FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Open-source system for cloning of RFID/NFC cards and tags
Student:	Michal Beneš
Supervisor:	Ing. Jan Fesl, Ph.D.
Study program:	Informatics
Branch / specialization:	Computer Security and Information technology
Department:	Department of Computer Systems
Validity:	until the end of summer semester 2024/2025

Instructions

Open-source system for cloning RFID/NFC cards and tags
Radio-Frequency Identification (RFID) and Near Field Communication (NFC) technologies have become essential for many industrial and commercial applications, including logistics, healthcare, industrial and security. Special cards/tags are usually required to use these technologies effectively. This bachelor thesis will focus on the development of an open-source device that can clone and emulate RFID/NFC cards/tags from different manufacturers.
Aims of the bachelor thesis:
1) Design an open-source device architecture for cloning RFID/NFC cards/tags (both software and hardware) with respect to modularity and future extensibility. The device must function as a standalone unit without the need to interface with a personal computer. Use a commercially available microcomputer such as Raspberry Pi, Arduino or similar to implement the device.
2) Develop software in C++ or Python programming language to clone and emulate RFID/NFC cards/tags of at least 3 different types such as Mifare Classis , Mifare Ultralight, Mifare Desfire v1 etc. Debug the software directly in the environment of the chosen microcomputer.
3) Implement a user interface to control the device that allows the user to easily interact with the device and use its features.
4) Test the finished device on different types of RFID/NFC cards/tags and evaluate its functionality.

Electronically approved by prof. Ing. Pavel Tvrdík, CSc. on 10 December 2023 in Prague.

Bachelor's thesis

OPEN-SOURCE SYSTEM FOR CLONING RFID/NFC CARDS AND TAGS

Michal Beneš

Faculty of Information Technology
Department of Information Security
Supervisor: Ing. Jan Fesl, Ph.D.
May 15, 2024

Contents

Acknowledgments vi

Declaration vii

Abstract viii

List of Abbreviations ix

Introduction 1

1 Goals 3

2 Theoretical Background 4
2.1 Definition of RFID and NFC . . . 4
2.1.1 Definition of RFID . . . 4
2.1.2 Definition of NFC . . . 5
2.2 International Standards . . . 5
2.3 Magic Cards . . . 6
2.4 Types of Tags and Their Characteristics . . . 7
2.4.1 EM410x Tags . . . 7
2.4.2 Mifare Ultralight . . . 8
2.4.3 Mifare Classic . . . 8
2.4.4 Mifare DESFire . . . 10
2.4.5 Legic Prime . . . 10

3 Analysis 12
3.1 Requirements . . . 12
3.1.1 RFID/NFC Reader Requirements . . . 12
3.1.2 Microcomputer Requirements . . . 13
3.1.3 User Interface Requirements . . . 13
3.2 Existing RFID/NFC Cloners . . . 13
3.2.1 Flipper Zero . . . 13
3.2.2 Chameleon Mini . . . 13
3.2.3 Chameleon Ultra . . . 14
3.2.4 NXP PN532 Module . . . 14
3.2.5 Proxmark . . . 15
3.3 Existing Microcomputers . . . 15
3.3.1 Arduino . . . 15
3.3.2 Raspberry Pi . . . 15
3.4 Comparison . . . 16
3.4.1 Comparison of Readers . . . 16
3.4.2 Comparison of Microcomputers . . . 16
3.5 Conclusion of the Analysis . . . 17

4 Device Design **18**
4.1 Hardware . . . 18
4.2 Software . . . 19
4.2.1 Operating System . . . 19
4.2.2 Communication With Proxmark . . . 19
4.2.3 Programming Language . . . 20
4.2.4 Program . . . 20

5 Implementation **22**
5.1 Installing the Operating System . . . 22
5.2 Required Packages . . . 22
5.3 Touch Screen Drivers Installation . . . 23
5.4 Proxmark Software Installation . . . 23
5.4.1 Flashing Proxmark 3 Easy . . . 23
5.5 Operating System Customization . . . 23
5.6 Program Implementation . . . 24
5.6.1 Architecture . . . 24
5.6.2 Communication With Proxmark . . . 24
5.6.3 Graphic User Interface . . . 24
5.6.4 User Input . . . 26
5.6.5 Parsing of the Output . . . 27

6 Device Testing **28**
6.1 User Notes . . . 28
6.1.1 Battery Life . . . 28
6.1.2 Boot Time . . . 28
6.1.3 Touch Screen . . . 28
6.1.4 Graphical User Interface . . . 29
6.2 Tested Tags . . . 29
6.3 Tag Search Testing . . . 29
6.4 Tags Cloning Testing . . . 30
6.4.1 Mifare Classic . . . 30
6.4.2 Mifare Ultralight . . . 30
6.4.3 Legic Prime . . . 31
6.5 Tags Emulating Testing . . . 31
6.5.1 Two Types of Emulation . . . 31
6.5.2 EM410x . . . 31
6.5.3 Mifare Classic . . . 32
6.5.4 Mifare Ultralight . . . 33
6.5.5 Mifare Desfire . . . 34
6.5.6 Legic Prime . . . 35
6.6 UID Changing Testing . . . 35
6.6.1 EM410x . . . 35
6.6.2 Mifare Classic . . . 36
6.6.3 Mifare Ultralight . . . 36

7 Related Work **37**

8 Conclusion **38**

Attachments **43**

List of Figures

2.1 A simplified BS-tag communication protocol. 5
2.2 RFID technology, standards and equipment. 6
2.3 The memory organization of EM410x tags. 7
2.4 The memory organization of Mifare Ultralight tags. 8
2.5 The memory organization of Mifare Classic 1k tags. 9
2.6 The memory organization of Mifare DESFire. 10

3.1 Photo of Proxmark 3 Easy 17
3.2 Photo of Raspberry Pi 4 B 17

4.1 Illustration of component wiring. 18
4.2 Preview of Proxmark commandline client. 19
4.3 Information flow between classes. 21
4.4 Wireframe 1. 21
4.5 Wireframe 2. 21
4.6 Wireframe 3. 21

5.1 Simplified class diagram describing the GUI part of the software. 25
5.2 Simplified activity diagram for reading and saving tag information. 26
5.3 Resulting GUI 1. 26
5.4 Resulting GUI 2. 26
5.5 Resulting GUI 3. 26
5.6 Screenshot of the created keyboard. 27

6.1 Output of the second Proxmark when reading the emulated UID of Mifare Classic 4k. 32
6.2 Two Proxmarks on top of each other — emulation testing. 33
6.3 Reading emulated blocks of Mifare Classic 1k. 33
6.4 Wrong ATQA value while emulating Mifare Ultralight. 33
6.5 Emulated Mifare DESFire, read by second Proxmark. 34
6.6 Emulating Legic Prime tag. 35

List of Tables

3.1 Readers comparison 16
3.2 Microcomputers comparison 16

List of code listings

5.1 Updating and downloading the necessary packages. 22
5.2 Installing the touchscreen driver. 23
5.3 Starting Proxmark client to execute given command. 24
5.4 Starting subprocess in Python. 24
5.5 Processing output of a command in Python. 27

I would like to express my highest gratitude to Mr. Jan Fesl for his guidance as my supervisor, to Mr. Tomáš Přeučil for loaning me required hardware and answering my questions, and especially to my family and friends for their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Czech Technical University in Prague has the right to conclude a licence agreement on the utilization of this thesis as a school work pursuant of Section 60 (1) of the Act.

In Prague on May 15, 2024

Abstract

This work focuses on the security of RFID and NFC tags and cards in infrastructure. The aim of the work is to design and implement a portable and extensible open-source device that will be able to clone and emulate some selected types of RFID cards and tags using a selected commercially available microcomputer. The work also involves the development of software with a graphical environment that enables the necessary capabilities of the device. The theoretical part of the thesis explains the basic principles of RFID technology, its security and shortcomings. Furthermore, an analysis of existing solutions that are used for cloning and emulation of RFID technology and the selection of suitable hardware for the creation of the resulting device are discussed. The practical part of the thesis deals with the design and actual implementation of the device from both hardware and software aspects. Finally, the functionality of the device is verified by testing a set of different RFID cards and tags.

Keywords RFID, NFC, cloning, emulation, information security, Proxmark, Python, Raspberry Pi

Abstrakt

Tato práce je zaměřena na problematiku bezpečnosti RFID a NFC tagů a karet v infrastruktuře. Cílem práce je návrh a implementace přenosného a rozšiřitelného open-source zařízení, které bude schopné některé vybrané typy RFID karet a tagů klonovat a emulovat za použití vybraného běžně dostupného mikropočítače. Práce je spojena i s vývojem softwaru s grafickým prostředím, který umožňuje potřebné schopnosti zařízení. V teoretické části práce se vysvětlují základní principy fungování technologie RFID, její zabezpečení a nedostatky. Dále probíhá analýza existujících řešení, které se používají pro klonování a emulaci RFID technologií, a výběr vhodného hardwaru pro vytvoření výsledného zařízení. Praktická část práce se zabývá návrhem a samotnou implementací daného zařízení z hardwarové i softwarové stránky. Nakonec je ověřena funkčnost zařízení testováním množinou rozdílných RFID karet a tagů.

Klíčová slova RFID, NFC, klonování, emulace, bezpečnost informací, Proxmark, Python, Raspberry Pi

List of Abbreviations

BS	Base-Station
CLI	Command Line Interface
GUI	Graphical User Interface
HF	High Frequency
LF	Low Frequency
NFC	Near Field Communication
UID	Unique Identifier
UI	User Interface
RFID	Radio Frequency Identification

Introduction

Nowadays, the importance of RFID and NFC technologies is becoming increasingly crucial. People come across these technologies daily, validating their tickets in public transportation, accessing their offices, and paying with a card in a store. However, their significance extends beyond these everyday applications.

For instance, this innovation revolutionized the way businesses operate and interact with their environments. In retail, RFID tags have brought improvements in commodity management, allowing retailers to have a perfect overview of the stock levels in real time and optimize the supply to maximum efficiency. Logistic management benefits from RFID's ability to track the goods throughout the whole distribution process, reducing delays and eliminating certain management errors. In healthcare, NFC devices help with patient identification and medication administration, improving safety, accuracy, and the speed of healthcare processes [1]. Similarly, the NFC technology is used in public transportation for ticketing, offering commuters a smart and convenient methods of payment and entry.

The history of RFID/NFC technologies can be traced back to the World War II. The air forces of several countries were using radar, which was discovered in 1935 by physicist Sir Robert Alexander Watson-Watt, to warn of approaching airplanes. Germans discovered a method of distinguishing an enemy aircraft from allied aircraft by scanning the difference in the radio signal reflected by the airplane when the airplane rolled. This could be considered as one of the earliest instances of passive RFID system in history. Later on, Sir Watson-Watt came up with the first active system, the so-called "identification friend or foe" system. Brits put an active transmitter on each airplane and when it received signals from the radars on the ground, it started broadcasting a signal back, identifying the airplane as friendly. Over the decades, significant advancements appeared. The first official active rewritable RFID tag was introduced in the 1970s by Mario W. Cardullo. That same year, Charles Walton received a patent for a passive transponder used to unlock doors. U.S. government had also an enormous influence on the development of this technology. Scientists from the Los Alamos National Laboratory developed a system to track nuclear materials in the area. Later, at the request of the Agricultural Department, Los Alamos also developed a system to track cows. Every cow has to be given hormones and medicines when it is ill and it was really hard to keep track of cows so that they do not accidentally receive two doses. [2]

As time went on, RFID and NFC technologies started to be used in more and more industries, as people realized the benefits it provides. These technologies continue to evolve rapidly, driven by innovations such as extended read ranges or enhanced data encryption. The adoption of NFC in smartphones has even further accelerated the integration into everyday life, enabling for example contactless payments or data transfer.

Many security systems heavily rely on RFID/NFC technologies for access control, authentication, or asset protection. The access systems that are commonly found in office buildings,

hotels, or even houses, use this technology to grant or restrict entry based on authorized credentials. Asset tracking solutions use tags to monitor the location or movement of valuable assets, reducing the risk of a theft.

My motivation to pursue this topic was driven by my interest in the subject of RFID access control and identification, which was invoked by popular cloning device solutions such as Flipper Zero and the discussions that arose around these technologies. The creation of the resulting device would help to add to my surface understanding of the subject and the field, which is nowadays increasingly utilized and whose security must be appealed to.

The outcome of this work should help readers to understand the basic principles of the RFID technology, with practical application enabled through the created device. Additionally, this study seeks to uncover security flaws in this particular technology sector, where some of them can be effortlessly exploited by the resulting device.

In this thesis, I undertake a detailed analysis of existing hardware used for RFID/NFC tag reading, cloning, or emulating, which is available on the market and could be used for the resulting open-source cloning device. Later on, I also address the actual design of the architecture of such device, along with the design of the software enabling cloning or emulation. This software includes a user interface allowing simple interaction and use of its features. Furthermore, I provide the implementation process of the software, including the steps taken to ensure hassle-free integration with the used hardware components. Finally, I test the developed device with different types of tags and evaluate its functionality and reliability.

Chapter 1
Goals

The main goal of this thesis is to build a portable open-source device, that will be able to clone and emulate RFID/NFC tags of various types. More precisely, the following sub-tasks are set:

- **Hardware and software architecture design** — design a hardware and software architecture of an open-source device with regard to modularity and future expandability. The device must function as a stand-alone unit without the need to connect to a personal computer. To implement the device, use one of the commonly available microcomputers such as Raspberry Pi, Arduino or similar.
- **Software development** — create software in C++ or Python programming language that allows cloning and emulation of RFID/NFC card/tags of at least 3 different types such as Mifare Classis, Mifare Ultralight, Mifare Desfire v1 etc. Debug the software directly in the environment of the selected microcomputer.
- **User interface implementation** — implement a user interface to control the device that allows the user to easily interact with the device and use its features.
- **Testing** — test the device on different types of RFID/NFC cards/tags and evaluate its functionality.

Chapter 2
Theoretical Background

This chapter deals with the basics of RFID and NFC systems. It explains the principles of communication between the reader and the tag, then introduces the different types of tags and their characteristics, including an explanation of their shortcomings.

2.1 Definition of RFID and NFC

First of all, it is important to explain what RFID and NFC technology actually is.

2.1.1 Definition of RFID

The main source for this subsection is [3], unless specified otherwise. RFID stands for "Radio Frequency Identification". It is a wireless sensor technology, which functions on the principle of detecting electromagnetic signals. A standard RFID system configuration consists of three basic elements:

- a tag, which is a small mobile communication circuit embedded on radiating element,
- a Base-Station (BS), which is a mobile or fixed transmitter or receiver, and
- a database system for the processing of the collected data.

A schematic of a simplified BS-tag communication protocol is depicted in Figure 2.1.
RFID tags can be distinguished by the frequencies in which they operate:

- low frequency (LF), corresponding to 125 kHz and 134.2 kHz,
- high frequency (HF), corresponding to 13.56 MHz,
- ultra-high frequency (UHF), corresponding to 869 MHz, 915 MHz and 950 MHz, and
- microwave, corresponding to 2.45 GHz and 5.8 GHz.

LF and HF tags are widely produced in the world and used in industries such as security, access control or animal identification. Their limitation is their range, which is less than one meter. However, this is compensated by the fact that there are many affordable and reliable solutions at these frequencies enabling many applications. For needs where greater range is essential, UHF and microwave bands are used — the standard reading distance is greater than 3 meters. In the

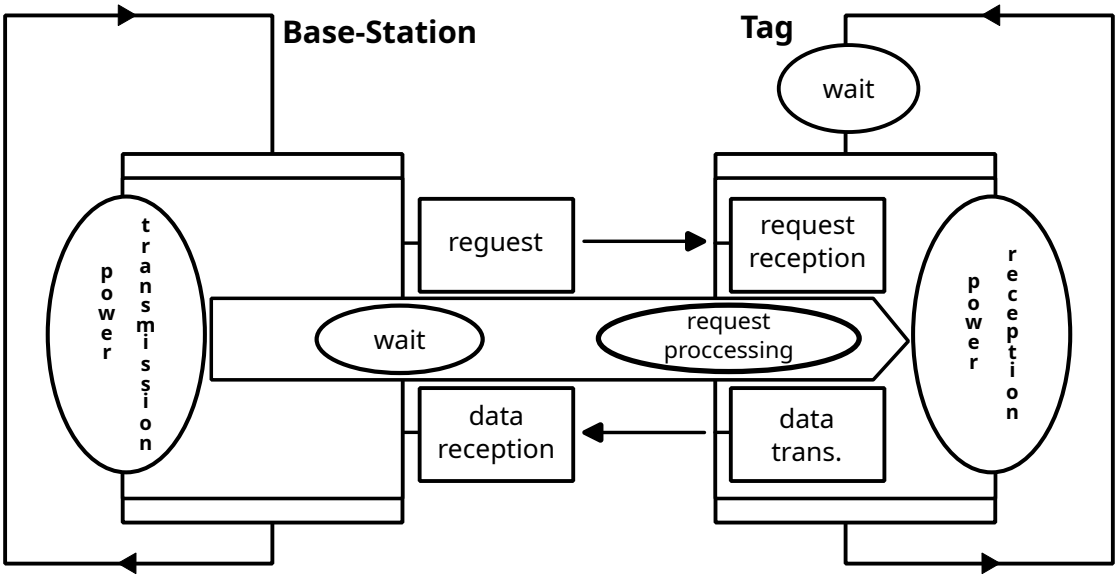

Figure 2.1 A simplified BS-tag communication protocol. Taken from [3], redrawn by the author.

practical part of this thesis we will focus only on LF and HF tags, because these types are used in security and access control systems.

Furthermore, according to their application, RFID tags can be distinguished into active and passive tags. The passive ones are powered by magnetic induction generated by the BS and thus do not need a battery. From now on, in this paper we will restrict ourselves to tags that are passive.

The tag itself is composed of a chip and an antenna, where the antenna has three main functions:

- to power the integrated circuit in the tag. The signal sent by the BS is a fixed frequency electromagnetic wave, generating a voltage in the tag circuit;
- to receive modulated signal from BS. This signal is a request that must be processed by the tag;
- to send the information requested by the BS.

2.1.2 Definition of NFC

NFC could be considered a subset of the RFID technology. The NFC communication range is limited to few centimeters and the NFC operates only on 13.56 MHz frequency. While RFID is only capable of one-way communication, the NFC is capable of two-way communication, meaning that the NFC tag can act as both the reader and the tag. RFID technology has anti-collision algorithms that enable scanning of multiple RFID tags at once, while only one NFC tag can be scanned at a time. [4]

2.2 International Standards

A large number of products and standards fall under RFID and NFC. RFID technology as we know it today began to develop in the 1980s. The International Organization for Standardization (ISO), as well as the International Electrotechnical Commission (IEC), played a fundamental role

in the creation of standards. Today's largest manufacturers of RFID chips and equipment include NXP Semiconductors based in the Netherlands and Alien Technology based in the USA. [5]

The different sectors of RFID technology, their specifications or applications can be found in Figure 2.2.

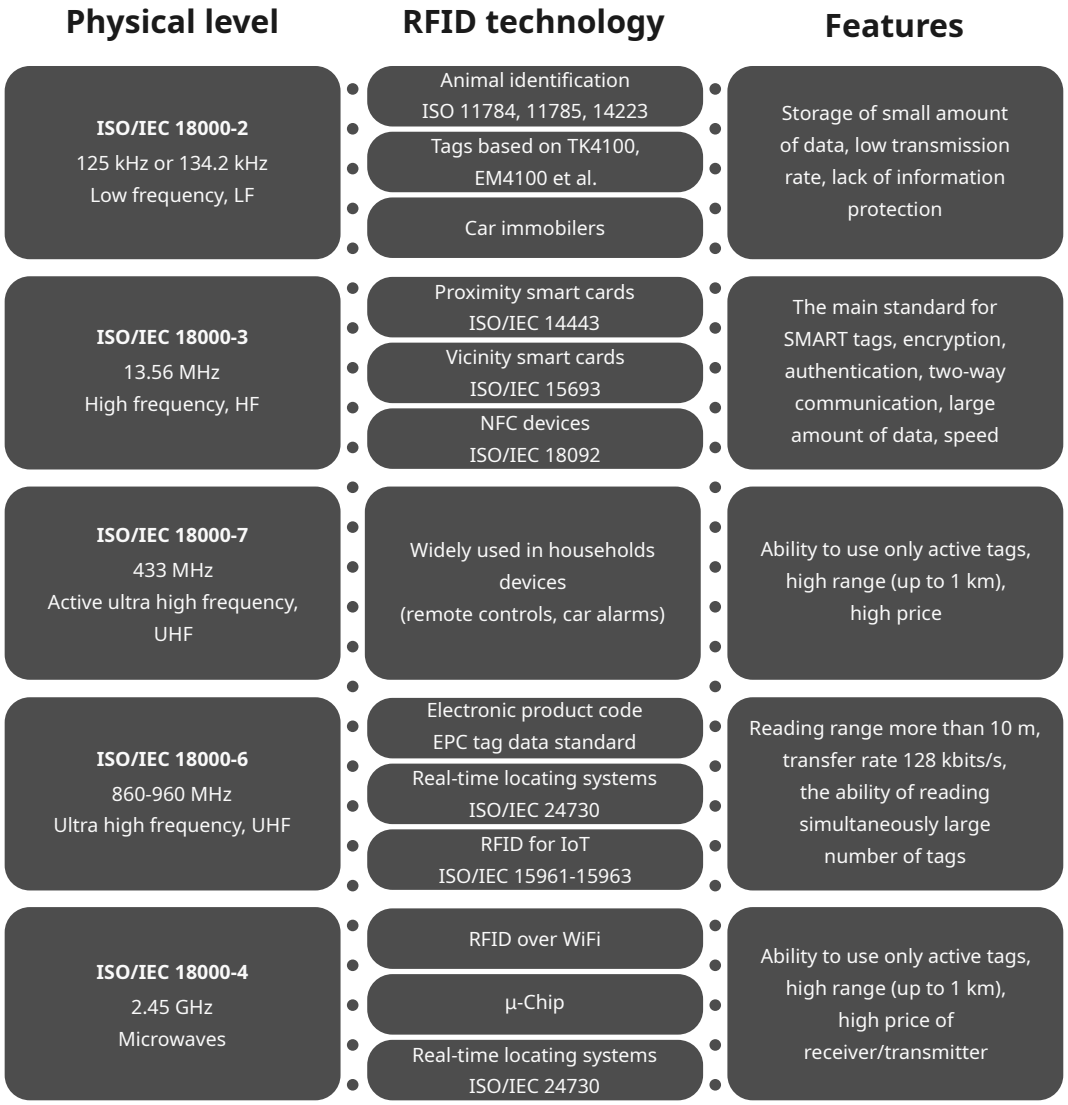

Figure 2.2 RFID technology, standards and equipment. Taken from [5], redrawn by the author.

2.3 Magic Cards

In most cases, tags have some part of their data unwritable, given by the manufacturer. This part of the data may include, for example, the immutable tag UID. In the case of Mifare Classic 1k, the UID has been globally coordinated between manufacturers to ensure that no two cards have the same UID. By the time the security of the Mifare Classic tag had been breached, "compatible chipsets", or also known as "magic chipsets", had started to be produced in China, which were capable of, among other things, forging the UID. These magic tags underwent a short evolution, became more stable and could emulate more types of cards. Later, the so-called

"Ultimate Magic Card", also known as "Gen 4", was released, emulating NTAG, Mifare Classic and Mifare Ultralight tags and all their variants. It also supports complete control over ATQA, SAK, ATS values[1], UIDs, as well as UID length, supporting 4, 7 and 10 byte UIDs. There are plenty of other magic tags on the market that support different chipsets and offer different functionalities. [8]

2.4 Types of Tags and Their Characteristics

This section introduces the specific types of tags that are used in the practical part of the thesis, and discusses their potential vulnerabilities. All of the mentioned tags can be cloned or at least partially emulated.

2.4.1 EM410x Tags

EM410x tags are simple LF tags commonly used in plastic cards. EM410x is a set of individual tag subtypes, such as EM4100, EM4102, EM4105 which are manufactured by EM Microelectronics. [9]

The tag itself contains a contactless transponder carrying only 8 bytes of read-only data and does not implement any encryption, making it quite simple. An unique code is assigned to each individual chip by laser fusing of poly silicon links. [9]

The Figure 2.3 shows the distribution of 64 bits in the tag memory. When the tag gets close to the RFID reader, it induces a voltage inside the tag and starts transmitting its 8 bytes of data, which it repeats as long as it has power [10].

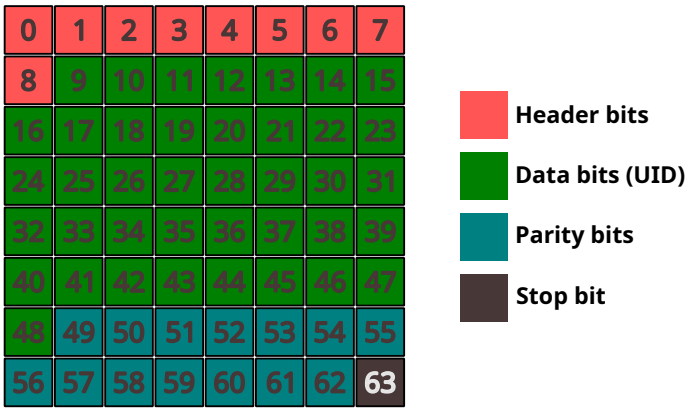

Figure 2.3 The memory organization of EM410x tags. Facts from [9], illustrated by the author.

As these tags do not employ any security features, the cloning process involves only copying the tag's ID and its associated data. [11]

These tags contain different types of chips, e.g., chip EM. This particular chip is not capable of being written to. On the other hand, the T5577 chip is programmable and is capable of emulating various types of LF chips, such as the aforementioned EM chip, HID, or Indala. [12]

[1] ATQA, SAK and ATS are values used to identify the manufacturer, tag type and application of a device. ATQA and SAK are part of the anti-collision phase when setting up a communication channel with a tag. ATS is also known as ATR for contact smartcards. [6, 7]

2.4.2 Mifare Ultralight

The simplest version of Ultralight tags, Mifare Ultralight, is an HF tag developed by NXP Semiconductors for use as a single-use or reusable ticket in transportation, as loyalty cards or day passes for events. The mechanical and electronic features are tailor-made to meet the requirements of paper ticket manufacturers. Mifare Ultralight adheres to the ISO/IEC 14443 A standard. The operating distance is up to 100 mm depending on antenna geometry and reader configuration. [13]

Mifare implements an intelligent anticollision algorithm which enables simultaneous multicard operation. The algorithm individually selects each card based on the tag UID and ensures that every transaction is correctly executed. [13]

The security of the card lies in the following measures:

- 7-byte UID in accordance with ISO/IEC 14443-3 standard,
- 32-bit user definable OTP area (One-Time Programmable area), and
- read-only locking function per page.

The cloning process of the Mifare Ultralight card is trivial, because the tag does not implement any encryption. It is only necessary to copy the contents of the EEPROM memory to another tag, or even change the UID (if the target tag is a magic tag). [13, 14]

The EEPROM 512-bit memory is organised into 16 pages with 4 bytes per page, as illustrated in Figure 2.4. [13]

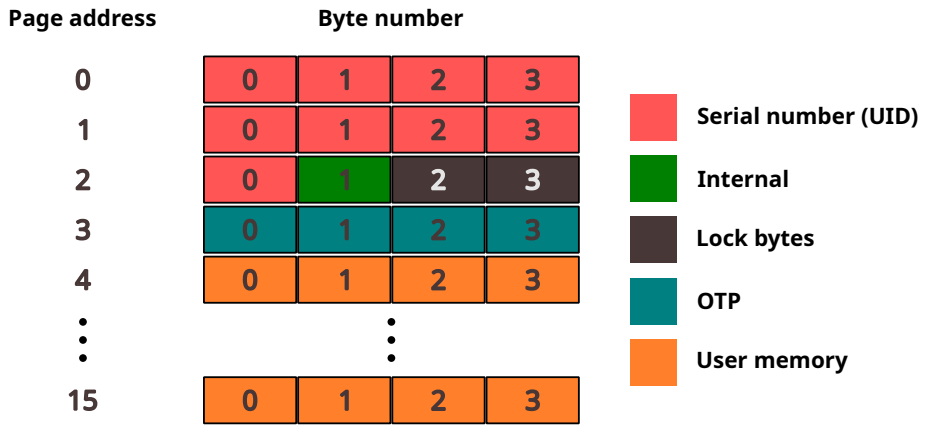

Figure 2.4 The memory organization of Mifare Ultralight tags. Facts from [13], illustrated by the author.

A slightly more complex version of this tag, the Mifare Ultralight EV1, uses password protection. In its configuration, it holds the page number from which all pages are password protected. Therefore, the EV1 does not need to have any secured pages, so it is backwards compatible with the basic version of Mifare Ultralight. Secured Mifare Ultralight EV1 pages can be read after sniffing the trafic between the card and the reader, without sniffing it is possible to read and clone only the non-secure ones. [15, 14]

2.4.3 Mifare Classic

Mifare Classic tags are HF tags also manufactured by NXP Semiconductors, used in public transportation, as school or campus tags, or in car parking. Mifare Classic tags are available in 1k and 4k versions. Like Mifare Ultralight, Mifare Classic tags implement an intelligent anticollision

function. It is also in compliance with ISO/IEC 14443 A standard. The tag memory is divided into data blocks of 16 bytes. The Mifare Classic 1k contains 16 sectors of 4 blocks and the larger Mifare Classic 4k memory contains the first 32 sectors of 4 blocks and the remaining 8 sectors of 16 blocks, making it backwards compatible with 1k version. A schematic of the Mifare Classic 1k memory can be seen in Figure 2.5. [16, 17]

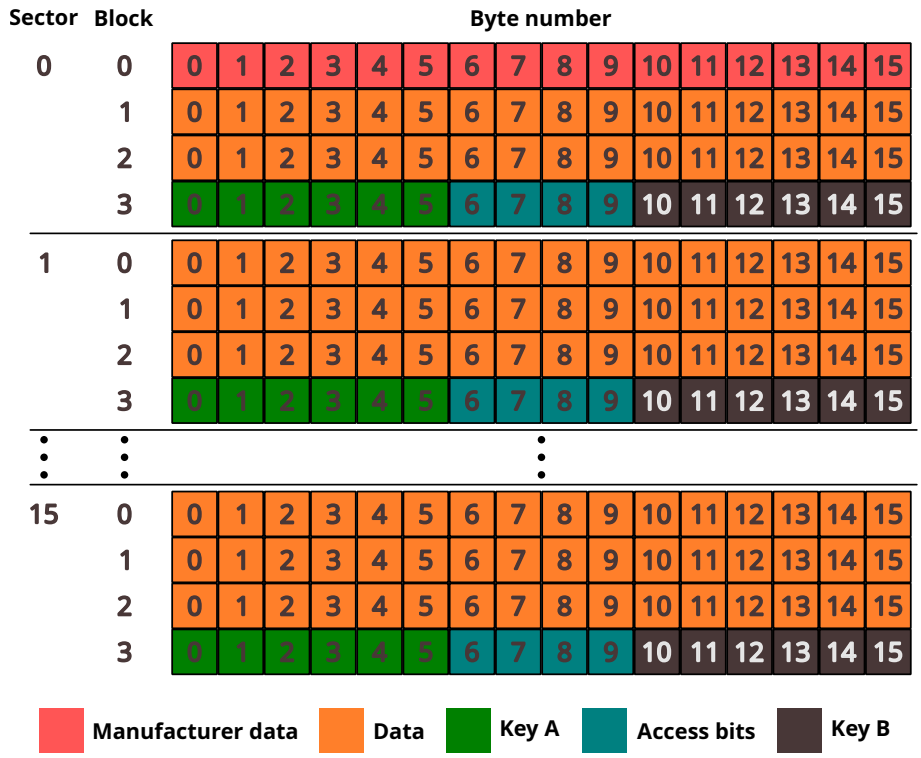

Figure 2.5 The memory organization of Mifare Classic 1k tags. Facts from [16], illustrated by the author.

The first 4 bytes of block 0 of sector zero contain the tag UID. The fifth byte contains the BCC value, which is calculated by successive XORing of all UID bytes. The remaining 11 bytes of sector zero are manufacturer data. [18]

At the end of each sector, there is a block containing keys A and B, which are used for authentication. The access conditions define the operations that can be performed in this sector. This block has special properties. Key A is not readable and Key B can be set to be readable. The reader must authenticate for a sector before any memory operations are allowed. [18]

The communication encrypted using the stream cipher CRYPTO1 was reverse engineered. The authentication proceeds as follows:

1. The tag is selected in the anticollision phase and sends its UID to the reader.
2. The reader asks for authentication of a specific block.
3. The tag sends a challenge nonce n_T.
4. The reader responds with its own encrypted challenge nonce n_R and with the response $a_R = \mathrm{suc}^{64}(n_T)$ where

$$suc(x_0 x_1 \dots x_{31}) := x_1 x_2 \dots x_{31} L_{16}(x_{16} x_{17} \dots x_{31})$$

and

$$L_{16}(x_0x_1 \dots x_{15}) := x_0 \oplus x_2 \oplus x_3 \oplus x_5$$

.

5. Authentication is concluded by the tag response $a_T = \text{suc}^{96}(n_R)$. [19]

There are several known vulnerabilities in Mifare Classic, such as short key lengths, predictable nonces or nested authentication. You can read more about them in [20].

2.4.4 Mifare DESFire

Mifare DESFire tags are HF tags considered one of the most advanced RFID technologies. Its EV1, EV2 and EV3 generations are considered secure and no attack has been described yet. It offers 3DES or AES for encrypting transmission data. It can be used in building access, transport, closed-loop e-payment schemes and others. It is compliant to all 4 levels of ISO/IEC 14443 A. Tags use 7 byte UIDs. [21, 14]

To give a brief example of the memory organisation, the 2/4/8 kB memory of the Mifare DESFire EV1 tag is organised using a flexible filesystem, which allows 28 different user application on a single tag, where each application provides up to 32 files. Every of the application is represented by its 3 bytes application identifier (AID). The memory organisation is depicted in Figure 2.6. Each file can be created at tag initialization or personalization. The tag can be controlled via tag, application, data manipulation, and ISO/IEC 7816 APDU level commands. [21, 14]

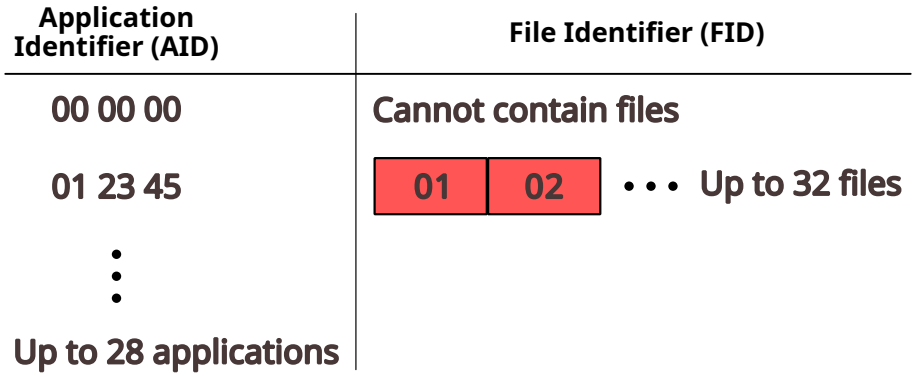

Figure 2.6 The memory organization of Mifare DESFire. Inspired by [22], illustrated by the author.

Prior to data transmission a mutual three-pass can be done employing either 56-bit DES (single DES, DES), 112-bit 3DES (triple DES, 2K3DES), 168-bit 3DES (3 key triple DES, 3K3DES) or AES. During the authentication, the security of all following commands is set. [21]

2.4.5 Legic Prime

Legic Prime is a HF tag developed by Legic. It was launched in 1992 and used in access control, but has also found use in payment applications. It can hold several applications, however, this is rarely ever seen. There are 3 variants:

- MIM22, which is outdated,
- MIM256, which has a memory of 234 bytes, and

- MIM1024, with a memory of 1002 bytes.

Legic takes obscurity to the extreme, there is no official documentation beyond layer 1+2, and compared to Mifare Classic it is much more challenging to get tags and readers on the free market. [23]

It uses its proprietary radio protocol and its own ”legic encryption”. The security system breaks in two places, where cryptography is missing:

1. both cards and readers do not hold secret keys and cannot authenticate each other, allowing an attacker to assume either role and read, write or spoof cards and readers, and
2. the lack of cryptography allows attacks against Legic's trust delegation model, which places cards in a hierarchy where higher level cards can produce lower level cards. Since there is no secret information to distinguish higher from lower levels of authorisation, any level can be spoofed for any Legic installation in the world.

For these reasons, the Legic Prime system is not suitable for access control systems in the long term. [24]

Chapter 3
Analysis

This chapter presents the analysis of the commercially available hardware. It outlines the capabilities of existing RFID/NFC readers, cloners, emulators, microcomputers, and other relevant hardware components. Specifically focusing on readers, it compares the devices based on various parameters, including compatibility with certain tag types, limitations, and pricing. Additionally, other hardware components are evaluated based on their capabilities and compatibility with RFID readers. The objective is to determine the most suitable device for integration into the portable device developed in this thesis, effectively meeting the specified requirements.

3.1 Requirements

Based on the goals specified in Chapter 1, this section sets the requirements for the components of the resulting device. It describes the desired characteristics of the RFID/NFC reader, microcomputer, and its components.

3.1.1 RFID/NFC Reader Requirements

As written in Chapter 2, there are types of tags that are easy to clone or emulate, as their main aim is to provide availability rather than integrity or confidentiality of data. In other words, some types of tags do not prioritize providing security measures. For instance, a simple 125 kHz EM410x tag holds data that can be easily read without any constraints and can be cloned or emulated quite effortlessly. Certain tags are designed to provide users with security but have failed in secure implementation. For example, their encryption algorithm has been reverse-engineered, and a variety of attacks have been published, effectively compromising their security[1]. These tags are a little more complicated to clone, as some type of attack must be executed against the tag, exploiting its vulnerabilities. Lastly, there are tags for which no vulnerabilities or published attacks are known.

It would be beneficial to utilize an open-source reader capable of reading, writing, emulating tags, and exploiting the known vulnerabilities of some of the tags all by itself. This would help to narrow the scope of the thesis. Also, the reader must be operable from the external microcomputer.

[1] A perfect example of such tags are those using the encryption algorithm Crypto-1

3.1.2 Microcomputer Requirements

There are various requirements for the microcomputer. One of the most crucial aspects when searching for a suitable microcomputer is its ability to communicate with the chosen reader effectively. The communication should be implementable using the chosen programming language, with minimal limitations. Another critical parameter is the ability to control the microcomputer. Importantly, there should be a way to connect a small screen to the device. The device should be controllable via a touchscreen or buttons. The power consumption of the microcomputer must also be taken into account to ensure that the resulting portable device is capable of lasting a satisfying period of time.

3.1.3 User Interface Requirements

The user interface (UI) of the device should meet several properties. One of the required properties of the UI may be considered the simplicity — the device should be easy to use without any extra thinking. Another important property is the ability to enter characters. However, the usage of physical buttons would not be a good solution, because there would have to be too many buttons and the entering of the characters would be too slow. This determines the direction to use a touch display instead. This way, the UI would be simple and user-friendly, without many limitations.

The power consumption of the display is also an important evaluation criterion, and so is the size of the display, which should not be too big — the device should be as compact as possible.

3.2 Existing RFID/NFC Cloners

This section covers various existing solutions and devices capable of cloning or emulating tags. It provides a general overview of their capabilities, prices, and potential limitations.

3.2.1 Flipper Zero

The Flipper Zero is one of the most well-known gadgets. Among its capabilities, such as emulating a BadUSB device[2] or controlling devices that utilize radio signals below 1 GHz, it can also operate at frequencies of 125 kHz and 13.56 MHz for RFID tags [26]. It offers a highly portable solution for reading, writing, cloning, and emulating tags, supporting various types, such as MIFARE Classic 1k or MIFARE Ultralight [27]. Its software is licensed under GPL-3.0, making it fully modifiable and distributable under the same license [28].

The device is powered by an integrated lithium-ion polymer rechargeable battery, with power management efficiently handled to provide operation for up to a month without recharging. [29]

This device, however, is already completely self-sufficient and functional, and therefore its use is not considered. On the contrary, it can serve as an inspiration for this work. The device is available for purchase at the official website for 165 € [30]. Although this price may seem high, it is essential to consider that the device offers significantly more functionalities than those outlined in Section 3.1.

3.2.2 Chameleon Mini

The Chameleon Mini is a versatile credit-card shaped RFID and NFC tool, often used for security compliance, analysis, penetration testing, and various applications. Its fully open-source platform

[2] A so-called BadUSB device is a device that is recognized by computers as a human interface device, such as a keyboard [25].

supports emulating perfect clones of various commercial smartcards, including the UID and even cryptographic functions. Its human-readable command set provides users the ability to configure settings and content for up to 8 internally stored virtual cards. Integrated buttons and LEDs enable user interaction while in standalone mode, when the device is not connected to the computer. [31]

Despite its capability to read and emulate various types of high frequency cards, its firmware does not provide the ability to write cards. To crack cards, the device itself supports only a MFKey32 attack, where the attacker has to have also an access to the reader itself to catch and decode transmitted keys. This does not quite meet our requirements. [32]

The Chameleon Mini's battery life is quite extensive, it contains a Li-ion battery which is capable of running the device for up to 1 year with an average usage of 15 seconds a day. The device is quite expensive, it can be purchased for around 130 €. [32]

This powerful and highly modifiable device supports the emulation of cards of various types, however, its software cannot write cards.

3.2.3 Chameleon Ultra

Chameleon Ultra is a relative of Chameleon Mini, one of the smallest solutions in existence. It is about the size of a keychain, capable of reading, emulating, writing, and even cracking tags. It boasts better emulation capabilities than the two other related devices — Flipper and Proxmark. [33]

Chameleon Ultra can be controlled via command line interface (CLI) and graphical user interface (GUI), but can also be controlled via a mobile app. Includes a 90 mAh LiPo battery that lasts up to a month without a charge. [33]

Chameleon Ultra is very capable for its size, meeting all our requirements. However, this is reflected in its price of 155 € [33], which needs to be considered when making the final decision.

3.2.4 NXP PN532 Module

The NXP PN532 is a highly integrated module operating at 13.56 MHz frequencies. It supports 6 different operating modes,

- ISO/IEC 14443A/MIFARE Reader/Writer,
- FeliCa Reader/Writer,
- ISO/IEC 14443B Reader/Writer,
- ISO/IEC 14443A/MIFARE Card MIFARE Classic 1K or MIFARE Classic 4K card emulation mode,
- FeliCa Card emulation, and
- ISO/IEC 18092, ECMA 340 Peer-to-Peer. [34]

With various libraries, it is possible to perform for example an offline nested attack [35] or a darkside attack [36] in the computer. However, these attacks are not done in the module itself, but in the connected computer.

Another library makes possible to write Gen 3 Magic MIFARE Classic tags and Gen 2 Magic MIFARE Classic tags. [37]

It can be classified as an affordable device, one can buy this device for only around 10 €. [38]

This device can be controlled via Arduino microcontroller, however, its downside is the restriction to 13.56 MHz frequencies.

3.2.5 Proxmark

The Proxmark is an RFID tool, supporting vast majority of RFID tags worldwide. It allows both low level and high level interactions with the tags. It is adapted to many industries and users, used by people from the academic research, product development or penetration testing. [39]

Proxmark's functionalities include the ability to read tags, write tags, write magic tags, simulate tags, and even crack the tags, meaning the Proxmark software includes attacks against various known tags vulnerabilities. [40]

There are a number of variants of the Proxmark on the market, including cheap clones, which however, can be flashed with the software and are supported. [41]

The device can be connected to a computer with USB cable and it is controlled via a command-line interface [42]. It also supports a standalone mode, in which the device can be utilized without connection to the computer. However, the standalone mode is quite limiting, because the device itself contains only one simple button and LED indicators — this does not allow more complicated user input. [43]

The price of one of the cheaper variants of the Proxmark, the Proxmark 3 Easy, is around 75 € [44].

3.3 Existing Microcomputers

This section covers various existing microcomputers. Provides a general overview of the capabilities, and options for connecting other modules, such as displays or the reader itself.

3.3.1 Arduino

Arduino is an open-source platform featuring easy to use hardware boards, containing a microcontroller, and input/output pins for connecting all kinds of peripherals. It is programmable by *Arduino programming language*, which is based on *Wiring*[3]. [46]

Many types of Arduino boards are available, varying in size, number of pins or computing power [47]. Moreover, Arduino can be enhanced with many accessories, including various types of screens, buttons, or even RFID/NFC readers — as an example could be used the aforementioned PN532 module.

It is still safe to say that the price of the boards is quite low, even more when Arduino clones[4] are taken into consideration.

3.3.2 Raspberry Pi

Raspberry Pi produces a series of single-board modular microcomputers that are built on the Arm architecture and also Raspberry Pi Pico boards for lower-power and real-time applications. The series of microcomputers run the Linux operating system, which makes them exceptionally potent for a diverse number of tasks. Every Raspberry Pi board is backed by extensive technical documentation, and enthusiastic community. [48]

Each board has several input/output pins, number of models contain also RJ45 network ports, USB ports, or even HDMI ports.

Similar to the Arduino, the Raspberry Pi has many modules available, including screens, for which there are open-source drivers and libraries available.

The price of a Raspberry Pi 4 model B, containing all the ports mentioned previously, goes from 35 € [49].

[3] *Wiring* is an open-source programming framework for microcontrollers [45].

[4] These clones are cheaper unofficial versions of the original Arduino boards.

3.4 Comparison

For clarity, following tables give an overview of the properties of all components. This will help to select the most suitable ones.

3.4.1 Comparison of Readers

Table 3.1 Comparison of tag readers

Device	LF	HF	Read	Crack	Write magic	Emulate	Price
Flipper Zero	✔	✔	✔	✔	✔	✔	High
ChameleonMini		✔	✔			✔	High
ChameleonUltra	✔	✔	✔	✔	✔	✔	High
PN532 module		✔	✔		✔	✔	Low
Proxmark	✔	✔	✔	✔	✔	✔	Middle

The Table 3.1 shows a comparison of the mentioned RFID/NFC tag readers. Only Proxmark and Chameleon Ultra meet the specified requirements, as the other readers mentioned lack important features, such as low-frequency card support for the NXP PN532 module and Chameleon Mini. The Flipper Zero does not meet our requirements as it is already a standalone unit, therefore, it could not be used as a component in the final device.

The Chameleon Ultra ticks all the boxes, but is very expensive for the purposes of this thesis. In contrast, the price of Proxmark is satisfactory because it is far from the highest and still meets all the conditions.

The big advantage of Proxmark is a great support — the Github repository containing the device firmware and also the desktop client is regularly updated. A disadvantage may be the size of the device, the Proxmark 3 Easy type has dimensions of 54 mm × 86.6 mm [50]. However, this might not matter so much, it will also depend on the size of the microcomputer selected.

Since the Proxmark is the most capable of the devices analyzed and the price is appropriate, this device is chosen as the reader used for this project.

3.4.2 Comparison of Microcomputers

Table 3.2 Comparison of microcomputers

Device	Easy communication with Proxmark	Energy consumption	Touch display available	Capabilities	Price
Arduino		Low [51]	✔	Low	Low
Raspberry Pi	✔	High [52]	✔	High	Low

The Table 3.2 shows a comparison of the mentioned microcomputers. Both options are strong candidates, but implementing Arduino communication with Proxmark would be much more challenging than providing Raspberry Pi communication via USB port.

In terms of extensibility, again, both are very good, but the Raspberry Pi is slightly better. With the computational power of the Raspberry Pi, other types of attacks could be implemented in the future, with the computations taking place within the microcomputer itself. The Raspberry itself has Bluetooth and Wi-Fi support built in by default, and in the future it would be much easier to control the device via, for example, a mobile phone than it would be with an Arduino.

Likewise, implementing a software user interface will be much easier on the Raspberry Pi, as it runs Linux and thus has many graphical programming language libraries to choose from.

The disadvantages of the Raspberry Pi may be its size and power consumption, which is high compared to the Arduino.

3.5 Conclusion of the Analysis

Several RFID/NFC reader solutions were analysed, and it was found that many of them could function as standalone units. After careful consideration of the requirements, Proxmark was selected as the most suitable reader for our project. Additionally, the Raspberry Pi was identified as the most appropriate microcomputer for this work.

After analyzing the different types of Proxmark, the Proxmark 3 Easy was selected. It is one of the cheapest variants of Proxmark and at the same time, it is not deprived of any important functionality.

After analyzing the different types of Raspberry Pi, the Raspberry Pi 4 B variant with 4 GB of RAM was selected. This variant is one of the newer and more powerful ones. However, it is also quite large, but this does not matter because the selected Proxmark 3 Easy is almost the same size and therefore choosing a smaller microcomputer would not reduce the overall size of the resulting device.

The Waveshare 3.5” LCD touchscreen is suitable as a display for the device, as it is the same size as the Raspberry Pi and Proxmark. It is connected and powered via 26 pins. There are freely available drivers to make it easily usable. [53]

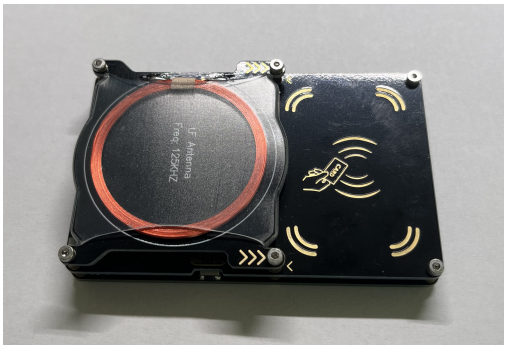

Figure 3.1 Photo of Proxmark 3 Easy

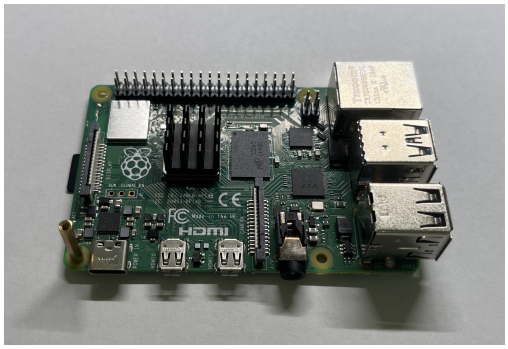

Figure 3.2 Photo of Raspberry Pi 4 B

Chapter 4
Device Design

In this chapter, the resulting cloning device, its hardware and software will be designed. The selection of components is based on the Chapter 3.

4.1 Hardware

After a thorough selection of hardware components, the best combination of the Proxmark 3 Easy reader with a Raspberry Pi 4 B microcomputer was found. These two devices will be connected by a USB-to-micro USB cable, where the male ends can be connected with the shortest possible connection cable to save space in the final device. This cable will provide all communication between the microcomputer and the reader, and will also power the Proxmark 3 Easy via this cable. On the Raspberry Pi itself, a Waveshare 3.5" LCD touchscreen of the same size as the microcomputer will be connected via pins. The image will be transmitted via the pins and the screen will be powered by this route. An illustration of the component wiring can be seen in the Figure 4.1.

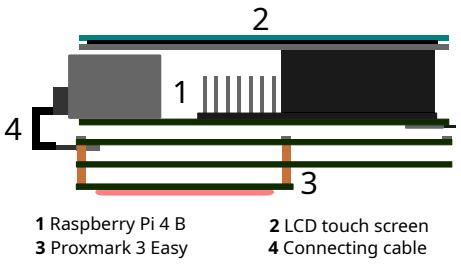

Figure 4.1 Illustration of component wiring.

To ensure greater battery life and the renewability of the power source, it is best to power the device using a power bank, which will be connected externally to the device using a classic USB-C cable. It is recommended that the power bank supplies the device with 5V/3A [52].

This design makes the device compact and easy to use. On the top touch screen, the user controls the device and attaches cloned tags from the bottom of the device. It should be noted that the Proxmark 3 Easy has the LF and HF antenna in different locations, the user must then attach the tag to the specific antenna as needed.

The Proxmark has a button on its longer side that is used to control some of its functions, including termination of tag emulation. It is therefore necessary to make this button accessible to the user.

4.2 Software

This section will justify the selection of the software components of the project, like operating system, programming language or used programming libraries. In addition, the software architecture and GUI design will be discussed here.

4.2.1 Operating System

With the selection of the Raspberry Pi microcomputer, it became clear that Linux would be used as the operating system of the device. It is possible to choose a specific Linux distribution, but for this purpose the official Raspberry Pi OS, which can be obtained for free from the Raspberry Pi [54] website, will suffice.

The instructions for installing the LCD touch screen drivers recommend the Linux version of bullseye, 32 bit. For the purposes of this device, this version of Linux is suitable. [53]

The operating system will run in the background of the device, the user will not be able to manipulate the operating system in any way — and therefore, when Linux starts, the program will automatically run and take up the entire screen. When the program is finished, the computer will shut down. A slight but tolerable drawback is the startup time.

Since the device will be controlled by a touch screen, the operating system will have to be adapted to this, including adjustments like an invisible mouse cursor.

4.2.2 Communication With Proxmark

Proxmark 3 Easy is controlled via the commandline client, see Figure 4.2. Single commands are sent to the reader to perform operations, such as reading or writing tags. It is also possible to start the client to perform only one specific operation. An output is sent back from Proxmark, which contains various information, such as the success of the operations, or an enumeration of the tag contents. The created program will therefore communicate with Proxmark by starting a process sending a specific operation to Proxmark for execution, then parsing the output and passing the necessary feedback to the user. It will also need to ensure that the program checks the availability of Proxmark to avoid errors.

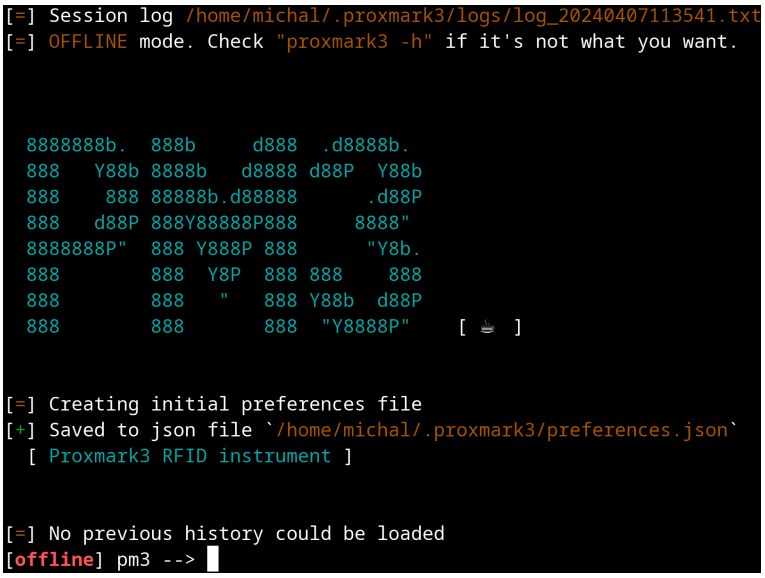

Figure 4.2 Preview of Proxmark commandline client.

Some Proxmark operations, such as emulation of some tag types, cannot be terminated by user input in the client, but by pressing a physical button on the Proxmark itself. For this reason, the button will need to be accessible.

4.2.3 Programming Language

The Subsection 4.2.2 shows that the program with the graphical environment itself will not do any complex calculations, it will only send commands to Proxmark and parse the output. For this reason, the Python programming language is an excellent choice. In Python, several graphical libraries are available and parsing the output is very straightforward.

4.2.3.1 Python Graphical User Interface Library

There are no big demands on the GUI library, from many suitable ones such as PyQt5 [55], Kivy [56], or TKinter [57] the PyQt5 library was chosen, which coincidentally is already part of the Raspberry Pi Os Bullseye 32 bit operating system. This library provides everything needed to create a GUI program.

4.2.4 Program

The program itself must therefore give the user the ability to easily control Proxmark. It must allow the user the following functions:

- display basic information about the attached tag,
- dumping and storing data of different tag types,
- cloning saved dumps into new tags,
- manually edit the UIDs of different tag types,
- emulate the content of different tag types, and
- emulate UIDs of different tag types.

When it comes to displaying various information about the attached tag, the Proxmark output itself is already clear and needs almost no editing.

Storing tag dump data will force the existence of a repository in Proxmark where the dumps will be stored. The program will then need to include an interface for browsing of these files.

Manual editing of UIDs will require some sort of on-screen keyboard where the user can enter input. There is a solution of a pop-up keyboard, but a more elegant solution is to create a custom keyboard where only the keys that are needed will be present. The UID is simply read in hexadecimal format, so it will be sufficient if the keyboard contains only the characters 0–9 and A–F.

4.2.4.1 Architecture

The PyQt5 library offers a relatively simple way to manage individual screens. A class named `QStackedWidget` will allow holding individual widgets[1]. Inside the widget, the screen layout can be arranged in a simple way by aligning or applying individual styles. Each screen, i.e. widget, will hold its sub-screens (submenus). This hierarchy allows for easy navigation through the code.

Next, a class called `CommandExecutor` needs to be created to take care of executing individual operations on the Proxmark.

[1] Widget is the class representing the screen for us at the moment

A class called `CommandBuilder` will prepare individual commands for the `CommandExecutor` class to execute, depending on the user's input.

Finally, a class processing the output is necessary. It will receive the output from the Proxmark, process it, and forward the signals back to the user.

Another class named `FileHandler` will facilitate working with files. It will be useful to create this class that will take care of this work. For easier debugging, a logging system will be useful, storing all logs for later evaluation.

The process of executing a command — how the classes will pass information to each other can be seen in Figure 4.3.

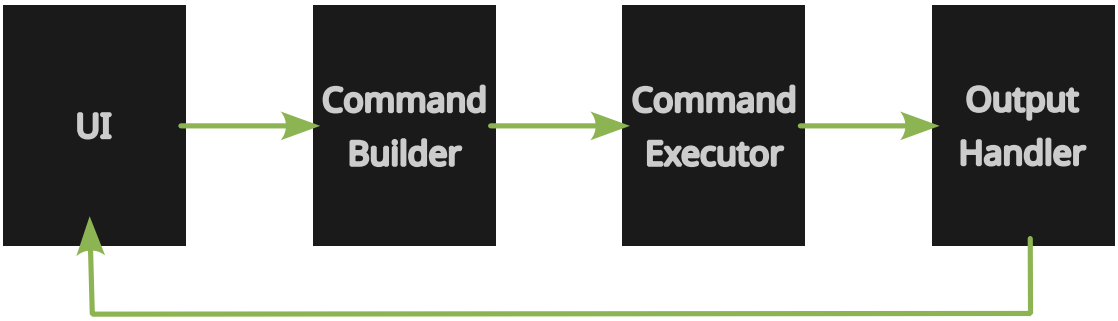

Figure 4.3 Information flow between classes.

4.2.4.2 GUI Design

First of all, it is important to design the visual features of the GUI. Since it will run on a small display with a resolution of only 480 × 320, the GUI must be well laid out and easy to use.

To keep it simple, the best solution will be to create menus and submenus that the user will navigate through back and forth. This will ensure simplicity of operation on the touchscreen and allow for a relatively small number of buttons in a single menu, efficiently using the space of the small screen. In some situations, however, there may be a problem that more content needs to be displayed than can fit on one screen, for example when displaying a list of saved tags. This can be resolved by adding a scrolling area to the GUI.

To get an idea of how such an interface would look like, we can refer to Figure 4.4, Figure 4.5 and Figure 4.6 containing black and white wireframes.

Figure 4.4 Wireframe 1.

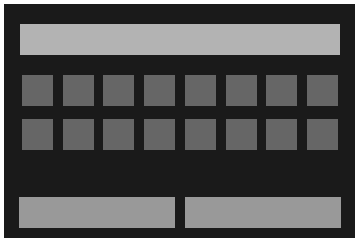

Figure 4.5 Wireframe 2.

Figure 4.6 Wireframe 3.

Chapter 5
Implementation

This chapter will describe the process of implementing the device — the commissioning and modification of the computer operating system, the installation of the necessary software and the actual implementation of the program. Any changes to the design of the device will be discussed.

5.1 Installing the Operating System

As mentioned in the design chapter, the appropriate operating system for the Raspberry Pi is Raspberry Pi OS Bullseye 32 bit, partly because of its compatibility with the touchscreen used. The installation is straightforward, using the Raspberry Pi Imager to install the selected system version on the SD card. A Kingston 32 GB SD card was used, although 8 GB would also be sufficient. To get the device working, it is also necessary to provide an internet connection to download the necessary packages, so the device was connected to WiFi. Also, SSH[1] needs to be set up so that the device can be configured remotely when the touchscreen is not yet in use. It should be noted that the system user name must be left `pi`, otherwise the touchscreen driver installation script does not work correctly. The error is caused by using an absolute path with the user name `pi` in the installation script, and therefore with a user of a different name the script does not work as it should.

After successfully uploading the operating system to the SD card, the card was inserted into the device to test its functionality by booting and connecting from a remote computer on the same network.

5.2 Required Packages

Once the operating system has been successfully commissioned, it is necessary to download the necessary packages to install both the screen driver and the Proxmark software. The following script in Code listing 5.1 updates the system and downloads necessary packages.

```
#! /bin/bash

sudo apt update
sudo apt upgrade
sudo apt install libreadline-dev gcc-arm-none-eabi libssl-dev cmake
   liblz4-dev libbz2-dev
```

[1] Secure shell.

Code listing 5.1 Updating and downloading the necessary packages.

5.3 Touch Screen Drivers Installation

After installing the necessary packages, it is necessary to install the driver for the Waveshare LCD 35B-V2 touch screen. This has been performed according to the tutorial in [53]. Additionally, in Code listing 5.2, the script for installing that driver is provided.

```
#! /bin/bash

cd ~
git clone https://github.com/waveshare/LCD-show.git
cd LCD-show/
chmod +x LCD35B-show-V2
./LCD35B-show-V2 # For this specific display, this  LCD35B-show-V2
   script must be used
```

Code listing 5.2 Installing the touchscreen driver.

5.4 Proxmark Software Installation

The Iceman Fork is a very well maintained repository with software for Proxmark devices. This repository on Github [58] will be used for Proxmark software installation. It contains a client, but also scripts to flash the Proxmark itself.

Using the installation procedure given in [59], the software can be installed without any problem. It should be mentioned that before starting the compilation, it is necessary to change the platform in the `Makefile.platform` configuration file to `PM3GENERIC`, for which the program will be compiled. It is also necessary to pay attention to the ModemManager, which could damage the device.

5.4.1 Flashing Proxmark 3 Easy

After successful compilation and installation, you can also follow the instructions on Github [59]. Proxmark 3 Easy can of course also be flashed from a computer other than a Raspberry Pi.

5.5 Operating System Customization

The following steps are provided to improve the appearance of the software itself, and to create some limitations for users, to have their use of the device limited to the single application:

- removing the mouse cursor from the screen,
- autostarting the program when the operating system starts,
- disabling all unused features such as WiFi or Bluetooth (not done at first for testing purposes).

To remove the cursor, the `/etc/lightdm/lightdm.conf` file needs to be modified, changing `xserver-command` to `xserver-command = X -nocursor`. As a result, the cursor is no lonoger visible but the touch screen control works as it should.

Executing the program automatically upon start is done by putting a desktop file in the `~/.config/autostart` folder.

5.6 Program Implementation

This section will explain the principle and description of implementation of the GUI program that controls Proxmark 3 Easy.

5.6.1 Architecture

The implementation followed the design outlined in Chapter 4. This means that the `PyQt5` library allowed for a great layout of individual screens. Thus, each class representing a screen also has a `QStackWidget` class variable that holds the individual sub-screens.

The program's basic architecture of is depicted in a simplified class diagram in Figure 5.1. The main class, `MainWindow`, represents the program window. Then, there are different screens like `MainScreen`, which contain more screens or other parts of the interface. The implementation also made a great use of class inheritance, where, for example, all screens inherit from one abstract class called `BaseScreen`. Class inheritance was also used in other cases such as buttons or notification toasts[2]. This made it easy to create a simple menu in the program. The command execution was implemented as planned in Chapter 4.

5.6.2 Communication With Proxmark

The program communicates with Proxmark as follows. The Proxmark client has an option to start itself to execute only single command, which when terminated, will terminate the process itself. An example of running the client this way is shown in Code listing 5.3.

```
pm3 -c 'lf search'
```

Code listing 5.3 Starting Proxmark client to execute given command.

The `CommandExecutor` class receives individual commands from the `CommandBuilder` class. These are then started in the `CommandExecutor` class as a new process, see Code listing 5.4. "*The subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes.*" [60]

```
process = subprocess.run(command, shell=True, capture_output=True,
    text=True)
```

Code listing 5.4 Starting subprocess in Python.

Then the output of the executed command is passed to the `OutputHandler` class, which extracts important information from the output depending on the command, e.g. the success rate of execution or some other information like the UID of the tag. This information is then passed to the UI itself. Before each command is executed, the connection of Proxmark 3 Easy to Raspberry Pi is checked. This is done to determine in case of failure exactly why the command was not executed. A simple activity diagram depicting the process of reading and saving a tag information can be seen in Figure 5.2.

5.6.3 Graphic User Interface

The creation of the GUI followed the design presented in the Chapter 4. The resulting design is therefore very simple and intuitive. An example of it can be seen in Figure 5.3, Figure 5.4 and Figure 5.5.

[2] A notification toast is a brief, non-interactive alert message that appears temporarily on a mobile device or computer screen, typically in a corner or at the top of the screen.

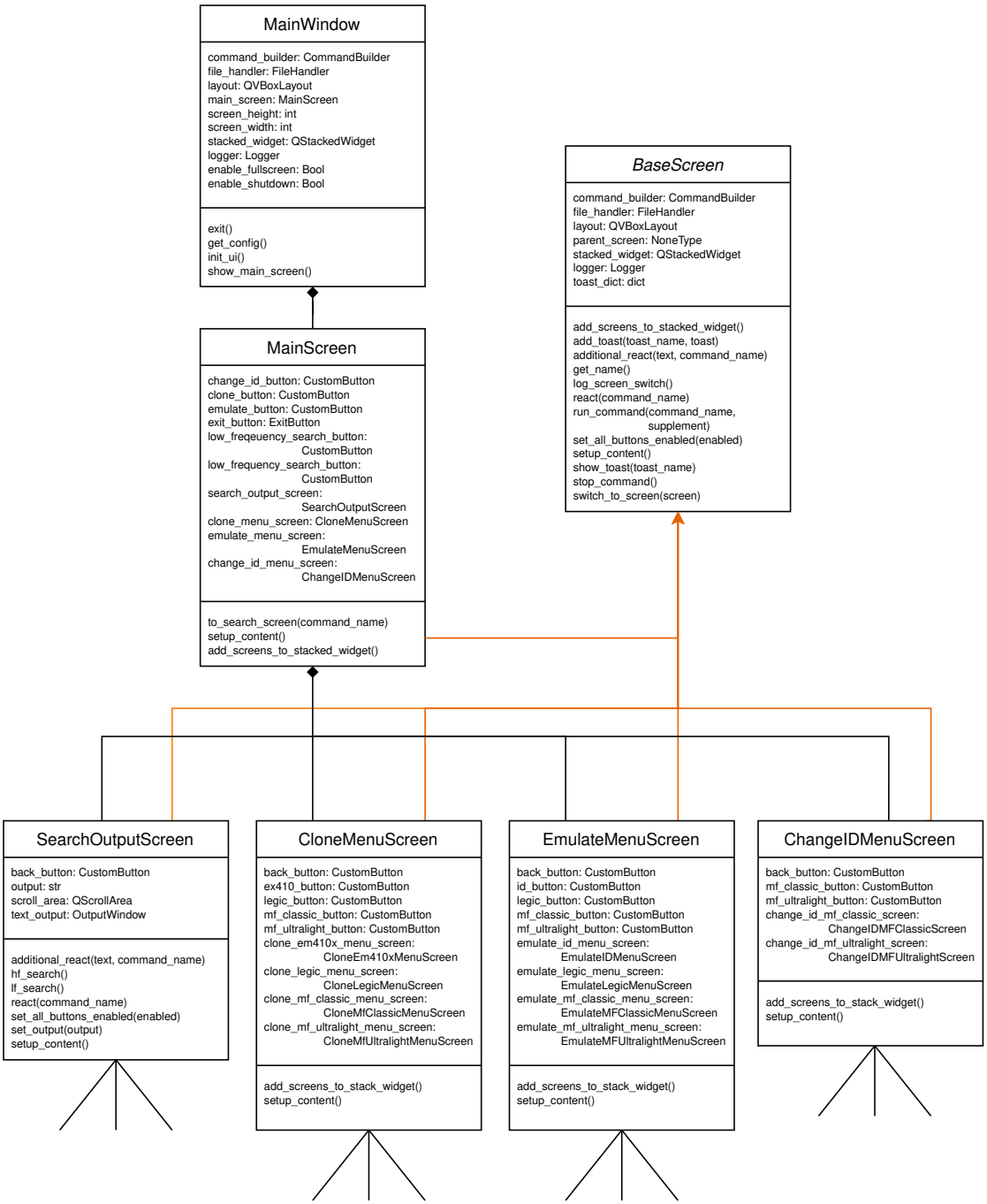

Figure 5.1 Simplified class diagram describing the GUI part of the software.

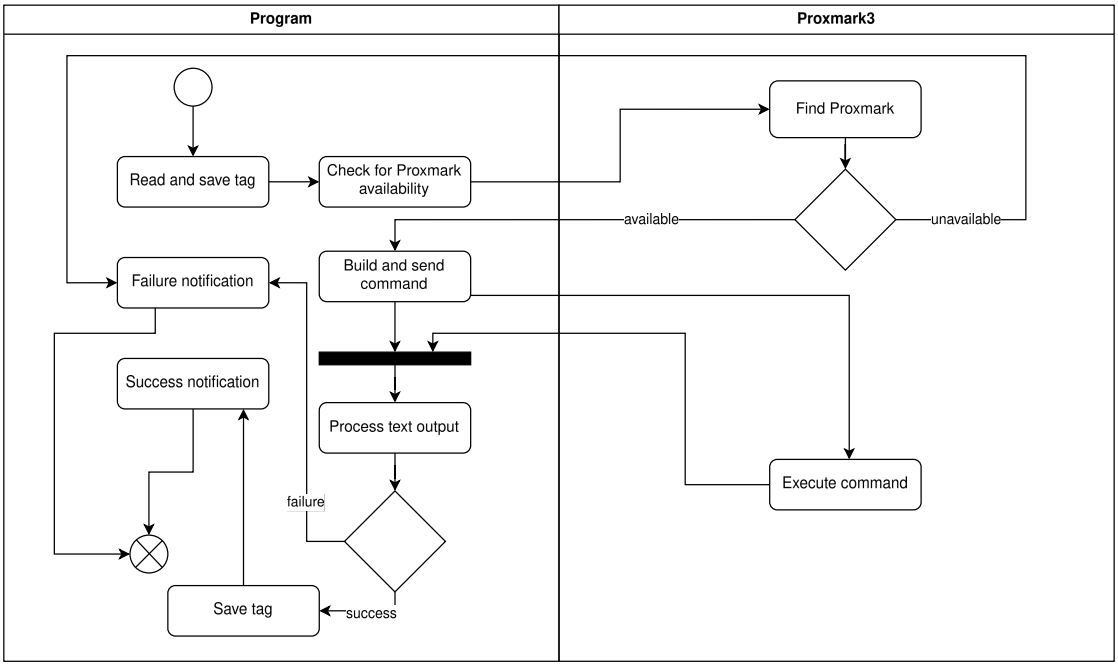

Figure 5.2 Simplified activity diagram for reading and saving tag information.

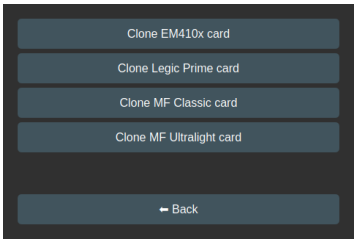

Figure 5.3 Resulting GUI 1.

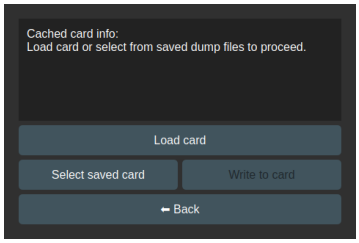

Figure 5.4 Resulting GUI 2.

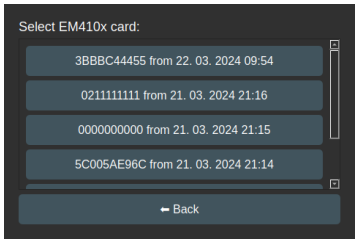

Figure 5.5 Resulting GUI 3.

5.6.3.1 Feedback to the User

For a better and more intuitive feeling of use, graphical elements had to be implemented to give the user a kind of feedback that conveys information about, for example, whether the entered command was executed correctly or not, or whether an operation is still in progress.

One of these elements was the blocking of buttons that for some reason should not yet be pressable, e.g. when UIDs are not entered when changing UIDs.

For better feedback to the user, notification toasts were used as a method of user notification. The existing `pyqt-toast` package available here [61] was used. The package license is included in the attachment.

5.6.4 User Input

For input purposes, a custom keyboard containing only the characters 0–9 and A–F was created to edit the UID tag. This option was chosen because it is simpler and more practical than a pop-up keyboard that would already be part of the operating system. The main reason was also to limit input, since the user does not need to and should not be able to enter characters other than those listed above anyway. Another reason for using a custom keyboard was also from a

visual point of view. A screenshot of the created keyboard can be seen in Figure 5.6.

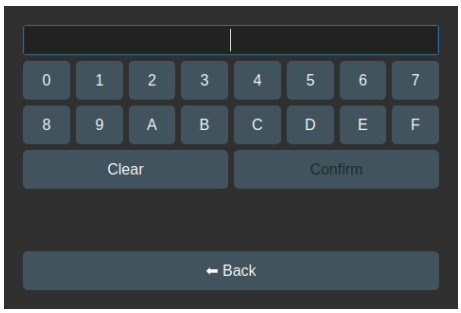

Figure 5.6 Screenshot of the created keyboard.

5.6.5 Parsing of the Output

After each command is executed, its output is processed. Since almost every command has different outputs, it was necessary to test all situations, how each command reacts for example to incorrect input or in what form important information appears in the text.

This is handled by the aforementioned `OutputHandler` class. An example of how the handling of the output of one of the commands, specifically the `hf mf autopwn` command, is handled can be seen in Code Listing A. The outputs of other commands were handled in a similar way — that is, always looking for a particular string in the output. The extraction of other information, here specifically file names from the output, can also be seen in the sample.

```
def handle_autopwn_mf_card(self):
        dump_file = self.extract_filename_from_backticks("bytes to
            binary file")
        keys_file = self.extract_filename_from_backticks("Found keys
            have been")
        if self.contains_sentence("No tag detected"):
            return constants.NO_CARD_FOUND
        if dump_file is None or keys_file is None:
            return constants.PROBLEM
        self.output = (keys_file, dump_file)
        return constants.OUTPUT_HANDLED
```

Code listing 5.5 Processing output of a command in Python.

Chapter 6
Device Testing

This chapter describes the process of testing of the functionality of the device. Testing refers to the robustness of the device itself and testing the functionality of cloning and tag emulation which took place in two layers. In the first layer, the capabilities of Proxmark were tested — how it can read cards, what information it gives out about the card, how it can detect the keys of different types of Mifare Classic tags and what error messages it gives. This layer builds seamlessly on layer two, which has tested the correctness of the program itself, whether it communicates correctly with Proxmark, and whether it can successfully respond to Proxmark's output, either to notifications of successful operations or to error messages.

6.1 User Notes

This section presents all the findings and shortcomings that emerged from the testing of the resulting device.

6.1.1 Battery Life

After testing the endurance of the device when powered from a 10,000 mAh power bank, it was found that an idle device, i.e. a switched on Raspberry Pi, with the LCD screen and Proxmark connected, lasts up to 8 hours. This result is very encouraging.

6.1.2 Boot Time

It is essential to mention that one problem starts at boot time. Once the device is plugged into the source, it takes at least about 20 seconds for the device to boot and be usable together with the program created. This is one of the impractical things that has resulted from using this device.

6.1.3 Touch Screen

The use of the touch screen has also shown that it has its shortcomings. Its small refresh rate reduces user-friendliness, and some touches are not detected once in a while. However, these shortcomings do not limit the functionality of the device in any way.

6.1.4 Graphical User Interface

After testing the GUI of the created program, many things were tweaked or corrected.

One of the things that had to be solved was the unpredictable freezing screen when executing a command. This was solved by creating a `Worker` class that inherits from `QObject`. This Worker then has the task of managing the created thread that executes the command. The actual implementation can be seen in the code. This method fixes the frozen screen.

Further debugging of the program concerned error messages, so that they are displayed at the right time and disappear appropriately, e.g. when the screen changes.

Another thing that was addressed was making user interaction impossible when a command was in progress — e.g., disabling buttons.

6.2 Tested Tags

For testing purposes, tags of different types were taken, mostly magic tags. Most of the tags were purchased from China, because of their low price and availability — there are only a few resellers in our country, who increase the price of these tags. Some of the tags for testing were acquired elsewhere. Here is a list of the specific types of tags that were available for testing the resulting device, most of them are at least two of each, to test cloning, and to be safe, as the tag may get damaged:

- Low frequency
  - EM410x
  - EM4x05
- High frequency
  - Mifare Classic magic tags
    - Mifare Classic 1k, 4-byte UID, GEN-2
    - Mifare Classic 1k, 4-byte UID, GEN-1a/GEN-4
    - Mifare Classic 4k, 7-byte UID, GEN-3
  - Mifare Classic non-magic tags
    - Mifare Classic 1k, 4-byte UID
    - Mifare Classic 1k, 7-byte UID
  - Mifare Ultralight EV1, non-magic tag
  - Mifare Ultralight EV1, GEN-2 magic tags
  - Mifare Desfire, generation 1, 2 or 3
  - Legic Prime

Some of these named tags are used for some purpose such as building entrances.

6.3 Tag Search Testing

The program has basic tag identification functions. It can list basic tag information such as UID, magic capabilities or chipset type. This output has been filtered from Proxmark to keep it free of irrelevant information and otherwise more or less preserved. Only 2 scenarios were sufficient to test the functionality, namely when the tag is not near the reader at all — the program identifies no tag found, and the case when the tag is successfully read. These two scenarios were tested for both LF and HF tags separately.

6.4 Tags Cloning Testing

Testing of tag cloning was done separately for each tag type. It should be noted that tag cloning means transferring the data content of one tag to another while preserving the original manufacturer data — i.e., for example, the tag UID. Changing the UID is tested later in this chapter.

6.4.1 Mifare Classic

Cloning Mifare Classic tags is one of the most complex cloning used in this work, but still quite straightforward.

First, it was tested if Proxmark could retrieve the keys of all the tested Mifare Classic tags. With only slight difficulties, all keys were obtained — even the keys of the non-magic tags. Difficulties encountered can only be attributed to the quality of the data transfer between the reader and the tag. Sometimes other tags near the reader interfered with the communication, sometimes it was enough to move the tag to a better position on the HF tag reader.

Testing was carried out on the available Mifare Classic magic tags and the two classic Mifare Classic tags that are used in real-world applications, for example entrance tags. One of the tags is used to enter an unnamed gym. Reading and searching for keys is done a little differently on each tag, depending on which Mifare Classic vulnerability Proxmark is currently exploiting. Most of the keys were read in about 20 seconds, with one exception which was read in about 60 seconds.

With all the keys retrieved, the tag dumps were also saved. In order to write this data to another tag, the keys of the destination tag must also be saved. Only then can the Mifare Classic tag be written.

Writing all sectors to a Mifare Classic 1k tag takes around 15 seconds, while writing to a Mifare Classic 4k tag sometimes takes up to one minute. The process of cloning this type of tag is the most complicated, also because of the duration.

Therefore, when testing Mifare Classic, I did not come into contact with a tag whose keys could not be obtained in this way. In these cases, the resulting device, in its current state, would not be able to clone the Mifare Classic cards. Verification that the write to the target tag was successful was done by comparing the dumpfiles.

6.4.1.1 Real-World Example

The keys of the aforementioned tag used to enter the gym were successfully retrieved by the device and so were the contents of the card. Unfortunately, this card has a 7 byte UID and is of Mifare Classic 1k type and the only available 7 byte magic tag was Mifare Classic 4k type, the other magic tags only had 4 byte UIDs. However, thanks to backwards compatibility, it was possible to copy the contents of the original tag into the first part of the memory and change the UID accordingly. As a result, there was a tag with the same UID and the same data blocks in sectors 0 to 15. After trying to access the gym with the newly cloned tag, it was found that the access was granted and the system therefore did not check the size of the Mifare Classic.

6.4.2 Mifare Ultralight

Unlike the Mifare Classic, the Mifare Ultralight tag is very straightforward to read. When the device is reading the simplest type of Mifare Ultralight, it reads all pages. However, when the type is Mifare Ultralight EV1, the device reads only the unprotected pages, see Chapter 2. This was verified by testing the reading of Ultralight tags. In a moment the card is read, the data dump is ready to move to another tag. Writing to the target tag went always smoothly, the success of the operation was verified by reading the contents of the written tag.

6.4.3 Legic Prime

Unfortunately, the cloning of the Legic Prime tag has not been tested as it should. Since there was only one tag available, it was only possible to read that tag and write the same tag again. In this respect, it relied on the software in Proxmark, where it was only possible to determine the success rate of writing to that one tag and not to any other.

6.5 Tags Emulating Testing

The testing of tag emulation was of course limited because the number of different readers is limited. The readers that were available were only those that I had already used myself to enter buildings or for identification, and thus no unauthorized access was gained in any case. Additionally, another Proxmark helped with testing the emulation, which always read what the device was emulating. The emulated data could be used to determine the success rate.

6.5.1 Two Types of Emulation

If the user is unable to get the entire content of the tag, the Proxmark software allows emulation only of the UID (and its associated manufacturer data) of these types of tags:

- EM410x,
- Mifare Classic,
- Mifare DESFire, and
- Mifare Ultralight.

The device therefore pretends to the reader as the tag type with the set UID. Of course, any additional data in the tag is incorrect and irrelevant.

Of course, many systems require the tag to contain other inaccessible data that is considered secure. In order to emulate these tags, the device must first successfully read the contents of the tag and, in the case of Mifare Classic, its keys. In this paper, the focus is on emulating the data content of these types of tags:

- Mifare Classic,
- Mifare Ultralight, and
- Legic Prime.

6.5.2 EM410x

When trying to emulate EM410x tags, there was a problem that the readers could not detect the emulation. A bug in the developed software can be ruled out thanks to the logs generated by the program. It can therefore be attributed to either the Proxmark3 software controlling Proxmark 3 Easy or it may be a hardware problem. A hardware problem is more likely, as after reviewing the Proxmark community posts, people were experiencing a similar problem, and for some the emulation worked without issue [62].

After reviewing the emulation and trying to load the emulated tag with a second Proxmark, the problem was confirmed, namely that the Proxmark does not emulate the tag type EM410x at all. If the above reasoning is correct, this could be fixed by replacing the Proxmark 3 Easy with another unit in which the hardware problem does not occur. However, when the Proxmark

of the resulting device was replaced with the second one, the emulation was also unsuccessful. Therefore, it is possible that both of the tested Proxmarks have this defect, but another error cannot be ruled out as well.

6.5.3 Mifare Classic

As mentioned in Subsection 6.5.1, the Proxmark software supports emulation of either the UID itself or emulation of the entire tag including sectors. For Mifare Classic, Proxmark supports emulation of the UID only for the types Mifare Classic 1k, Mifare Classic 4k. Emulation of only the UID was successful — the output of the second Proxmark, that read the emulated data can be seen in Figure 6.1

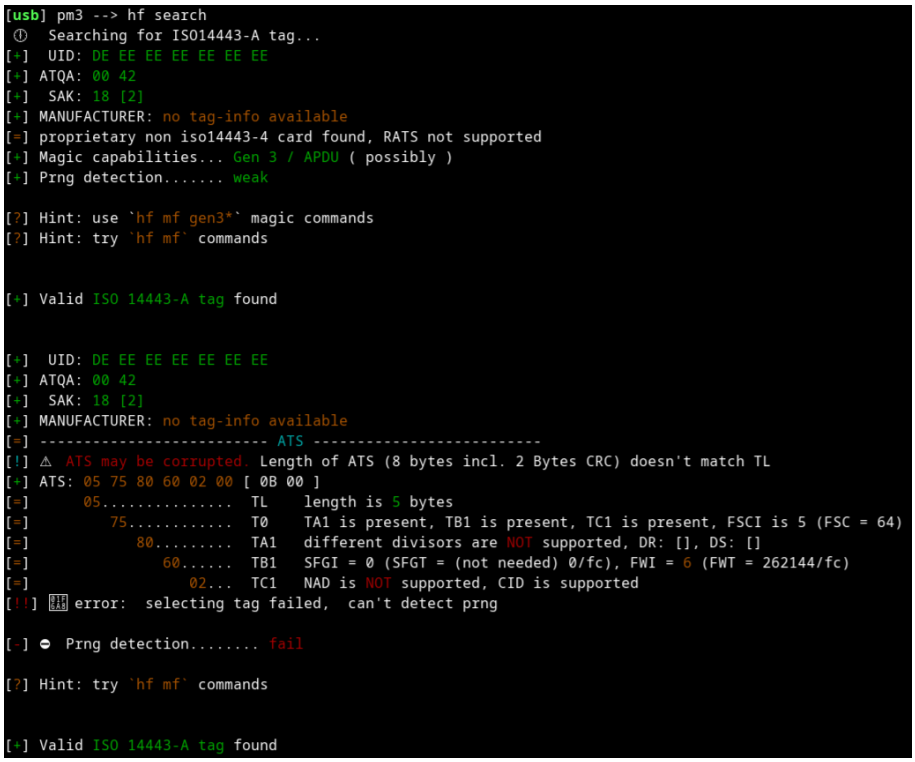

Figure 6.1 Output of the second Proxmark when reading the emulated UID of Mifare Classic 4k.

Emulation of all sectors of the tag is supported by Proxmark for types

- Mifare Classic 1k,
- Mifare Classic 4k,
- Mifare Classic 2k,
- Mifare Classic Mini.

For proper emulation it was first necessary to read a Mifare Classic tag and save a dump of its memory. Testing the reading and cracking of Mifare Classic is described in Subsection 6.4.1. The emulation was tested with another Proxmark that read the emulated tag to verify correctness. A photo of this process can be seen in Figure 6.2 and a screenshot of how the second Proxmark read the emulated data can be seen in Figure 6.3.

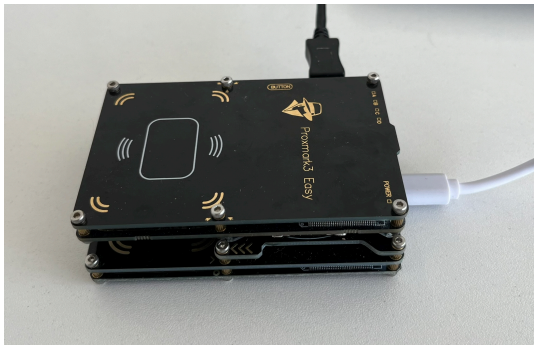

Figure 6.2 Two Proxmarks on top of each other — emulation testing.

```
[=]       |  45 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  46 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  47 | FF FF FF FF FF FF 00 00 00 00 FF FF FF FF FF FF | ................
[=]    12 |  48 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  49 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  50 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  51 | FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF | .........i......
[=]    13 |  52 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  53 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  54 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  55 | FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF | .........i......
[=]    14 |  56 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  57 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  58 | 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 | ................
[=]       |  59 | FF FF FF FF FF FF 00 00 00 00 FF FF FF FF FF FF | ................
[=]    15 |  60 | 50 4F 4B 45 77 6F 6B 65 4C 4F 56 45 64 6F 70 65 | POKEwokeLOVEdope
[=]       |  61 | 68 6F 6D 65 42 4F 4E 45 64 6F 6D 65 48 4F 50 45 | homeBONEdomeHOPE
[=]       |  62 | 50 4F 4B 45 77 6F 6B 65 4C 4F 56 45 64 6F 70 65 | POKEwokeLOVEdope
[=]       |  63 | FF FF FF FF FF FF FF 07 80 69 FF FF FF FF FF FF | .........i......
[=] -----+-----+-------------------------------------------------+----------------
```

Figure 6.3 Reading emulated blocks of Mifare Classic 1k.

Unfortunately, I did not have the opportunity to test the emulation of the Mifare Classic tag in a working system, e.g. the emulation of the tag used to enter the gym, which I managed to clone, see Subsection 6.4.1.

6.5.4 Mifare Ultralight

Mifare Ultralight tags can be emulated by Proxmark — pages that have been read. There is again the possibility of emulating just the UID with undefined data pages. Both were tested with a second Proxmark, where a problem was identified. The Proxmark correctly emulated the entered UID of the tag, however, the ATQA and SAK values were wrong. The original Mifare Ultralight tag has ATQA value `0044` and SAK `00`. As seen in Figure 6.4, the device emulated a tag with ATQA value `0344` and SAK `00`.

```
 Searching for ISO14443-A tag...
[+]  UID: 99 99 99 99 99 99 99
[+] ATQA: 03 44
[+]  SAK: 00 [2]
[+] MANUFACTURER: no tag-info available
[+]    Possible AZTEK (iso14443a compliant)
[=] proprietary non iso14443-4 card found, RATS not supported
```

Figure 6.4 Wrong ATQA value while emulating Mifare Ultralight.

After loading a whole Mifare Ultralight tag into emulator's memory, another problem arose.

It was not possible to read the blocks of data of the emulated tag. Unfortunately, I did not come into contact with a real-world use of the Mifare Ultralight tag during the testing, so I do not provide a real-world example. It is not a significant drawback that the emulation of Mifare Ultralight does not work as expected, as there was no problem with Mifare Ultralight cloning.

6.5.5 Mifare Desfire

For Mifare DESfire type tags, Proxmark allows emulation of the tag UID only, and this is of course because the card is not known to be attacked and is considered secure [14].

Testing was therefore conducted again with the second Proxmark as the reader that could read the UID correctly with appropriate ATQA, SAK and ATS values, see Figure 6.5.

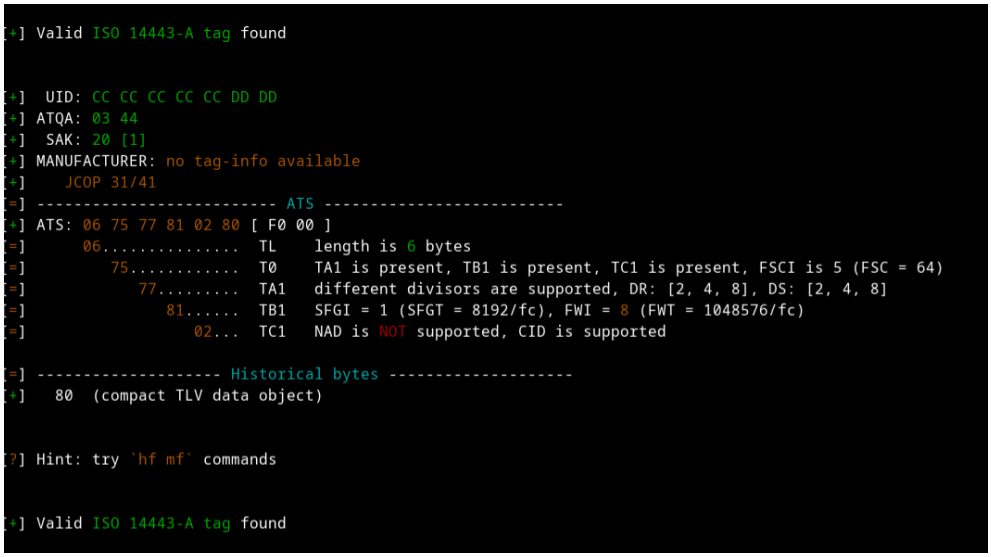

Figure 6.5 Emulated Mifare DESFire, read by second Proxmark.

6.5.5.1 Real-World Example

To test the Mifare DESfire tag emulation, I had two tags used to access an unnamed organization and an unnamed school building — this tag is used, among other things, to pay for lunches in the school cafeteria and to enter the dormitory.

For the unnamed organization, simply emulating the 7 byte UID DESFire tag was not enough — access to the building was not granted. Conversely, for the unnamed school's building entrance, emulation of the UID tag alone passed, for the main entrance and for entrances to individual classrooms within the building, even for entry to the dorm areas. Payment using the emulated tag in the student cafeteria was not tested.

However, it is important to mention that the RFID reader by the entrance to the school dormitory could only be fooled by emulating exactly Mifare DESFire type. Some of the readers inside of the school building could be fooled only by emulating the card of type Mifare Classic, the emulation of Mifare DESFire was not successful there.

For testing, a given 7 byte UID of the school DESFire tag was written to a 7 byte Mifare Classic 4k tag. With this newly created Mifare Classic tag, access was only granted in the school building and not in the dormitory areas. This is probably because some of the readers are also checking the type of the tag — ATQA and SAK values, which were in this case different.

However, they were correctly emulated by the device before, that explains the granted access to the school dormitory for the emulated DESFire tag.

6.5.6 Legic Prime

The Legic Prime tag emulation was tested again using another Proxmark, which successfully read the emulated Legic Prime tag. A photo from the process can be seen in Figure 6.6. However, the testing was very limited by the number of available Legic Prime tags.

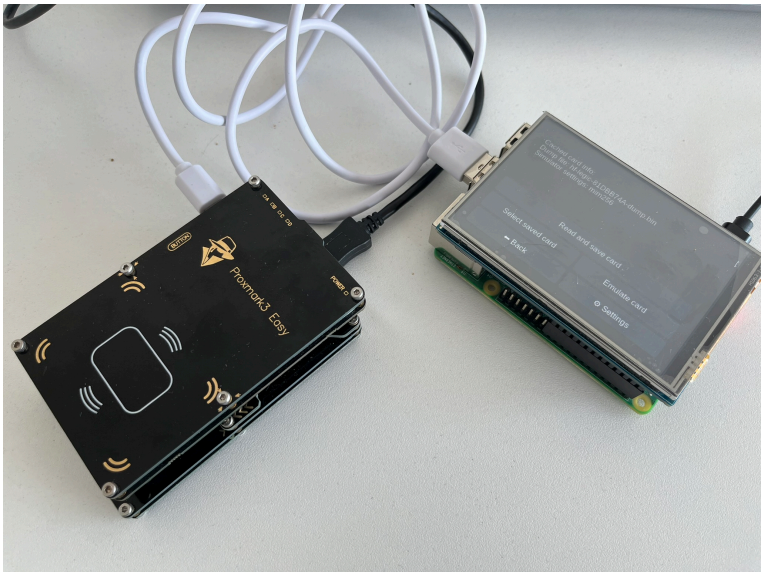

Figure 6.6 Emulating Legic Prime tag.

6.6 UID Changing Testing

Generally speaking, changing the UID of magic tags went smoothly, but it always needs to be considered e.g. which generation of magic tag it is, because these tags are controlled by different commands. A couple of times it happened that the tag was broken in response to the wrong command.

6.6.1 EM410x

Exactly with that tag, it depended on the chipset inside. If the tag UID was changed with the T5577 chipset but the encoding for Q5/T5555 was used, the tag was broken. The tag could be fixed by changing the UID and using the correct encoding.

Otherwise, changing the UID of these tags was very straightforward and there were no problems, which is due to the simple tag architecture.

6.6.1.1 Real-World Example

Since there were a lot of EM410x type LF tags available, both the magic ones that can change the UID, but also the ones that are already in use for some operational purposes, such as building entrances or identification in an unnamed canteen, testing the change of the UID was very straightforward and simple. The device was able to change UID of every given magic tag of type

EM410x. Thus, the readers in the given buildings read the content of the magic tags without any problem, granting access for those that had the UID of the original tag.

6.6.2 Mifare Classic

As seen in section 6.2, there were several types of magic Mifare Classic tags available. Each is controlled by different commands and thus there is a distinction between the different types in the device. Therefore, when changing the UID of a magic Mifare Classic tag, it is first necessary to find out what generation it is. This can be easily determined by an HF search, which will give out important information about the tag. Then the appropriate generation needs to be set in the settings. Changing the UID then goes smoothly, during multiple testing of all available tags no problems arose, except when the generation of the magic tag was set incorrectly. Verification of the UID change was done simply by reloading the tag with the device.

6.6.3 Mifare Ultralight

When testing the Mifare Ultralight tag UID change, there was only a minor issue with one of the three tags. The tag had to be positioned perfectly on the reader for the UID change to succeed. The other two tags had no problem with this. After the UID change, the operation was verified by reloading the tag.

Chapter 7
Related Work

Several additional steps are needed to improve this prototype device to make it more compact, and more user-friendly.

One of these steps is to incorporate some batteries directly into the device. While a power bank makes the device portable, the same cannot be said for compactness. To this would be added the incorporation of some sort of start button to start the device.

It is definitely worth thinking about a different and better touch screen, because with the current one there were problems related to the refresh rate or even in the actual detection of the user's touch.

The next step should be to create a device box, for example in a 3D printer. This would increase the shatter resistance of the device, the overall appearance, but also its compactness, making the manipulation easier.

It would also be beneficial to more modify the operating system running on the Raspberry Pi, mainly to reduce boot time. Another modification could be a modification to the Proxmark3 program itself, where it would be possible to disable the emulation of certain tags with a simple command rather than a button on the Proxmark.

The door is also open for creating support for more tag types that Proxmark3 itself can handle, or will be able to handle in the future.

Finally, it is definitely important to test the device properly with a larger number of tag types, to find as many bugs as possible within the device software.

Chapter 8
Conclusion

The aim of the work was to build an open-source device that would be able to clone and emulate RFID cards and tags. Firstly, designing the hardware and software architecture of a portable device, with respect to modularity and future extensibility using a commercially available microcomputer such as Raspberry Pi. Then creating a program allowing cloning and emulation of at least 3 different tags, with debugging directly in the created device. Associated with this is the development of a user interface that will allow easy control and interaction with the device. The last part of the work was to involve testing of the device with different types of RFID tags and cards and evaluating its functionality.

A prototype of a portable and scalable device was designed using a Raspberry Pi microcomputer and the RFID tool Proxmark, which allows cloning or emulation of a number of the most well-known RFID card and tag types such as Mifare Classic, Mifare Ultralight or Legic Prime. A Python program has been developed with a graphical interface that can operate the device in a simple and intuitive way.

The prototype device was carefully tested on purchased and existing tags and cards. It turned out that it can read, clone or emulate the tested cards and tags. Only two specific types of tags could not be emulated, due to a propable hardware issue of the used RFID module. However, this is compensated by the fact that the device can clone these types without significant issues.

At this stage of development, the device is functional for the most of the given types of RFID technology, but further modifications will be needed to improve some features.

Bibliography

1. PIRRONE, José; HUERTA, Mónica; ALVIZU, Rodolfo; CLOTET, Roger. Mobile Identification: NFC in the Healthcare Sector. In: 2012, pp. 39–42. Available from DOI: 10.1109/Andescon.2012.19.
2. VIOLINO, Bob. *The History of RFID Technology* [online]. RFID Journal, 2005-01-16. [visited on 2024-02-25]. Available from: https://www.rfidjournal.com/the-history-of-rfid-technology.
3. TEDJINI, Smail; VUONG, Tan-Phu; BEROULLE, Vincent. Antennas for RFID tags. 2005, pp. 19–22. ISBN 1-59593-304-2. Available from DOI: 10.1145/1107548.1107557.
4. *What is the difference between RFID and NFC* [online]. rfidcard.com, 2024. [visited on 2024-04-04]. Available from: https://www.rfidcard.com/what-is-the-different-between-rfid-and-nfc/.
5. MASYUK, M. Information security of RFID and NFC technologies. *Journal of Physics: Conference Series*. 2019, vol. 1399. Available from DOI: 10.1088/1742-6596/1399/3/033093.
6. NFC-TOOLS. *ISO/IEC 14443 Type A* [online]. 2024. [visited on 2024-05-05]. Available from: https://nfc-tools.github.io/resources/standards/iso14443A/.
7. *MIFARE type identification procedure* [online]. NXP, 2024. [visited on 2024-05-05]. Available from: https://www.nxp.com/docs/en/application-note/AN10833.pdf.
8. STEVE, Lab401. *Know your magic cards* [online]. 2019. [visited on 2024-04-03]. Available from: https://lab401.com/blogs/academy/know-your-magic-cards.
9. KRUMNIKL, Michal; MORAVEC, Pavel; OLIVKA, Petr; SEIDL, David. EM410x RFID cloned card detection system. In: *2015 International Conference on Pervasive and Embedded Computing and Communication Systems (PECCS)*. 2015, pp. 76–82.
10. PRIORITY1DESIGN. *EM4100 protocol description* [online]. 2024. [visited on 2024-04-03]. Available from: https://www.priority1design.com.au/em4100_protocol.html.
11. KOSCHER, Karl; JUELS, Ari; BRAJKOVIC, Vjekoslav; KOHNO, Tadayoshi. EPC RFID tag security weaknesses and defenses: Passport cards, enhanced drivers licenses, and beyond. In: 2009, pp. 33–42. Available from DOI: 10.1145/1653662.1653668.
12. AMAL. *Know if chip is writable* [online]. 2021. [visited on 2024-04-03]. Available from: https://forum.dangerousthings.com/t/know-if-chip-is-writable/13473/3.
13. *MF0ICU1* [online]. NXP, 2014. [visited on 2024-04-04]. Available from: https://www.nxp.com/docs/en/data-sheet/MF0ICU1.pdf.
14. PREUCIL, Tomas; NOVOTNÝ, Martin. Surveying the security of access systems in Uppsala, Sweden. In: 2023, pp. 1–5. Available from DOI: 10.1109/MECO58584.2023.10154971.

15. *MF0ULX1* [online]. NXP, 2019. [visited on 2024-04-04]. Available from: https://www.nxp.com/docs/en/data-sheet/MF0ULX1.pdf.

16. *MF1S50YYX V1* [online]. NXP, 2018. [visited on 2024-04-06]. Available from: https://www.nxp.com/docs/en/data-sheet/MF1S50YYX_V1.pdf.

17. *MF1S70YYX_V1* [online]. NXP, 2017. [visited on 2024-04-06]. Available from: https://www.nxp.com/docs/en/data-sheet/MF1S70YYX_V1.pdf.

18. GANS, Gerhard; HOEPMAN, Jaap-Henk; GARCIA, Flavio. A practical attack on the MIFARE classic. In: 2008, vol. 5189. ISBN 978-3-540-85892-8. Available from DOI: 10.1007/978-3-540-85893-5_20.

19. TEZCAN, Cihangir. Brute Force Cryptanalysis of MIFARE Classic Cards on GPU. In: 2017, pp. 524–528. Available from DOI: 10.5220/0006262705240528.

20. MEIJER, Carlo; VERDULT, Roel. Ciphertext-only Cryptanalysis on Hardened Mifare Classic Cards. In: *22nd ACM Conference on Computer and Communications Security (CCS 2015)*. ACM, 2015, pp. 18–30.

21. *MF3ICDX21_41_81_SDS* [online]. NXP, 2015. [visited on 2024-04-06]. Available from: https://www.nxp.com/docs/en/data-sheet/MF3ICDX21_41_81_SDS.pdf.

22. *Starting development with TapLinx SDK* [online]. Mifare, 2024. [visited on 2024-05-10]. Available from: https://www.mifare.net/wp-content/uploads/2020/09/AN11876.pdf.

23. HENRYK PLÖTZ, Karsten Nohl. *Legic Prime: Obscurity in Depth* [online]. 2009. [visited on 2024-04-06]. Available from: https://fahrplan.events.ccc.de/congress/2009/Fahrplan/attachments/1506_legic-slides.pdf.

24. *Legic Prime RFID cards rely on obscurity and consequently did not withstand scrutiny* [online]. Security Research Labs, 2010. [visited on 2024-04-06]. Available from: https://www.srlabs.de/blog-post/analyzing-legic-prime-rfids.

25. *Bad USB — Flipper Zero* [online]. Flipper, 2024. [visited on 2024-03-19]. Available from: https://docs.flipper.net/bad-usb.

26. *Flipper Zero Documentation* [online]. Flipper, 2024. [visited on 2024-03-19]. Available from: https://docs.flipper.net/.

27. *Reading NFC cards — Flipper Zero* [online]. Flipper, 2024. [visited on 2024-03-19]. Available from: https://docs.flipper.net/nfc/read.

28. *Flipper Zero firmware source code* [online]. Flipper, 2024. [visited on 2024-03-19]. Available from: https://github.com/flipperdevices/flipperzero-firmware.

29. *Power — Flipper Zero* [online]. Flipper, 2024. [visited on 2024-03-19]. Available from: https://docs.flipper.net/basics/power.

30. *Flipper Zero Shop* [online]. Flipper, 2024. [visited on 2024-03-19]. Available from: https://shop.flipperzero.one/.

31. EMSEC. *ChameleonMini GitHub Wiki* [online]. 2024. [visited on 2024-03-20]. Available from: https://github.com/emsec/ChameleonMini/wiki.

32. *Proxgrind ChameleonMini RevG* [online]. lab401, 2024. [visited on 2024-03-20]. Available from: https://lab401.com/products/proxgrind-chameleon-mini-revg?variant=31284469039215.

33. *Chameleon Ultra* [online]. Lab401, 2024. [visited on 2024-03-28]. Available from: https://lab401.com/products/chameleon-ultra.

34. *PN532/C1, Near Field Communication (NFC) controller* [online]. lab401, 2017. [visited on 2024-03-20]. Available from: https://www.nxp.com/docs/en/nxp/data-sheets/PN532_C1.pdf.

35. NFC-TOOLS. *MFOC* [online]. 2024. [visited on 2024-03-28]. Available from: https://github.com/nfc-tools/mfoc.

36. NFC-TOOLS. *MFCUK* [online]. 2024. [visited on 2024-03-28]. Available from: https://github.com/nfc-tools/mfcuk.

37. JUMPYCALM. *PN532 Cloner* [online]. 2024. [visited on 2024-03-28]. Available from: https://github.com/jumpycalm/pn532-cloner.

38. *PN532 NFC RFID V3* [online]. Neven, 2024. [visited on 2024-03-20]. Available from: https://www.neven.cz/kategorie/elektronicke-soucastky/nfc-a-rfid/ctecky/pn532-nfc-rfid-v3-modul/.

39. *Proxmark web page* [online]. Proxmark, 2024. [visited on 2024-03-22]. Available from: https://proxmark.com/.

40. PROXMARK. *Proxmark3 Wiki* [online]. 2024. [visited on 2024-03-22]. Available from: https://github.com/Proxmark/proxmark3/wiki.

41. RFIDRESEARCHGROUP. *Proxmark3 — How to build* [online]. 2024. [visited on 2024-03-22]. Available from: https://github.com/RfidResearchGroup/proxmark3?tab=readme-ov-file%5C#how-to-build.

42. PROXMARK. *Proxmark3 wiki — Commands* [online]. 2024. [visited on 2024-03-22]. Available from: https://github.com/Proxmark/proxmark3/wiki/commands.

43. RFIDRESEARCHGROUP. *Proxmark3 — Standalone mode* [online]. 2024. [visited on 2024-03-22]. Available from: https://github.com/RfidResearchGroup/proxmark3/wiki/Standalone-mode.

44. *Proxmark3 Easy 512kbit* [online]. Neven, 2024. [visited on 2024-03-22]. Available from: https://www.neven.cz/kategorie/elektronicke-soucastky/nfc-a-rfid/ctecky/proxmark3-easy-512kbit/.

45. *Wiring* [online]. Wiring, 2024. [visited on 2024-03-26]. Available from: https://wiring.org.co/.

46. *What is Arduino?* [online]. Arduino, 2024. [visited on 2024-03-26]. Available from: https://www.arduino.cc/en/Guide/Introduction.

47. *Arduino Hardware* [online]. Arduino, 2024. [visited on 2024-03-26]. Available from: https://www.arduino.cc/en/hardware.

48. *Raspberry Pi — About* [online]. Raspberry Pi, 2024. [visited on 2024-03-27]. Available from: https://www.raspberrypi.com/about/.

49. *Raspberry Pi 4 model B* [online]. Raspberry Pi, 2024. [visited on 2024-03-27]. Available from: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/.

50. *Proxmark3 V3.0 DEV Easy Kits Manual* [online]. Emartee, 2024. [visited on 2024-03-28]. Available from: https://www.emartee.com/Attachment.php?name=42234.pdf.

51. *Mega 2560 Rev3 — Tech Specs* [online]. Arduino, 2024. [visited on 2024-03-28]. Available from: https://docs.arduino.cc/hardware/mega-2560/%5C#tech-specs.

52. *Raspberry Pi Documentation* [online]. Raspberry Pi, 2024. [visited on 2024-03-28]. Available from: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.

53. *3.5inch RPi LCD (A) — Waveshare Wiki* [online]. Waveshare, 2024. [visited on 2024-04-06]. Available from: https://www.waveshare.com/wiki/3.5inch_RPi_LCD_(A).

54. *Raspberry Pi — Software* [online]. Raspberry Pi, 2024. [visited on 2024-04-07]. Available from: https://www.raspberrypi.com/software/.

55. THOMPSON, Phil. *PyQt5* [online]. 2024. [visited on 2024-04-07]. Available from: https://pypi.org/project/PyQt5/.

56. *Kivy: Cross-platform Python Framework for NUI* [online]. Kivy, 2024. [visited on 2024-04-07]. Available from: https://kivy.org/.

57. *tkinter — Python interface to Tcl/Tk* [online]. Python, 2024. [visited on 2024-04-07]. Available from: https://docs.python.org/3/library/tkinter.html.

58. HERRMANN, C.; TEUWEN, P.; MOISEENKO, O.; WALKER, M., et al. *Proxmark3 – Iceman repo* [https://github.com/RfidResearchGroup/proxmark3]. [N.d.].

59. RFIDRESEARCHGROUP. *Proxmark3 — doc* [online]. 2024. [visited on 2024-04-07]. Available from: https://github.com/RfidResearchGroup/proxmark3/tree/master/doc.

60. *subprocess — Subprocess management* [online]. Python, 2024. [visited on 2024-04-08]. Available from: https://docs.python.org/3/library/subprocess.html.

61. YJG30737. *PyQt Toast* [online]. 2024. [visited on 2024-04-10]. Available from: https://github.com/yjg30737/pyqt-toast.

62. FONTENO. *Em410x emulation not working* [online]. 2024. [visited on 2024-05-10]. Available from: https://github.com/RfidResearchGroup/proxmark3/issues/2197.

Attachments

- `README.md` ..... a brief description of the content of the medium
- `impl` ..... implemented source codes
  - `README.md` ..... instructions for setting up the program
  - `src` ..... the python source code
- `text` ..... the text of the thesis
  - `thesis.pdf` ..... text of the thesis in PDF format
  - `thesis` ..... source form of the work in LaTeX format