Positive value, the active and the predicted value has to be true, the predicted and the active value are both correct. A True Negative happens if the active value and the predicted value is negative, so both the active and the predicted value is wrong. If the predicted value is positive, and the active value is negative, it is called False Positive. Otherwise, if a predicted value is negative and the active value is positive, it is called False Negative. With the help of the confusion matrix, it is possible to estimate the Recall, Precision and Accuracy [7, 49]:

$$Recall = \frac{TP}{TP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

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 the source code into the micro controller. In the third 2.3.3 the environment and the setup for the measurement will be explained.

2.3.1 Microcontroller Board: Arduino Due

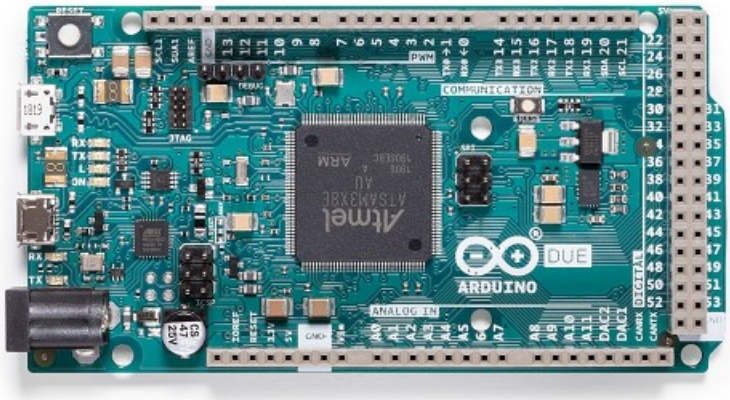

Figure 2.10 Arduino Due Board [1]

The Arduino Due board (figure 2.10) 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 is used in IoT (Internet of Things) to contactless payments and smart cards [31]. The Board provides pins, a larger size of a CPU which make measurements and the implementation of AES more convenient. The Arduino Device provides 32 pins. It is possible to access the pins in a C-Code implementation.

2.3.2 Microcontroller Implementation

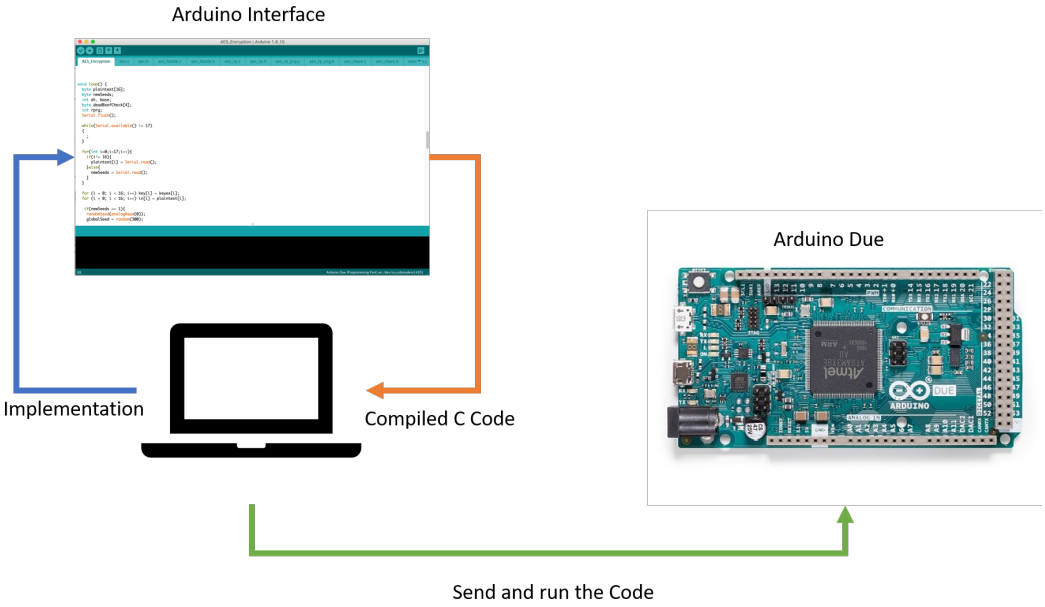

Figure 2.11 Process of Code Implementation into the Micro Controller

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 countermeasures. 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 2.11 shows the mentored process of the integration 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 parameters 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

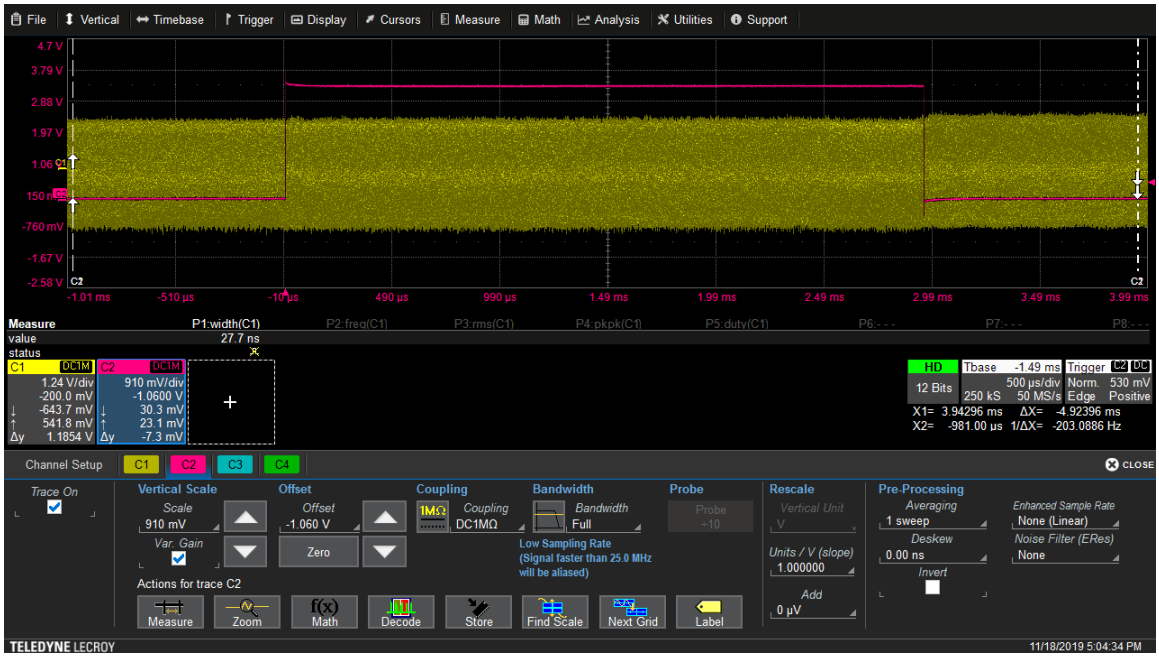

Figure 2.12 Oscilloscope GUI: Measurement of Traces (yellow) and Trigger (red)

The oscilloscope can be used to measure the power/electromagnetic consumption (traces) of a device. To measure traces of a CPU a sensor has to be placed on the CPU. During the experiments in chapter 3.3.2 a grid search is used to find the position for the measurements on the CPU. During a grid search at multiple positions of a CPU traces will be measured to find the correct position where the CPU is calculating the encryption. Without knowing the architecture of a CPU, techniques like a grid search can help to find the correct position. The sensor measures the whole time the power consumption of the CPU even when nothing will be calculated with CPU. It is not possible during the measurements to find the start and the end of an encryption in a trace. A trigger visualizes the exact area of the encryption. Figure 2.12 shows the graphical interface of the oscilloscope with the traces

and the trigger. The yellow line represents a trace. The red line represents the trigger. 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 2.13 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.

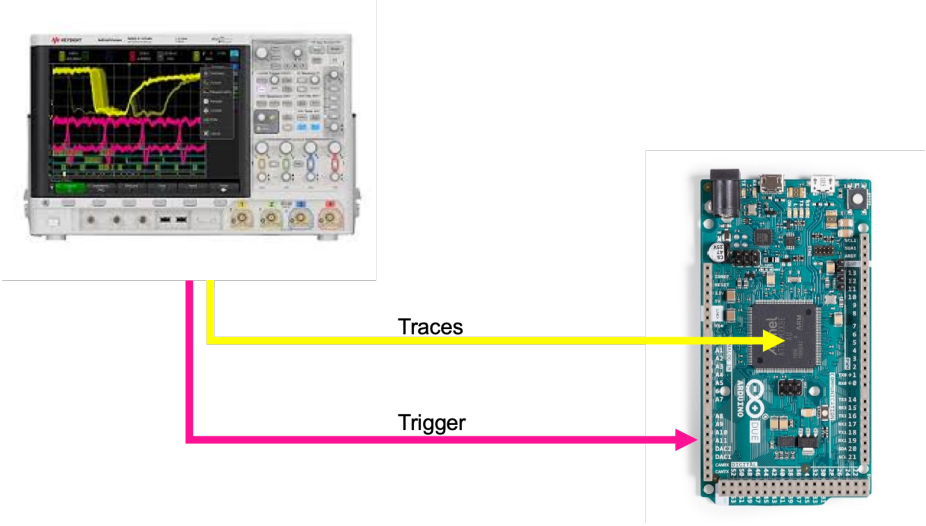

Figure 2.13 Oscilloscope Connection to the CPU and the Pins

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 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. 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 high 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.

```
//Initialize the pin for the trigger
int LEDpin = 33;
pinMode(LEDpin, OUTPUT);

//Start high signal at the trigger
digitalWrite(LEDpin, HIGH);

//Run encryption while trigger is on hight signal
run_encryption(...);

//End high signal at the trigger
digitalWrite(LEDpin, LOW);
```

Source Code 2.1: Control of the Trigger During the Encryption

The implementation of the trigger is 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 set is getting large. The figure 2.14 and 2.15 shows the trace with a low and a regular time sampling rate.

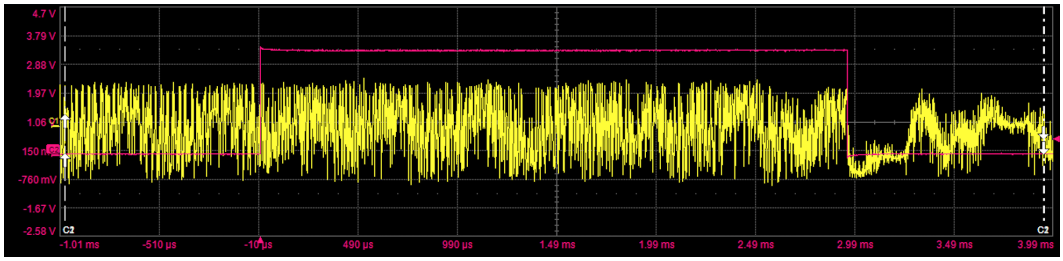

Figure 2.14 Low Time Sampling Rate

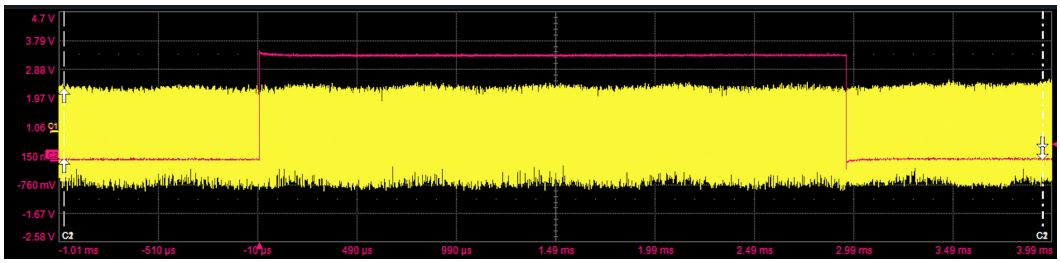

Figure 2.15 Normal Time Sampling Rate

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 into single traces. Figure 2.16 and 2.17 shows the measurement of a sampling rate of 10 and 50 encryptions.

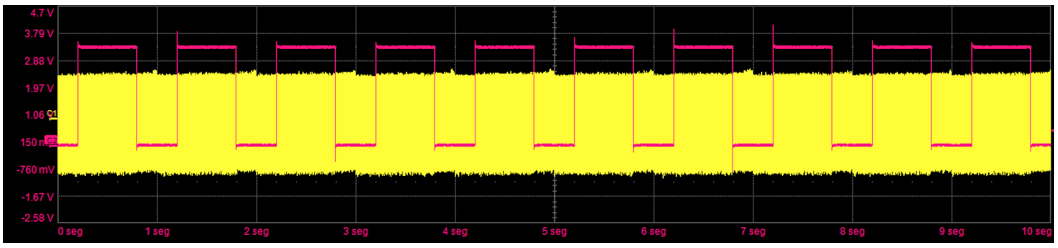

Figure 2.16 Ten Measurements in One Sequence

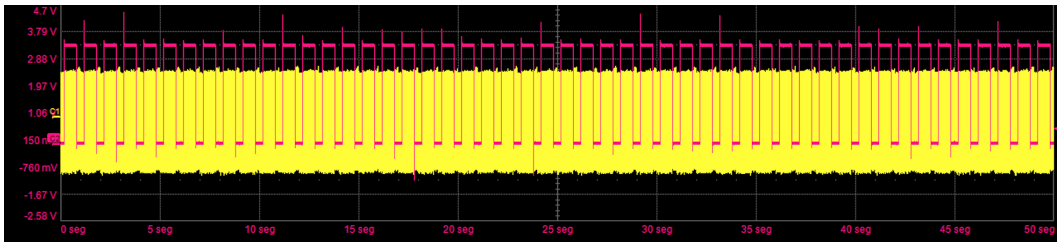

Figure 2.17 Fifty Measurements in One Sequence

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 Data Encryption

In this chapter all basics are described on which the research is based in the field of data encryption. First of all the principle of substitution box and side-channel attacks is outlined. 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.

In section 2.4.1 the encryption technique of Advanced Encryption Standard are 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 are described in detail. Section 2.4.4 describes detailed how side-channel attacks works and how countermeasure in section 2.4.6 can help to prevent this kind of attacks. In section 2.4.7, a technique is applied to find unknown leakages. Section 2.4.8 uses mathematical basics how the leakage detection works based on the current mathematical techniques, section 2.4.9 describes a new technique how leakage detection could work based on deep learning.

2.4.1 Advanced Encryption Standard

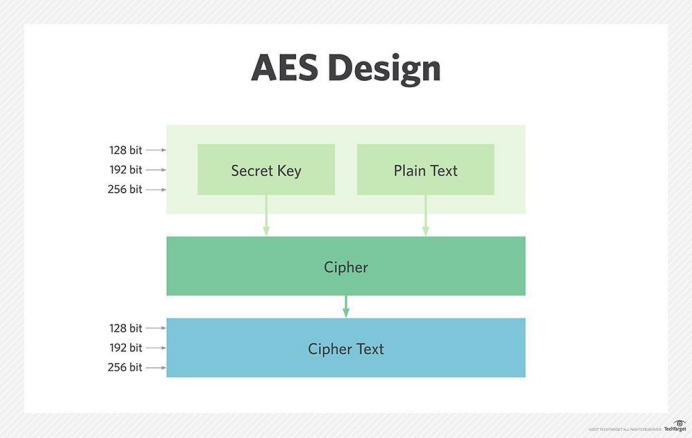

Figure 2.18 Design of the Advanced Encryption Standard [47]

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 2.18 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 is described 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 2.19). 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 [48].

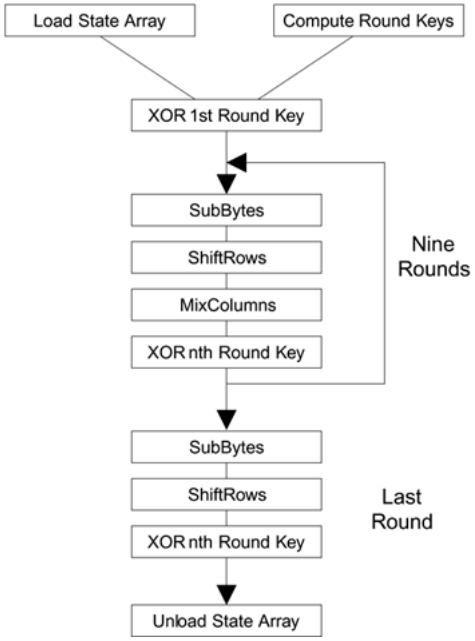

Figure 2.19 AES Rounds in Detail (128 bits) [14]

In detail, the encryption in AES with 128 bits is as follows [18, 26]. Each round in AES goes through the following 4 phases [8]. 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 [8].

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 [8].

3. MixColumns:

In the third stage, all columns are transformed by multiplying it with a specific multiplication polynomial [8].

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 [8].

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 2.20 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 2.20). 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 because the encryption of the S-Box works byte wise, 8 bits represent 1 byte [9].

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 [36]. 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 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: Implementation 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 ones [13]:

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. Since the ciphertext will not be blurred this technique is a disadvantage [5].

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

In the PCBC technique, the plain text block will be XOR with an initialization vector which 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 ciphertext [5].

3. CFB - (Cipher Feedback Mode)

The CFB mode is similar to the PCBC mode, in every round the data from the previous round is encrypted and not the plain text. This technique makes it possible to use just one thread for the encryption, also the decryption of the blocks can be split in multiple threads [5].

4. OFB - (Output Feedback Mode)

In the OFB mode, a keystream of bits is created. It is used for encrypting subsequent blocks. The encryption and decryption can be used only one thread at a time [5].

5. CTR - (Counter Mode)

In CTR a nonce represented as a unique number is 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 [5].

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 [13].

2.4.4 Side-Channel Attacks

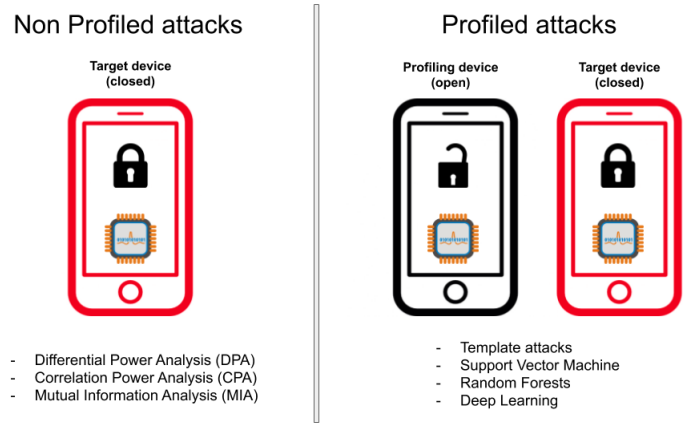

Figure 2.21 Non-Profiled and Profiled attacks [55]

There are many ways how side-channel attacks can be used (figure 2.21). In this research work side-channel attacks are based on electromagnetic measurement traces. Based on this traces side-channel attacks can be used to find leaks or to recover the key of an encryption. To execute a side-channel attack two different techniques are applied, the profiled- and non-profiled side-channel attack.

In a profiled side-channel attack (PA), the attacker has two identical devices and access to one of these two devices witch is similar to the one being attacked. The other one serves as an attacking device. The most common techniques for attacks are template attacks or machine learning. The attack pattern is similar to supervised learning with a training set (Data) and a test set (Target) [55].

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 range 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 [55].

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 Signal to Noise Ratio

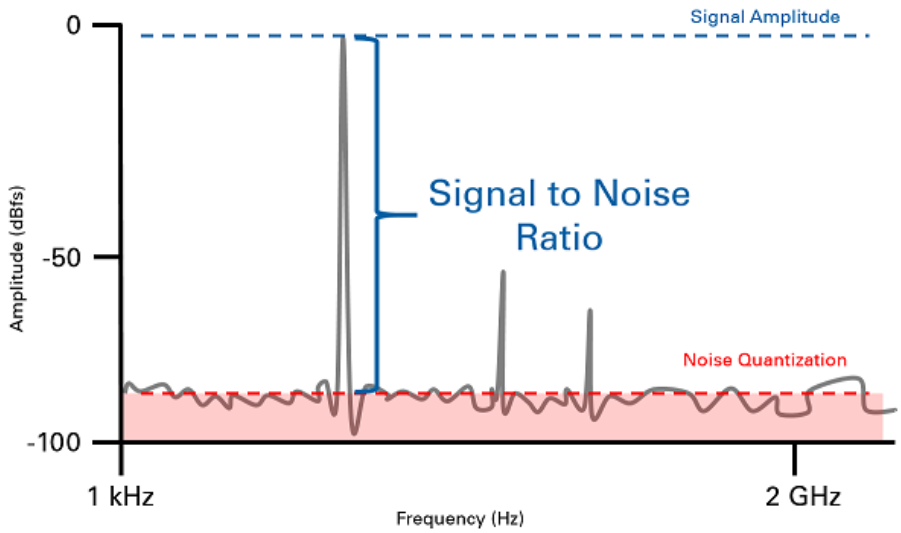

Figure 2.22 Signal to Noise Ratio [40]

An SNR (signal-to-noise) 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. If the signal-to-noise ratio is increasing a lesser amount of traces is needed to engender higher efficiency [60].

The accuracy of SNR depends on the size of the dataset. Consequently as many traces as possible have to be measured to get a significant SNR. The SNR is calculated by the formula [36, 44]:

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

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 2.22 shows a plot of a signal to noise ratio.The red line marks the baseline. If the baseline will be crossed the signal has information value. Data points below the baseline of the SNR are noise and more of interest.

2.4.6 Countermeasures against Side-Channel Attacks

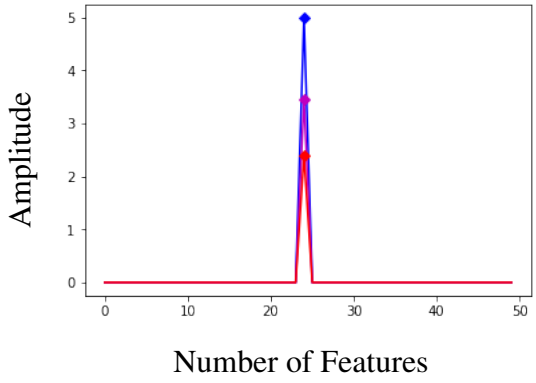

Figure 2.23 Traces with Noise

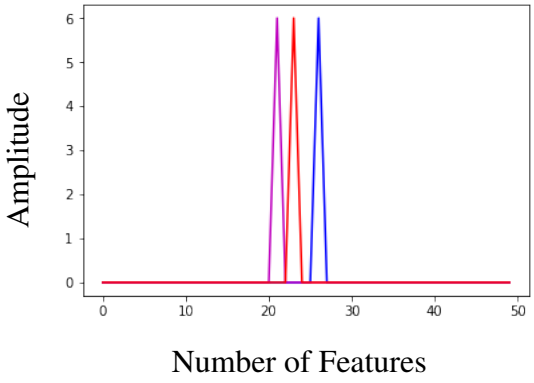

Figure 2.24 Traces with Random Delay

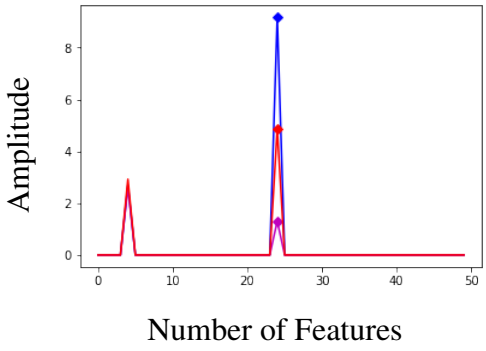

Figure 2.25 Traces with Masking

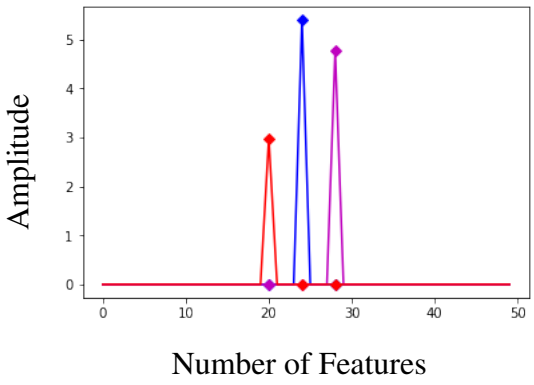

Figure 2.26 Traces with Noise and Random Delay

It is usual practice to protect implementations with countermeasure against side-channel attacks. In the following the most common techniques of countermeasures like noise, random delay and masking were investigated and described. Countermeasures can be applied to disguise the data and the signal and make it harder or impossible to find leakages in traces. In the experiments in chapter 3 traces are protected with noise, random delay and masking. The following explains the countermeasures by settings used in the experiments:

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 2.23 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 2.24 shows how random delay can affect a sample. For the experiments 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 intermediate value including the key. All the operations are performed on the masked value. 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 traces and is considered resistant to the traditional T-Test. If n stands for the number of masks the implementation of n masks is called n-order. Figure 2.25 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, noise and random delay may be combined. Figure 2.26 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.

2.4.7 Leakage Detection

The idea of leakage detection 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). Figure 2.27 shows an example of the structure of a fixed-vs-fixed dataset. The plain text per group will not change but is different from the other group [58]. Security in smart cards has become more important since the advent of contactless payment. In the field of side-channel attacks, procedures are developed to evaluate the security of devices and uncover possible vulnerabilities.

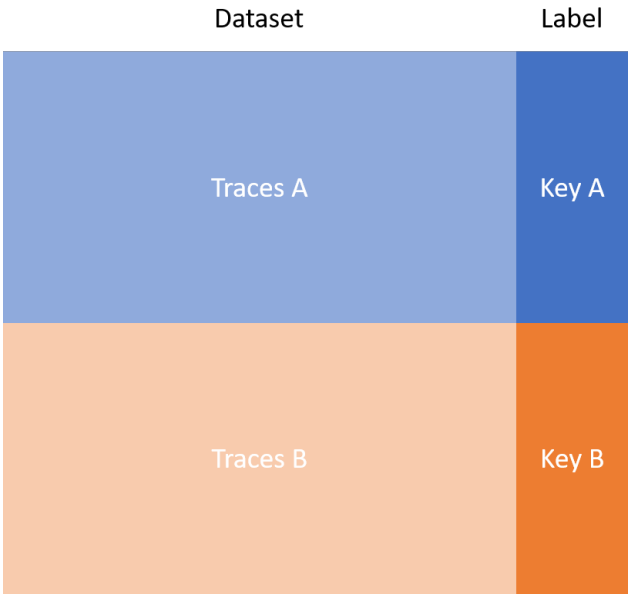

Figure 2.27 Example of a fixed-vs-fixed Dataset

2.4.8 T-Test Leakage Detection

The T-Test is a well known technique to find leakages in data. Especially in side-channel attacks the T-Test is a common technique to identify the dependency between two datasets and possible leakages in traces. Figure 2.28 shows a negative and figure 2.29 a positive T-Test. Every T-Test has a baseline (red marked in the figures). If a feature will reach the baseline or below, the feature is leaking information which can be used to recover the key of the AES encryption. In the negative T-Test no feature is crossing the baseline at the positive T-Test, the feature at position 24 crosses significantly the baseline. At this position information is leaking.

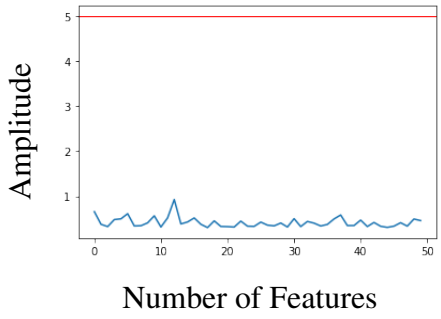

Figure 2.28 False T-Test

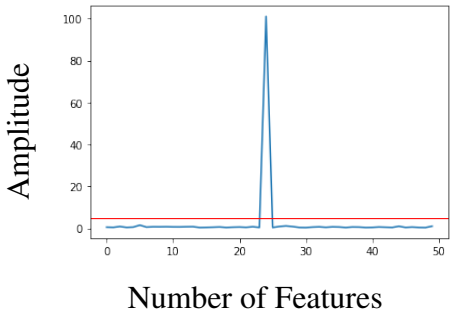

Figure 2.29 Positive T-Test

To compute the T-Test two datasets Q_1 and Q_2 are denoted 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 [58]:

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

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 [58, 51, 39] .

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

It is usual practice to set the threshold to $|t|$> 5 for simplicity. If the threshold is $|t|$ > 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 [39] [58].

2.4.9 Deep Learning Leakage Detection

Wegener et al. [58] describe a technique based on Sensitivity Analysis (SA) to detect leakages with the help of deep learning. The Deep Learning Leakage Detection (DL-LD) is used instead of a classical T-Test. The intent of the new deep learning technique is to compute the sensitivity by a Sensitivity Analysis (SA). The sensitivity is a value based of the gradients for every feature similar to calculation of the T-Test. The sensitivity can provide the information if a feature in a trace is leaking information. A high sensitivity for a feature means leaking of information and a possible security gap to recover the key of the AES encryption. The DL-LD technique with Sensitivity Analysis 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 with features

Figure 2.30 shows a T-Test compared to the Sensitivity Analysis oft the Deep Learning Leakage Detection in figure 2.31. The results of Moradi, Wegner and Moos [58] show the Sensitivity Analysis with leakages in the traces. The neural network is trained with traces from a CPU of a SAKURA-G Board.

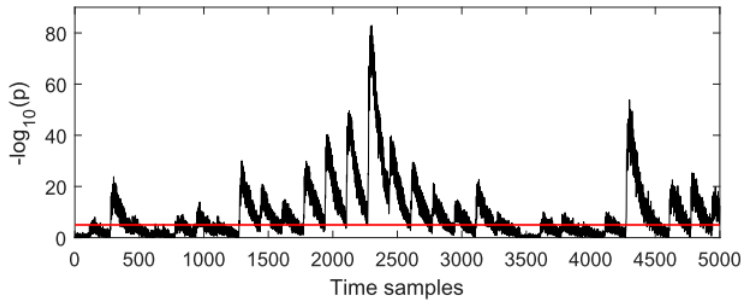

Figure 2.30 T-Test by Moradi, Wegner and Moos [58]

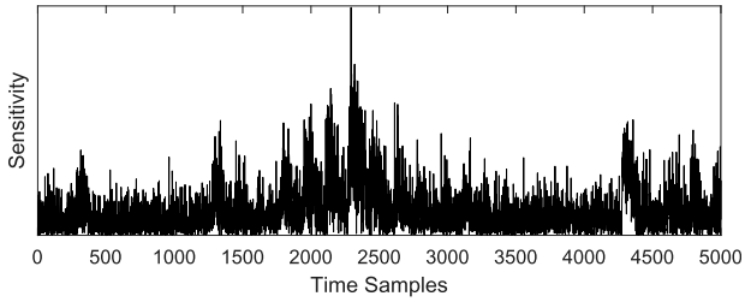

Figure 2.31 Sensitivity Analysis by Moradi, Wegner and Moos [58]

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 leakages easier with deep learning. Also, in DL-LD, 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 with 1. Both groups are mixed randomly and split by the label in a test and train dataset. If DL-LA reports a hight accuracy it is possible to detect leakages in the traces.

After splitting the dataset the neural network is trained with the training dataset. Then the model is used to classify the test dataset. After the training Sensitivity Analysis 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 [58]:

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

x_i describes the i-th input of the neural 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 [58].

The idea behind the calculation of the sensitivity is that it can be used as an indicator of whether a neural network has learned. 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. Moradi, Schneider and Moos describe in detailed the parameter of the neural network for the Deep Learning Leakage Detection. The model was created in python with the following settings [58]. The source code 2.3 shows the settings of the model for the deep learning leakage detection:

```
model = Sequential([
    Dense(120, activation = 'relu', input_shape= (tracelength,) ),
    BatchNormalization(),
    Dense(90, activation = 'relu'),
    BatchNormalization(),
    Dense(50, activation = 'relu'),
    BatchNormalization(),
    Dense(2, activation = 'softmax') ])
```

Source Code 2.3: Model of the Deep Learning Leakage Detection

Further they used the mean squared error as loss function and Adam as optimizer. ReLU was used as activation function in every layer. In the last layer softmax was used as a binary classificator. The model in chapter 3 is based on the mentioned parameter of Moradi, Schneider and Moos [58].

Chapter 3

Research

This chapter deals with the research and is divided into three parts. In all experiments the data are examined with the Sensitivity Analysis based on deep learning and then compared with the T-Test. The first experiment 3.1 consists of a baseline test. The traces contain only noise or randomly generated data. The goal is to investigate if the Sensitivity Analysis detects false positive leakages. In the second experiment in chapter 3.2 eight data sets are artificially generated with different countermeasures. It will be examined if countermeasures have an effect on the leakage detection and if the Sensitivity Analysis can find leaks in the traces. In the third experiment in chapter 3.3, real traces are measured on Arduino Due and then examined with the Sensitivity Analysis and the T-Test. The CPU in this experiment is also used in smart watches.

3.1 Experiment: Baseline Tests

In the first experiment, the T-Test and the deep learning will be tested with artificial traces without countermeasures. This experiment needs two datasets with virtual traces. The first dataset is created just with noise. In the second dataset every position of the trace are fixed except for the three fixed positions. At the three fixed positions, the values are random. The idea of these two experiments is to investigate the DL-LD with traces without any countermeasures. The T-Test is a known technique to find leakages in traces and will be used to validate the deep learning results. If none of the two techniques finds any leakages in the traces, it is a sign that noise or random data will not incorrectly detect as a leakage. Figure 3.1 represents a trace with noise, figure 3.2 represents a trace with three spots of random data.

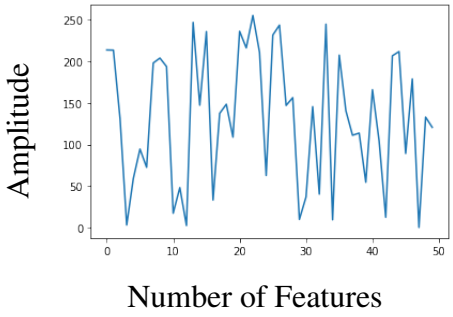

Figure 3.1 Trace with noise data

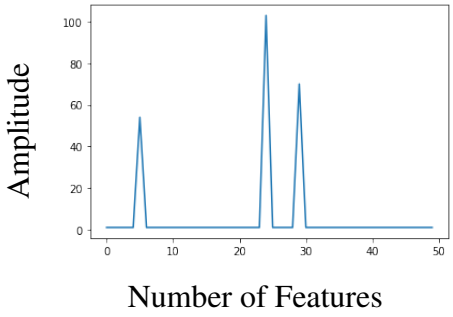

Figure 3.2 Trace with three spots of random data

3.1.1 Dataset 1: Noise

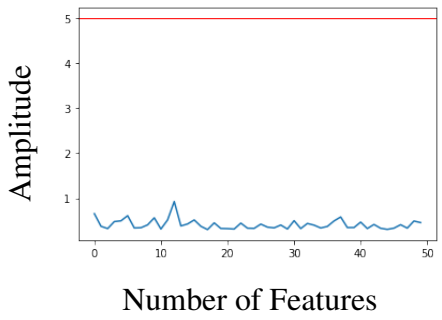

Figure 3.3 T-Test with Noise

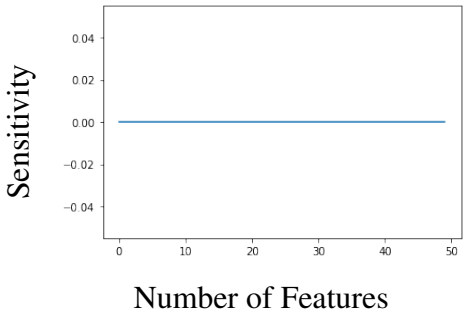

Figure 3.4 Deep Learning with Noise

In this experiment traces with noise are created. The traces do not contain any leakages or countermeasures. The dataset has a size of 10000 Traces and 50 features. The dataset will be analysed with the T-Test and deep learning.

T-Test

Figure 3.3 shows the T-Test of the traces without noise. The red line at position 5 represent the baseline. If a feature reached the baseline or below a leakage is found at this position of the trace. The experiment shows, that none of the points in the trace reaches the baseline of 5.

Sensitivity Analysis

In this test deep learning was used to analyse the traces with noise. The Sensitivity Analysis of the deep learning at figure 3.4 shows no impact and is still zero and no leakage is shown.

3.1.2 Dataset 2: Random Data

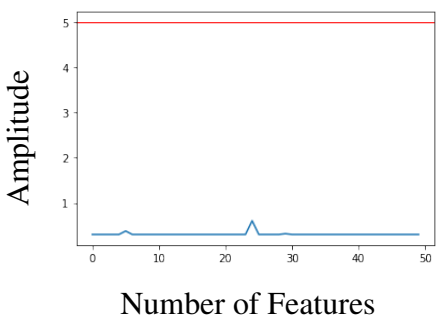

Figure 3.5 T-Test with Random Values

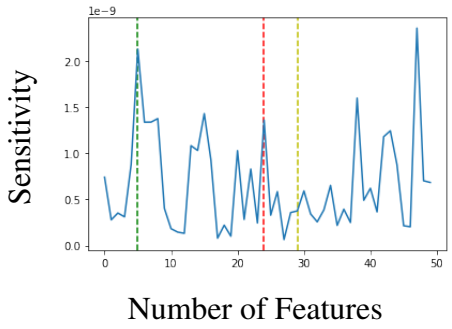

Figure 3.6 Deep Learning with Random Values

In this experiment three positions of the trace random values are created. The dataset was created artificially with a size of 10000 traces and 50 features. The traces include no leakages, noise or countermeasure. To validate the deep learning the T-Test is used as baseline.

T-Test

In this experiment figure 3.5 shows the T-Test of the dataset with 3 random values. In one of the 50 features the baseline of 5 percent quantile is not reached and no leaks are found in the traces.

Sensitivity Analysis

In the experiment with deep learning at figure 3.6 no significant leakages are found. The three marked positions (green, red and yellow) show the random data. In comparison to the other features in the trace the leakages are not clearly visible. The first (green) random shows a higher peak, but other peaks without any informations are still higher. In total the whole Sensitivity Analysis detect no significant leakages because none of the feature is reaching the 5 percent quantile.

3.1.3 Summary: Baseline Tests

		Prediction(leakage) T-Test	
		no	yes
leakage	no	2	0
	yes	0	0

Figure 3.7 Confusion Matrix T-Test Baseline Tests

		Prediction(leakage) SA	
		no	yes
leakage	no	2	0
	yes	0	0

Figure 3.8 Confusion Matrix Sensitivity Analysis Baseline Tests

The idea of the baseline test is to investigate if the Sensitivity Analysis in traces without leakages incorrectly indicates leaks, so-called false positives. If the results of the two baseline tests are summarised, it can be assumed that no leaks were found in either test. In the first test, neither the T-Test or the Sensitivity Analysis found a leak. In the second test the T-Test did not show any leaks. The Sensitivity Analysis showed spikes but all spikes were not significant enough, because none of the traces reached the 5 percent quantile. The highest point of the Sensitivity Analysis indicates a point where random data is present, but compared to other points of the Sensitivity Analysis the point is not significant. The results represented in a Confusion Matrix showed that the T-Test, as well as the Sensitivity Analysis, performed correctly (figure 3.7 and figure 3.8). In summary, the Sensitivity Analysis reliably indicates no leakage for data without leakage. Also the T-Test, which is considered a reliable method in practice, did not show any leakages. In the following experiments, it is now necessary to investigate how the Sensitivity Analysis behaves with simulated traces with countermeasures and with real data from a CPU.

3.2 Experiment: Simulated Traces

In research with side-channel attacks, it is common and a approved practice to use simulated traces. The advantage of simulated traces is that they can be generated easily and quickly with different levels of countermeasures. A disadvantage is the setup of simulated traces could be different to real traces. Section 3.2.1 deals with the data preparation and generation of the simulated traces. These experiments analyse the impact of the different kinds of countermeasure. In section 3.2.10 the results of the simulated traces are summarized.

3.2.1 Data Preparation

Dataset	Countermeasure
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. How the countermeasures work is described in the fundamentals in Section 2.4.6.

The implementation of the parameter for 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: Parameter 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 leakage point. The label is the output of the S-Box and the leakage is the label with generated noise added 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

3.2.2 Dataset 1: Noise

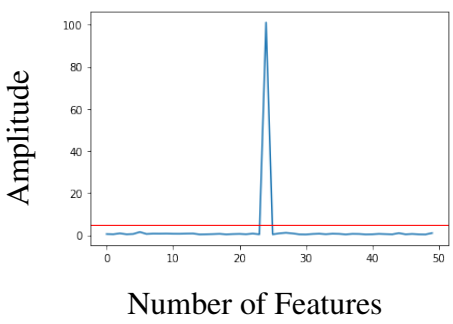

Figure 3.9 T-Test with Noise

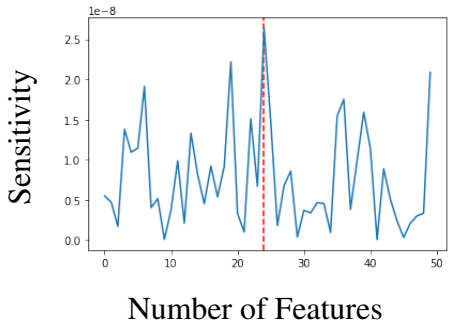

Figure 3.10 Sensitivity Analysis with Noise

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 a different plaintext. The leakage point was created at position 24. A data set of 100000 traces was created for the experiment.

T-Test

First, a T-Test was performed, followed by a deep learning test. The T-Test (figure 3.9) clearly shows a leakage at location 24 that exceeds the 5.0 make significantly. The values up to 100 are 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 a leakage.

Sensitivity Analysis

The Sensitivity Analysis in the deep learning test (figure 3.10) shows several peaks at position 7,18, 24 and 49. The highest peak is at position 24 and represents the leakage point (red line). The other points 7, 18 and 49 are random data without any informations and leaks. The classification of the DL-LA shows an accuracy of 0.983 (98.30%).

3.2.3 Dataset 2: Random Delay

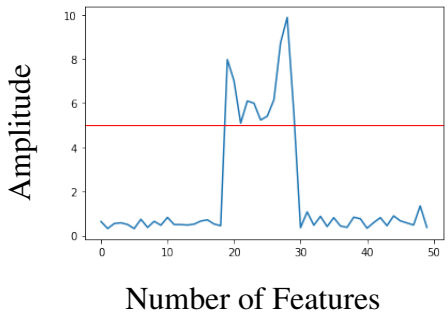

Figure 3.11 T-Test with Random Delay

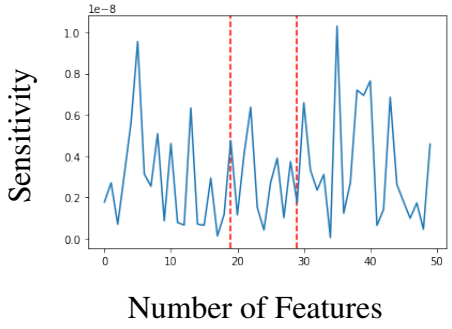

Figure 3.12 Sensitivity Analysis with Random Delay

In this experiment unprotected traces are generated with random delay. The trace contains 50 features and the leakage point is placed randomly in range from 19 to 29. The red mark in figure 3.11 shows the threshold at 5 which must be exceeded. The two red lines in figure 3.12 indicate the area where the leakage point is placed. A data set of 100000 traces was created for the experiment.

T-Test

In the T-Test 3.11, the value exceeds the value 5 percent quantile at all points in the leakage range. All other points have no noticeable increase.

Sensitivity Analysis

The plot 3.12 shows the Sensitivity Analysis of the deep learning model after training with random delay as countermeasure. The red lines represent the area were the leakage point is randomly placed. The plot shows at position 5 and at position 35 hight peaks. The leakage range starts from 19 and ends at 29. Especially in the area of the leakages there are no visible peaks. The classification of the DL-LA shows an accuracy of 0.992 (99.20%).

3.2.4 Dataset 3: Noise and 1 Mask

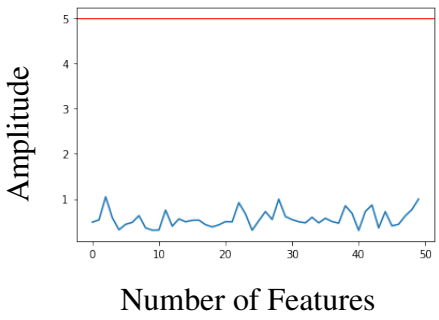

Figure 3.13 T-Test with Noise and 1 Mask

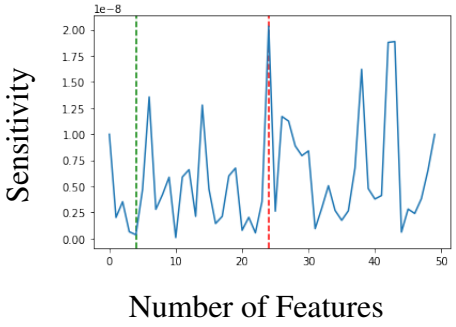

Figure 3.14 Sensitivity Analysis with Noise and 1 Mask

In this experiment a simulated trace is protected with a masking. 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. The red mark in figure 3.13 shows the threshold at 5 which must be exceeded. The two red lines in figure 3.14 indicate the leakage point, the green line represents the masking point. A data set of 100000 traces was created for the experiment.

T-Test

In the T-Test figure 3.13 no leakage point is found. None of the features is reaching the 5 percent quantile.

Sensitivity Analysis

The plot 3.14 shows the Sensitivity Analysis 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 informations about any leaks. Every other point in the trace is random. The classification of the DL-LA shows an accuracy of 0.981 (98.10%).

3.2.5 Dataset 4: Random Delay and 1 Mask

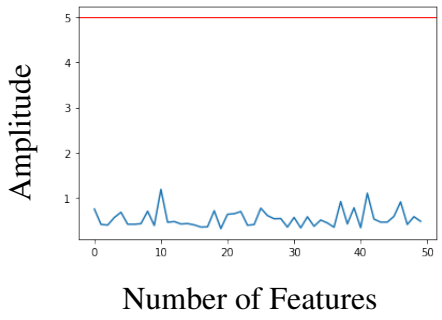

Figure 3.15 T-Test with Random Delay and 1 Mask

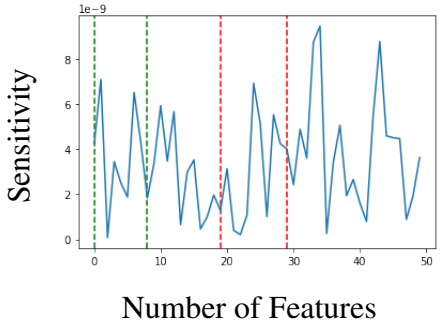

Figure 3.16 Sensitivity Analysis with Random Delay and 1 Mask

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 3.15 shows the threshold at 5 which must be exceeded. The two red lines in figure 3.16 indicate the range of the leakage points from 19 to 29, the green lines represent the range of the masking points from 0 to 9. A data set of 120000 traces was created for the experiment.

T-Test

In the T-Test figure 3.15 no leakage point is found. None of the features is reaching the 5 percent quantile.

Sensitivity Analysis

The 3.16 presents the Sensitivity Analysis 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. The classification of the DL-LA shows an accuracy of 0.767 (76.70%).

3.2.6 Dataset 5: Noise and 2 Masks

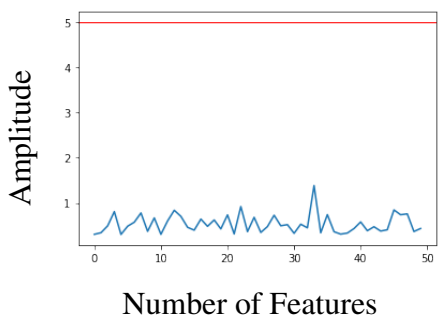

Figure 3.17 T-Test with Noise and 2 Masks

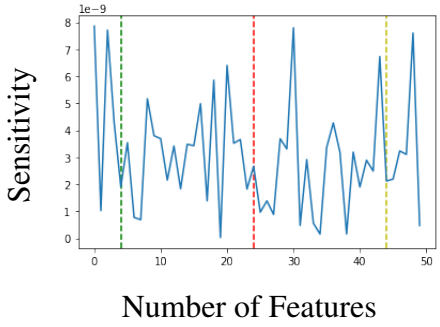

Figure 3.18 Sensitivity Analysis with Noise and 2 Masks

This experiment simulates traces which are protected by two masks and is more complex and difficult to be attacked. The implementation of these protection, 3 data points were modified in the simulated traces. The red mark in figure 3.17 shows the threshold at 5 which must be exceeded. The red line in figure 3.18 indicates the leakage point at position 24, the green lines represent the first mask at point 4 and the yellow point at position 44 represent the second mask. A data set of 200000 traces was created for the experiment.

T-Test

In the T-Test figure 3.17 no leakage point is found. None of the features is reaching the 5 percent quantile.

Sensitivity Analysis

In this experiment a dataset with 2 masks and 1 leakage point was used for deep learning. Figure 3.18 represents 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 Analysis. The classification of the DL-LA shows an accuracy of 0.708 (70.80%).

3.2.7 Dataset 6: Random Delay and 2 Masks

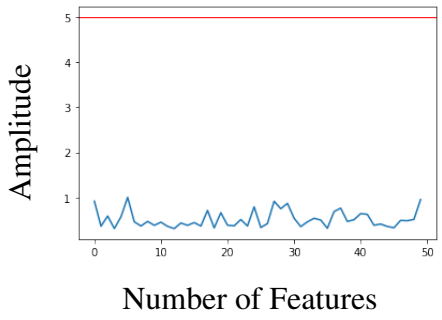

Figure 3.19 T-Test with Random Delay and 2 Masks

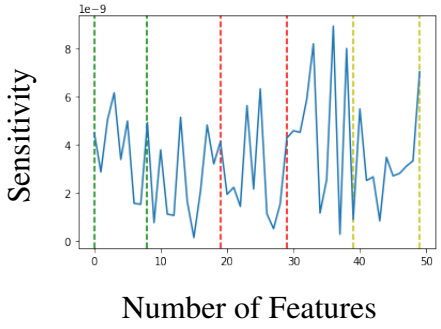

Figure 3.20 Sensitivity Analysis with Random Delay and 2 Masks

This experiment simulates traces which are protected by two masks and random delay. The red mark in figure 3.19 shows the threshold at 5 which must be exceeded. The red lines in figure 3.20 indicates the leakage points range from 19 to 29, the green lines represent the range of the first mask from 0 to 9 and the yellow lines the range of the second mask at from 39 to 49. A data set of 200000 traces was created for the experiment.

T-Test

In the T-Test figure 3.19 no leakage point is found. None of the features is reaching the 5 percent quantile.

Sensitivity Analysis

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 Analysis 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. The classification of the DL-LA shows an accuracy of 0.794 (79.40%).

3.2.8 Dataset 7: Noise and 3 Masks

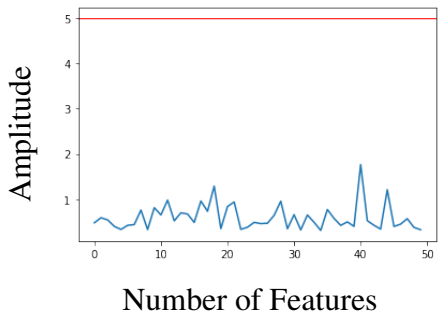

Figure 3.21 T-Test with Noise and 3 Masks

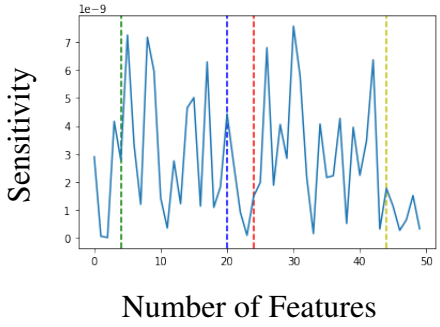

Figure 3.22 Sensitivity Analysis with Noise and 3 Masks

This experiment simulates traces which are protected by three masks and noise. The red mark in Figure 3.21 shows the threshold at 5 which must be exceeded. The red lines in figure 3.22 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 1800000 traces was created for the experiment.

T-Test

In the T-Test figure 3.21 no leakage point is found. None of the features is reaching the 5 percent quantile.

Sensitivity Analysis

The Figure 3.22 shows the Sensitivity Analysis 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. No leaks were found with the Sensitivity Analysis. The classification of the Dl-LA shows an accuracy of 0.734 (73.40%).

3.2.9 Dataset 8: Random Delay and 3 Masks

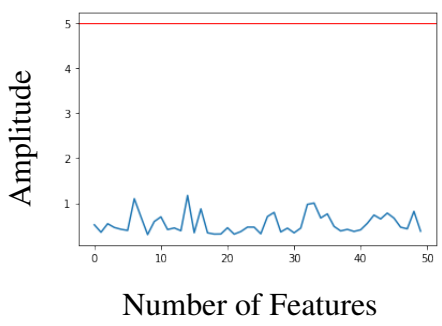

Figure 3.23 T-Test with Random Delay and 3 Masks

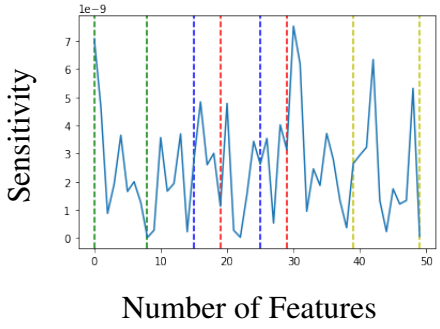

Figure 3.24 Sensitivity Analysis with Random Delay and 3 Masks

This experiment simulates traces which are protected by three masks and noise. The red mark in figure 3.23 shows the threshold at 5 which must be exceeded. The red lines in figure 3.24 indicate the leakage points range from 19 to 29, the green lines represent the range of the first mask from 0 to 9 and the yellow lines the range of the second mask from 39 to 49. The blue lines represent the area with the third mask from 16 to 24, the leakage point overlaps with one of the masks. A data set of 1800000 traces was created for the experiment.

T-Test

In the T-Test figure 3.23 no leakage point is found. None of the features is reaching the 5 percent quantile.

Sensitivity Analysis

The Sensitivity Analysis at figure 3.24 is based on a dataset with three 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. The classification of the DL-LA shows an accuracy of 0.696 (69.60%).

3.2.10 Summary: Simulated Traces

		Prediction(leakage) T-Test	
		no	yes
leakage	no	6	0
	yes	0	2

Figure 3.25 Confusion Matrix T-Test Simulated Traces

		Prediction(leakage) SA	
		no	yes
leakage	no	0	0
	yes	8	0

Figure 3.26 Confusion Matrix Sensitivity Analysis Simulated Traces

The series of tests with simulated traces are useful only to a limited extent for Sensitivity Analysis. Already at the beginning the sensitivity shows leaks inaccurately even in the traces without countermeasure. However, since the data were created artificially the leaks are manifest only at one data point. The T-Test with random delay scored significantly better than that with Sensitivity Analysis. It also showed that based on the accuracy deep learning leaks 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 experiments, no significant leakage could be found with the help of Sensitivity Analysis 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 Sensitivity Analysis reacts differently to specific countermeasures. The random delay seems to be a more significant challenge for the neural network than noise.

A confusion matrix of the results (figure 3.25 and figure 3.26) shows Sensitivity Analysis with the current set-up scores worse than the T-Test. Of eight experiments, six were true negative and two true positive in the T-test. In comparison to the Sensitivity Analysis, all eight experiments were false negative. On the other hand the accuracy of each experiment has shown, that the deep learning based Leakage detection (DL-LA) can break countermeasures but the result with Sensitivity Analysis were not able to detect the leakage points accurately. Table 3.2 shows the accuracy of each experiment with the simulated traces.

Dataset	Countermeasure	Accuracy
1.	Noise	0.983
2.	Random Delay	0.992
3.	Noise and 1 Mask	0.981
4.	Random Delay and 1 Mask	0.767
5.	Noise and 2 Masks	0.708
6.	Random Delay and 2 Masks	0.794
7.	Noise and 3 Masks	0.734
8.	Random Delay and 3 Masks	0.696

Table 3.2: Accuracy of the Simulated Traces

3.3 Experiment: Real Traces

This experiment investigates how the T-Test and DL-LD behave on a CPU. Table 3.3 lists the used datasets and countermeasures for the experiments with real traces.

Dataset	Countermeasure
1.	Unprotected
2.	Random Delay and 1 Mask

Table 3.3: Datasets of Real Traces and Countermeasures

3.3.1 Microcontroller Implementation

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 and errors are not 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 interface 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 randomness, the Marsaglia Xorshift Random Generator [37] 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. Depending on the static parameters the randomness can be controlled, and it is possible to reproduce the same randomness multiple times. 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: Implementation of the Marsaglia Xorshift Random Generator

3.3.2 Grid Search

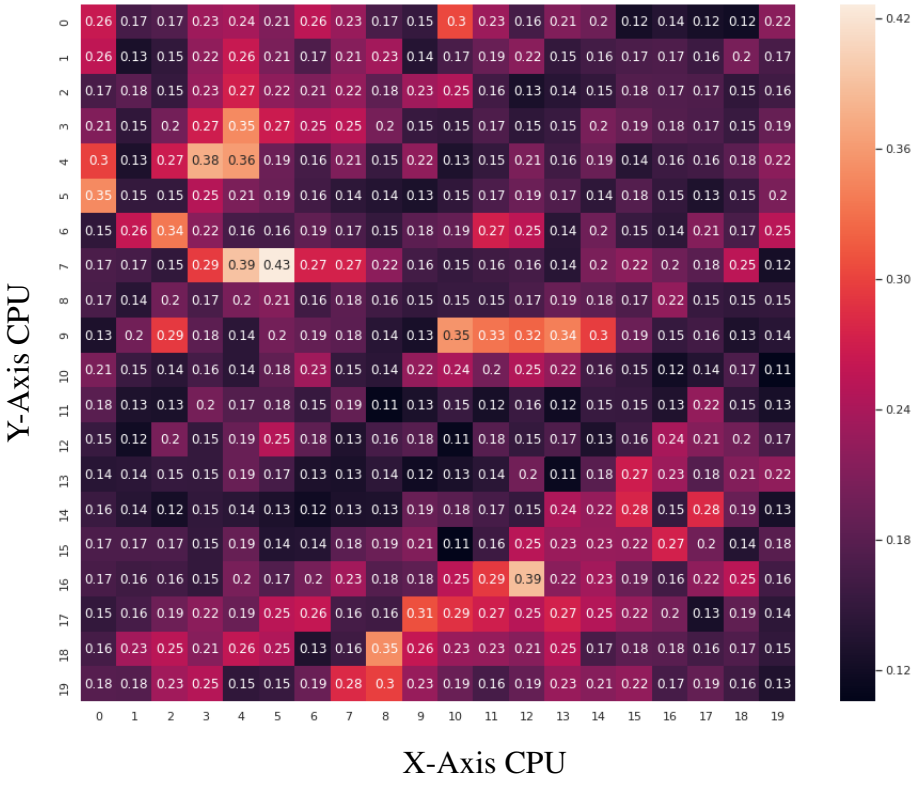

Figure 3.27 Heatmap with 400 measured correlation points at the CPU

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 400 different positions. A correlation attack follows each of the 400 results. The better the position on the CPU, the higher the results of the correlation attack. Subsequently, all 400 results are plotted as a heat map. The heat map visualizes which positions are best suited for the measurements. Figure 3.27 shows a heatmap with the individual correlations.

First results showed that the correlation is much too low at each point. It can depend on the CPU and its architecture. Furthermore, the measuring instruments can influence the quality of the measurements. Improving 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.

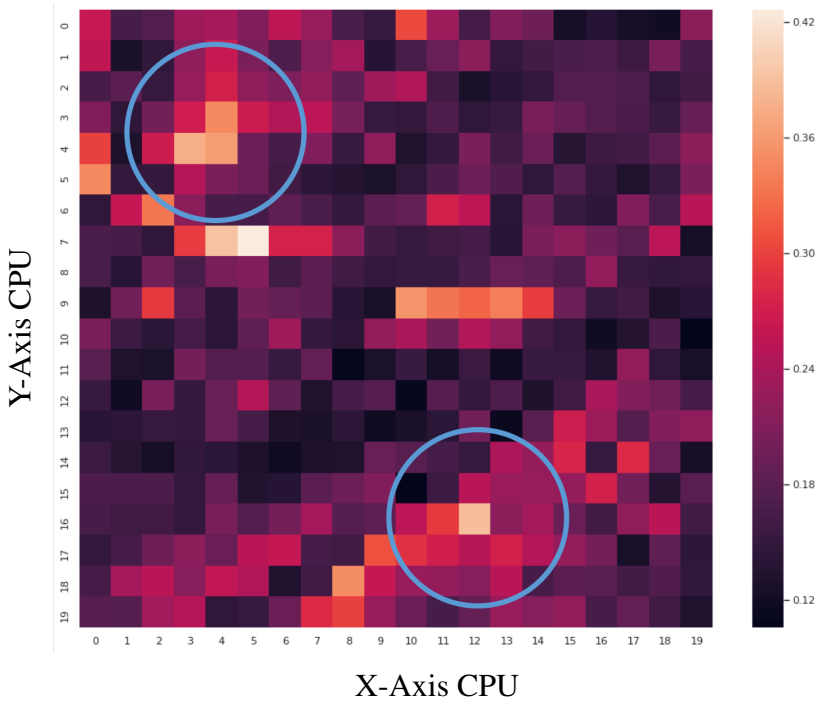

Figure 3.28 Heatmap with two possible positions for the measurement

Figure 3.28 shows two areas suitable for measurements. To perform a grid search on the CPU technical requirements were necessary because the manual placement of the sensor on the CPU can be very inaccurate at 400 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 control of the Sensor at the CPU

Figure 3.29 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 3.29 the sensor is placed at the marked position in figure 3.28 to measure the traces.

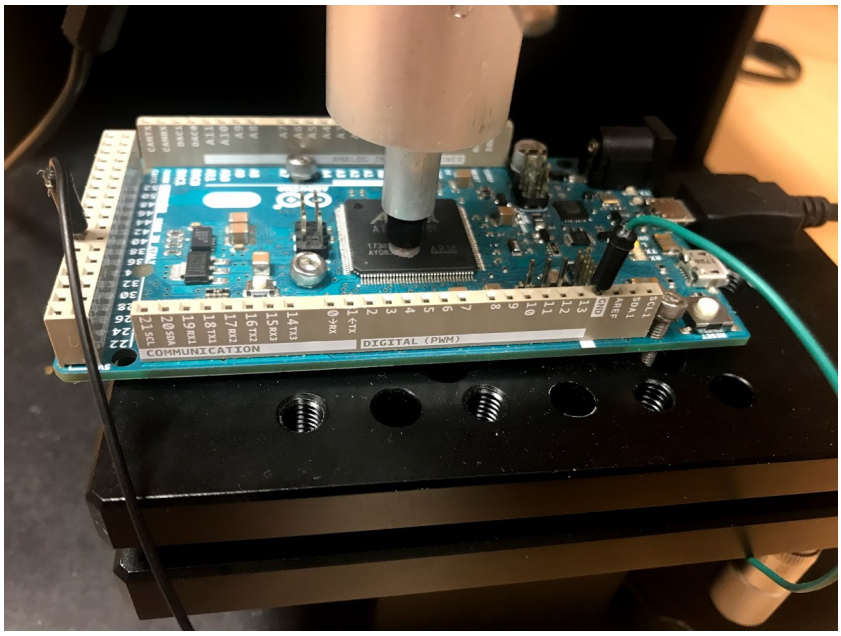

Figure 3.29 Arduino Due with a Sensor placed on the CPU for the Measurements

3.3.3 Analyse Signal to Noise Ratio

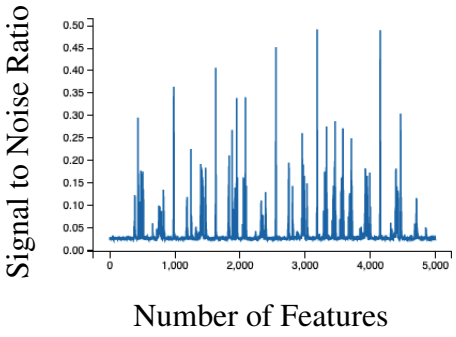

Figure 3.30 SNR Unprotected Traces

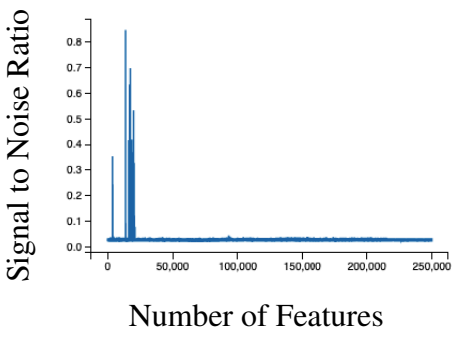

Figure 3.31 SNR Protected Traces - 10 Rounds AES

After the measurement the SNR of the datasets was analysed. Figure 3.30 reveals the SNR of the traces with an unprotected AES encryption. The information gain is clearly visible at the high peaks. At figure 3.31 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 challenge is the amount of data being to large to compute the functions. The first dataset consists of 20000 traces and 5000 features per trace. The second data set consists of more than 40000 traces with 250000 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 3.32 shows in pseudo code the implementation of the SNR function. At line 6 and 7 the dataset will be read row wise and stored temporarily 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 3.32 Pseudo Code: SNR Calculation by Row

3.3.4 Dataset 1: Unprotected

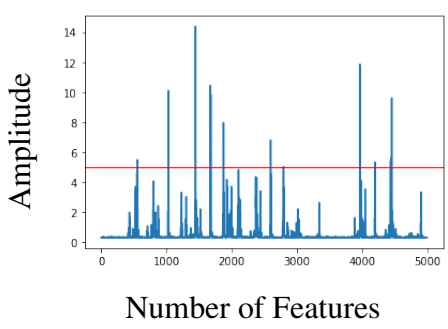

Figure 3.33 T-Test: Real Trace Unprotected

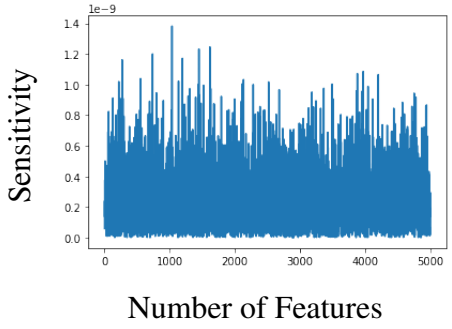

Figure 3.34 Sensitivity Analysis with Real Traces Unprotected

For this experiment, 80000 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 5000 features and contain the first round of AES encryption.

T-Test

Figure 3.33 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.

Sensitivity Analysis

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 correct approximately and do not show any leaks. The classification of the DL-LA shows an accuracy of 0.957 (95.70%).

3.3.5 Dataset 2: Protected with Random Delay and 1 Mask

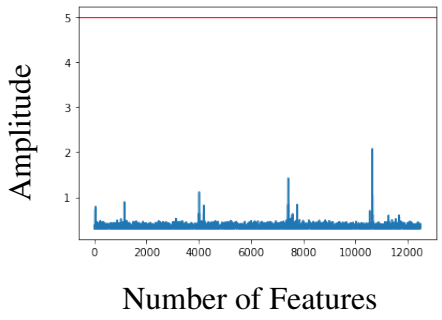

Figure 3.35 T-Test: Real Trace Protected

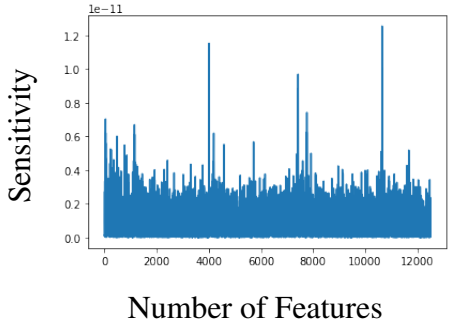

Figure 3.36 Sensitivity Analysis with Real Traces Protected

In this experiment 10000000 protected traces have been created. Next averaging was performed to reduce the number of traces in the dataset to 1000000 entries. Masking and random delay protect the traces. The traces contain 12000 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.

Sensitivity Analysis

This experiment investigates the real traces with deep learning. The result is similar to the results of the T-Test. The sensitivity shows three peaks between the points 4000 and 11000. 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. The classification of the DL-LA shows an accuracy of 0.920 (92.00%).

3.3.6 Summary: Real Traces

		Prediction(leakage) T-Test	
		no	yes
leakage	no	1	0
	yes	0	1

Figure 3.37 Confusion Matrix T-Test Real Traces

		Prediction(leakage) SA	
		no	yes
leakage	no	0	0
	yes	2	0

Figure 3.38 Confusion Matrix Sensitivity Analysis 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 Sensitivity Analysis 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. Figure 3.37 and 3.38 represent the results in form of a confusion matrix. From the two experiments the T-Test reached a true negative and true positive, the Sensitivity Analysis two false negatives. But the accuracy of the two experiments has shown that deep learning-based leakage detection (DL-LA) can break countermeasures. Sensitivity Analysis was not able to detect the leakage points accurately. Table 3.4 shows the accuracy of each experiment with the real traces. The accuracy after the classification of the unprotected traces reached 95.70% and 92.20% with the protected traces.

Dataset	**Countermeasure**	**Accuracy**
1.	Unprotected	0.957
2.	Random Delay and 1 Mask	0.920

Table 3.4: Accuracy of the Real Traces

Chapter 4

Discussion and Future Work

A detailed analysis of the research with regard to chapter 1.2 as well as to provide an outlook on further research leads to the following results:

In summary, it can be said that the basement tests have shown that noise and randomly generated data have no effect on false positives. The result of the baseline test is essential. If the Sensitivity Analysis already detects leaks at this stage, it can be assumed that noise or randomly generated data will lead to false detections. The detection of false positives is necessary to analyse the correctness of the Sensitivity Analysis and is based on the principle of baseline tests. The Sensitivity Analysis with the parameters used by Moradi, Schneider and Moos [58] cannot be applied to a CPU on Arduino Due. It also turned out that the parameters cannot be applied to simulated traces. The Sensitivity Analysis could not find the position of leaks in any of the data. The accuracy with the protected traces reached 92%. A classification of such height is evidence that the DL-LD found leakages in traces even if the Sensitivity Analysis can not find the exact positions. Based on the obtained results, no effects of countermeasures on Sensitivity Analysis can be detected. Comparing Sensitivity Analysis with the T-Test, the T-Test is very reliable and independent of parameters. Yet, without knowing the exact parameters, it is impossible to find leakages with the Sensitivity Analysis in a new CPU. It is possible that optimizations of the neural network may improve the results. The experiments have shown that the T-Test struggles to perform as the countermeasures become complex or combinations of countermeasures are used. On the other hand, deep learning based detection has shown decent accuracy across datasets both for simulated and real measurements. However, the Sensitivity Analysis did not work as expected. Thus deep learning based detection (DL-LD) has good potential for leakage detection but Sensitivity Analysis for feature detection needs further investigation.

If the results are considered from the economic point of view, some important consequences emerge. The use of security analysis methods such as the T-Test or Sensitivity Analysis differs in practice and science. In science, it is primarily assumed that the user has a profound understanding of mathematical principles and the theoretical foundations. Besides, science assumes that all steps of a procedure can be reproduced. Also economical consequences are not part of scientific research they are definitely of importance for further usage. The user applies the procedures for the analysis of safety-relevant processes. Since a user does not have the mathematical basics of a scientist, preference is given to applications and tools that are low in complexity and designed to be user-friendly. The previous use of the deep learning based detection with Sensitivity Analysis is based on the use in a scientific environment.

The research of this thesis was divided into several areas: First, different scenarios were examined for possible false positives in a baseline test, then traces with and without countermeasures were examined in experiments. In order to adjust the procedure for practical use, the tests could be divided into different levels. The lower the level, the more critical the tests are. Level 1 tests are of fundamental importance, only if level 1 tests are successful the test continues with level 2. E.g. level 1 tests could be the baseline tests. Subsequently, level 2 tests could be performed on traces where encryption was used without countermeasures. Tests on level 3 could be with encryption and countermeasures. The classification into levels can be represented graphically in an application e.g. green for successful and red for failed tests. Additional the accuracy of the DL-LD can be used as an indicator whether leaks were found in the traces. Such an application would increase the acceptance and usability in practice without complex previous knowledge.

The research results have shown that Sensitivity Analysis as implemented in its current form using the parameters from previous work did not show impressive results and further investigation is needed to find correct parameters and settings for each target case. The future question is if the parameters can lead to the Sensitivity Analysis detecting leaks in simulated traces as well as in real traces. Another question that has not yet been clarified is what effects countermeasures have on Sensitivity Analysis. Similar to the paper from Kim et al. [25], it could be investigated whether adding artificial noise improves the results of Sensitivity Analysis. It could also be investigated whether neural networks like CNN have an impact on anomaly detection. Previously an MLP was always used for Sensitivity Analysis. However, research in other areas of side-channel attacks has shown that MLPs and CNNs react differently to specific countermeasures. Also, it has not yet been investigated whether a network especially adapted for one CPU can be applied to other CPUs. In addition it would have to be investigated which adjustments would be

necessary if the neural network could be transferred to identical CPUs. So far it has not been investigated whether encryption modes such as GCM have an effect on Sensitivity Analysis. The results in this thesis have shown the difficulty to interpret results of the Sensitivity Analysis in contrast to the T-Test. The use of the Sensitivity Analysis makes sense only if it is possible to know which parameters are needed for the Sensitivity Analysis or if techniques exist to find the right parameters. This may lead to further research. In addition, it is yet not possible to interpret the Sensitivity Analysis correctly. The Sensitivity Analysis should be used as a supplement to the T-Test and cannot replace it. The T-Test has higher reliability at the current state of the art because no parameters are needed for the adjustment. During the experiments, it was shown that the deep learning consumes many resources. It is not possible to split up calculations during the deep leaning process and perform them independently. During deep learning, all data are always required during the learning process and must be stored in the memory. If deep learning could be a reliable method, it will be necessary to investigate how it can be optimised in the future.

Deep learning as part of Artificial Intelligence is significantly increasing in a wide range of applications. Therefore all aspects of security will require further development in order to keep a high standard to maintain user acceptance. Research in comparing available tools in detail for practical use to detect security gaps will remain a constant challenge.

Chapter 5

Bibliography

[1] Arduino.cc. *Arduino Due | Arduino Official Store*. URL: https://store.arduino.cc/arduino-due (visited on 12/29/2019).

[2] Atmel. "Sam3X / Sam3a [Datasheet]". In: (2015), p. 1450. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-11057-32-bit-Cortex-M3-Microcontroller-SAM3X-SAM3A%7B%5C_%7DDatasheet.pdf.

[3] Timo Bartkewitz and Kerstin Lemke-Rust. "Efficient template attacks based on probabilistic multi-class support vector machines". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2013. ISBN: 9783642372872. DOI: 10.1007/978-3-642-37288-9_18.

[4] Ryad Benadjila et al. *Study of Deep Learning Techniques for Side-Channel Analysis and Introduction to ASCAD Database-Long Paper*. Tech. rep. 2018.

[5] 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.

[6] Begül Bilgin et al. "Higher-order threshold implementations". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 8874. 2014, pp. 326–343. DOI: 10.1007/978-3-662-45608-8_18.

[7] Riggio Christopher. *What's the deal with Accuracy, Precision, Recall and F1?* 2019. URL: https://towardsdatascience.com/whats-the-deal-with-accuracy-precision-recall-and-f1-f5d8b4db1021 (visited on 07/06/2020).

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

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

[10] Arden Dertat. *Applied Deep Learning - Part 1: Artificial Neural Networks*. 2017. URL: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2 (visited on 06/26/2020).

[11] A Adam Ding, Cong Chen, and Thomas Eisenbarth. "Simpler, faster, and more robust t-test based leakage detection". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9689. 2016, pp. 163–183. ISBN: 9783319432823. DOI: 10.1007/978-3-319-43283-0_10.

[12] Divyansh Dwivedi. *Machine Learning For Beginners - Towards Data Science*. 2018. URL: https://towardsdatascience.com/machine-learning-for-beginners-d247a9420dab (visited on 02/26/2020).

[13] 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.

[14] 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*. 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/ (visited on 12/29/2019).

[15] Benedikt Gierlichs et al. "Empirical comparison of side channel analysis distinguishers on DES in hardware". In: *ECCTD 2009 - European Conference on Circuit Theory and Design Conference Program*. 2009. ISBN: 9781424438969. DOI: 10.1109/ECCTD.2009.5275003.

[16] Richard Gilmore, Neil Hanley, and Maire O'Neill. "Neural network based attack on a masked implementation of AES". In: *Proceedings of the 2015 IEEE International Symposium on Hardware-Oriented Security and Trust, HOST 2015*. Institute of Electrical and Electronics Engineers Inc., 2015. ISBN: 9781467374200. DOI: 10.1109/HST.2015.7140247.

[17] Google. *Host-based card emulation overview | Android-Entwickler*. URL: https://developer.android.com/guide/topics/connectivity/nfc/hce.html (visited on 01/30/2020).

[18] 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.

[19] David Hutchison and John C Mitchell. *Constructive Side-Channel Analysis and Secure Design*. Ed. by Ilia Polian and Marc Stöttinger. Vol. 9. Lecture Notes in Computer Science. Springer International Publishing, 2012. ISBN: 9783540465591. DOI: 10.1016/0020-7101(78)90038-7. URL: http://link.springer.com/10.1007/978-3-030-16350-1%20http://www.mendeley.com/research/lecture-notes-computer-science-2/.

[20] Sunghyun Jin et al. "Recent advances in deep learning-based side-channel analysis". In: *ETRI Journal* 42.2 (2020), pp. 292–304. ISSN: 22337326. DOI: 10.4218/etrij.2019-0163. URL: https://onlinelibrary.wiley.com/doi/abs/10.4218/etrij.2019-0163.

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

[22] Balasch Josep, Gierlichs Benedikt, and Grosso Vincent. "On the Cost of Lazy Engineering for Masked Software Implementations". In: (2014). ISSN: 16113349. DOI: 10.1007/978-3-319-16763-3.

[23] Nahua Kang. *Multi-Layer Neural Networks with Sigmoid Function - Deep Learning for Rookies (2)*. URL: https://towardsdatascience.com/multi-layer-neural-networks-with-sigmoid-function-deep-learning-for-rookies-2-bf464f09eb7f (visited on 12/25/2019).

[24] Jaehun Kim et al. "Make Some Noise Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis". In: *tCHES 2019* (2019). URL: https://hal.inria.fr/hal-02010599.

[25] Jaehun Kim et al. “Make Some Noise Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis Make Some Noise: Unleashing the Power of Convolutional Neural Networks for Profiled Side-channel Analysis”. In: (2019). DOI: 10.13154/tches.v2019.i3.148-179Ãŕ. URL: https://hal.inria.fr/hal-02010599.

[26] JooHyung Kim. *What is AES? — Step by Step - zeroFruit - Medium*. URL: https://medium.com/@14wnrkim/what-is-aes-step-by-step-fcb2ba41bb20 (visited on 12/29/2019).

[27] Takaya Kubota et al. “Deep Learning Side-Channel Attack Against Hardware Implementations of AES”. In: *Proceedings - Euromicro Conference on Digital System Design, DSD 2019*. Institute of Electrical and Electronics Engineers Inc., 2019. ISBN: 9781728128610. DOI: 10.1109/DSD.2019.00046.

[28] Andrew J Leiserson, Mark E Marson, and Megan A Wachs. *Gate-Level Masking Under a Path-Based Leakage Metric*. Tech. rep. 2014.

[29] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. “A machine learning approach against a masked AES: Reaching the limit of side-channel attacks with a learning model”. In: *Journal of Cryptographic Engineering* 5 (2015). ISSN: 21908516. DOI: 10.1007/s13389-014-0089-3.

[30] Liran Lerman, Gianluca Bontempi, and Olivier Markowitch. “Power analysis attack: An approach based on machine learning”. In: *International Journal of Applied Cryptography* 3 (2014). ISSN: 17530571. DOI: 10.1504/IJACT.2014.062722.

[31] Arm Limited. *Microprocessor Cores and Technology – Arm*. URL: https://www.arm.com/products/silicon-ip-cpu (visited on 11/12/2019).

[32] Vikashraj Luhaniwal. *Analyzing different types of activation functions in neural networks-which one to prefer?* URL: https://towardsdatascience.com/analyzing-different-types-of-activation-functions-in-neural-networks-which-one-to-prefer-e11649256209 (visited on 12/25/2019).

[33] Houssem Maghrebi. “Deep Learning based Side Channel Attacks in Practice”. 2019. URL: https://eprint.iacr.org/2019/578.pdf.

[34] Hamza Mahmood. *Hamza Mahmood - Towards Data Science*. URL: https://towardsdatascience.com/softmax-function-simplified-714068bf8156 (visited on 12/25/2019).

[35] Ahmed Mahmoud et al. “Combined Modeling and Side Channel Attacks on Strong PUFs”. In: *IACR eprint-2013-632* 2013 (2013). URL: http://www.aceslab.org/sites/default/files/632.pdf.

[36] 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.

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

[38] Merchant Savvy. *50+ Global Mobile Payment Stats, Data & Trends (Feb 2020)*. 2020. URL: https://www.merchantsavvy.co.uk/mobile-payment-stats-trends/ (visited on 05/14/2020).

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

[40] NATIONAL INSTRUMENTS CORP. *Understanding Frequency Performance Specifications - National Instruments*. 2020. URL: https://www.ni.com/de-de/support/documentation/supplemental/06/understanding-frequency-performance-specifications.html (visited on 07/08/2020).

[41] NVIDIA-Corporation. *Deep Learning | NVIDIA Developer*. URL: https://developer.nvidia.com/deep-learning (visited on 01/30/2020).

[42] Jain Pawan. *Complete Guide of Activation Functions - Towards Data Science*. 2019. URL: https://towardsdatascience.com/complete-guide-of-activation-functions-34076e95d044 (visited on 06/26/2020).

[43] Guilherme Perin, Baris Ege, and Jasper Van Woudenberg. “Lowering the Bar: Deep Learning for Side-Channel Analysis (White-Paper)”. In: *Black Hat, USA* (2018).

[44] 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.

[45] Emmanuel Prouff and Matthieu Rivain. *Masking against side-channel attacks: A formal security proof*. Tech. rep. 2013, pp. 142–159. DOI: 10.1007/978-3-642-38348-9_9.

[46] Zheng Qi and Tao Long. *System and methods for side-channel attack prevention, Broadcom Corporation, Irvine, CA (US)*. Patent No.: US 8,781,111 B2. 2014.

[47] Ludovic Rembert. *What is AES Encryption? A Beginner Friendly Guide - Privacy Canada*. 2020. URL: https://privacycanada.net/aes-encryption-guide/ (visited on 03/10/2020).

[48] Margaret Rouse. *What is Advanced Encryption Standard (AES)? - Definition from WhatIs.com*. URL: https://searchsecurity.techtarget.com/definition/Advanced-Encryption-Standard (visited on 11/12/2019).

[49] Sarang Narkhede. *Understanding Confusion Matrix – Towards Data Science*. 2018. URL: https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62 (visited on 07/06/2020).

[50] Pascal Sasdrich et al. "Side-channel protection by randomizing look-up tables on reconfigurable hardware pitfalls of memory primitives". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. Vol. 9064. 2015, pp. 95–107. ISBN: 9783319214757. DOI: 10.1007/978-3-319-21476-4_7.

[51] Tobias Schneider and Amir Moradi. *Leakage Assessment Methodology - a clear roadmap for side-channel evaluations*. Cryptology ePrint Archive, Report 2015/207. 2015. URL: https://eprint.iacr.org/2015/207.

[52] François Xavier Standaert. "How (not) to use welch's T-test in side-channel security evaluations". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2019. ISBN: 9783030154615. DOI: 10.1007/978-3-030-15462-2_5.

[53] Statista. *Global mobile payment usage penetration by region 2018 l Statistic | Statista*. URL: https://www-statista-com.sire.ub.edu/statistics/820853/used-a-mobile-payment-service-in-the-last-month-region/ (visited on 05/12/2020).

[54] Statista. "Mobile Payment Usage Worldwide". In: *Statista* (2019). URL: https://www.statista.com/study/39303/mobile-paymentusage-%20worldwide/.

[55] Benjamin Timon. *Non-Profiled Deep Learning Side-Channel attacks with Sensitivity Analysis*. URL: https://medium.com/eshard/non-profiled-deep-learning-side-channel-attacks-with-sensitivity-analysis-cb164c0a3ae2 (visited on 04/30/2020).

[56] Benjamin Timon. "Non-Profiled Deep Learning-based Side-Channel attacks with Sensitivity Analysis". In: *tCHES 2019* 2019 (2019). DOI: 10.13154/tches.v2019.i2.107-131. URL: https://tches.iacr.org/index.php/TCHES/article/view/7387.

[57] Huanyu Wang et al. “How Diversity Affects Deep-Learning Side-Channel Attacks”. In: *2019 IEEE Nordic Circuits and Systems Conference, NORCAS 2019: NORCHIP and International Symposium of System-on-Chip, SoC 2019 - Proceedings*. 2019. ISBN: 9781728127699. DOI: 10.1109/NORCHIP.2019.8906945.

[58] Felix Wegener, Thorben Moos, and Amir Moradi. *DL-LA: Deep Learning Leakage Assessment: A modern roadmap for SCA evaluations*. Cryptology ePrint Archive, Report 2019/505. 2019. URL: https://eprint.iacr.org/2019/505.

[59] Lingxiao Wei et al. “I know what you see: Power side-channel attack on convolutional neural network accelerators”. In: *ACM International Conference Proceeding Series*. ACM, 2018. ISBN: 9781450365697. DOI: 10.1145/3274694.3274696. eprint: 1803.05847v2. URL: https://doi.org/10.1145/3274694.3274696.

[60] Shuai Wei Zhang et al. “A Highly Effective Data Preprocessing in Side-Channel Attack Using Empirical Mode Decomposition”. In: *Security and Communication Networks* 2019 (2019). ISSN: 19390122. DOI: 10.1155/2019/6124165.

[61] Leo Zou. *Some suggestions on ARM-based smart watch silicon solution - Wearables forum - Wearables - Arm Community*. URL: https://community.arm.com/iot/wearables/f/discussions/2603/some-suggestions-on-arm-based-smart-watch-silicon-solution (visited on 05/13/2020).

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